臺北市立大安高級工業學校

迷你數位示波器

Mini Digital Oscilloscope

組長: 電子三甲 林峻彣

組員: 電子三甲 陳致維

電子三甲 簡廷諭

電子三甲 顏冠穎

指導老師: 楊仁元 老師

摘要

示波器是一種用途十分廣泛的電子測量儀器。它能把肉眼看不見的電信號變換成看得見的圖像，利用示波器還能觀察各種不同信號幅度隨時間變化的波形曲線，還可以用它測量各種不同的電壓、電流、頻率、相位差、調幅度等等。

Abstract

Oscilloscope is a very extensive electronic measuring instrument. It can convert invisible electrical signals into visible images. Oscil loscope can also be used to observe the waveform curves of various signal amplitudes changing with time, and it can also be used to test various voltages, currents, frequencies, phase differences, amplitude modulation or something else.

目錄

圖目錄

表目錄

第一章 緒論

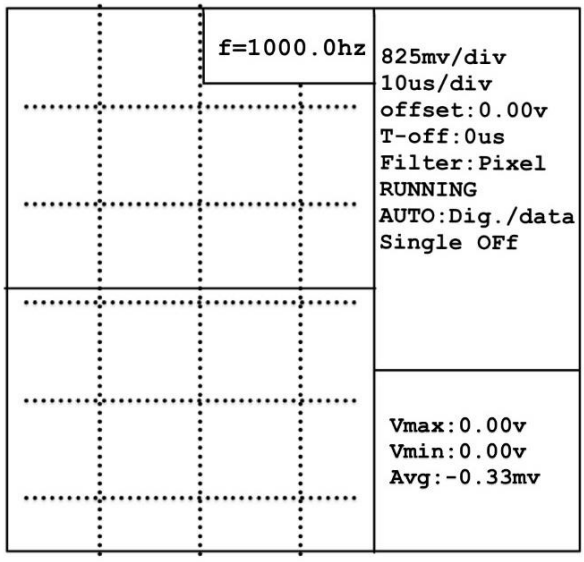
1-1 背景及目的

我們做示波器的原因是它對我們學電學的學生是一個不可缺少的存在，像製作各種電路時會用來判斷 Vi 與 Vo 的相位差，目前用到的只是一小部分，未來會用到它的時間還遙遙無期。但是示波器的重量是很難讓我們可以隨身攜帶的，所以我們為了解決這個困擾，我們想做出一台迷你示波器。

1-2 預期成果

完成所有步驟號後，我們預期它有可以測量各項數據，包含電路之電壓，擁有一般示波器的選單功能(如圖1-1)，並且能選擇要觀測的數據。

▼ 圖1-1 迷你數位示波器選單示意圖



第二章 理論探討

2-1 ESP32-WROOM (DEVKIT V1)

ESP32-WROOM DEVKIT V1(如圖2-1)使用 Xtensa 雙核心32位元的 CPU，並採用LX6的微處理器，工作時脈有160MHz 和240MHz 兩種模式，運算能力可達到600MIPS。而我們專題使用的是中國上海樂鑫資訊科技開發，台灣的台積電製造，並採用40奈米技術。是 ESP8266的後繼產品，具有低成本和低功耗的特性。

▼ 圖2-1 ESP32-WROOM DEVKIT V1

開發版外型

34個 GPIO

12-bit SAR ADC（18個通道）

2個 8位元 D/A 轉換器

10個觸控感應器

4個 SPI (Serial Peripheral Interface Bus)

2個 I2S(Integrated Interchip Sound)

2個 I2C(Inter-Integrated Circuit)

3個 UART(Universal-Asynchronous-Receiver Transmitter)

2-2 TFT ILI9486

TFT ILI9486(如圖2-2)為4.0寸彩色螢幕，支持 RGB 65K 顯示，且採用480X320解析度，可選擇觸控功能。採用 SPI 串列匯流排，只需幾個 IO 即可點亮顯示，並自帶 SD 卡槽方便進行擴展實驗，和提供豐富的示列程序、軍工級的工藝標準、底層驅動技術的支援。而軍工級的工藝製成，使得 TFT ILI9486可以穩定的長期工作，為 ILITEK 所製作的產品 。

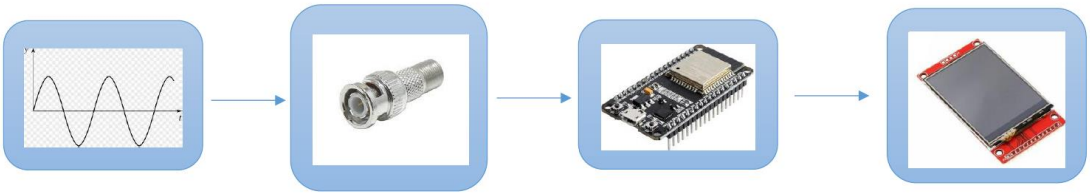
▼ 圖2-2 TFT ILI9486

第三章 專題準備

3-1 系統流程

我們的示波器系統流程維信號輸出到 BNC 接頭，再由 BNC 接頭傳給 ESP32信號處理，最後由 LCD 螢幕輸出。

圖3-1 系統流程圖



3-2 甘特圖

下圖3-2為我們的工作分配甘特圖

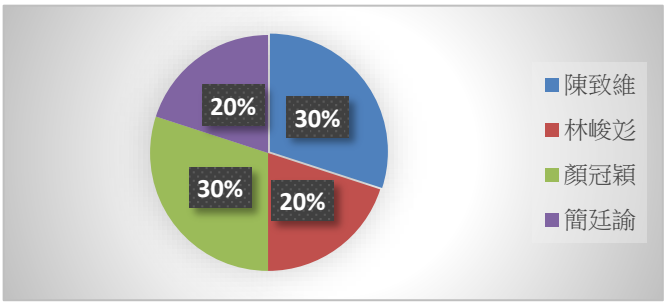
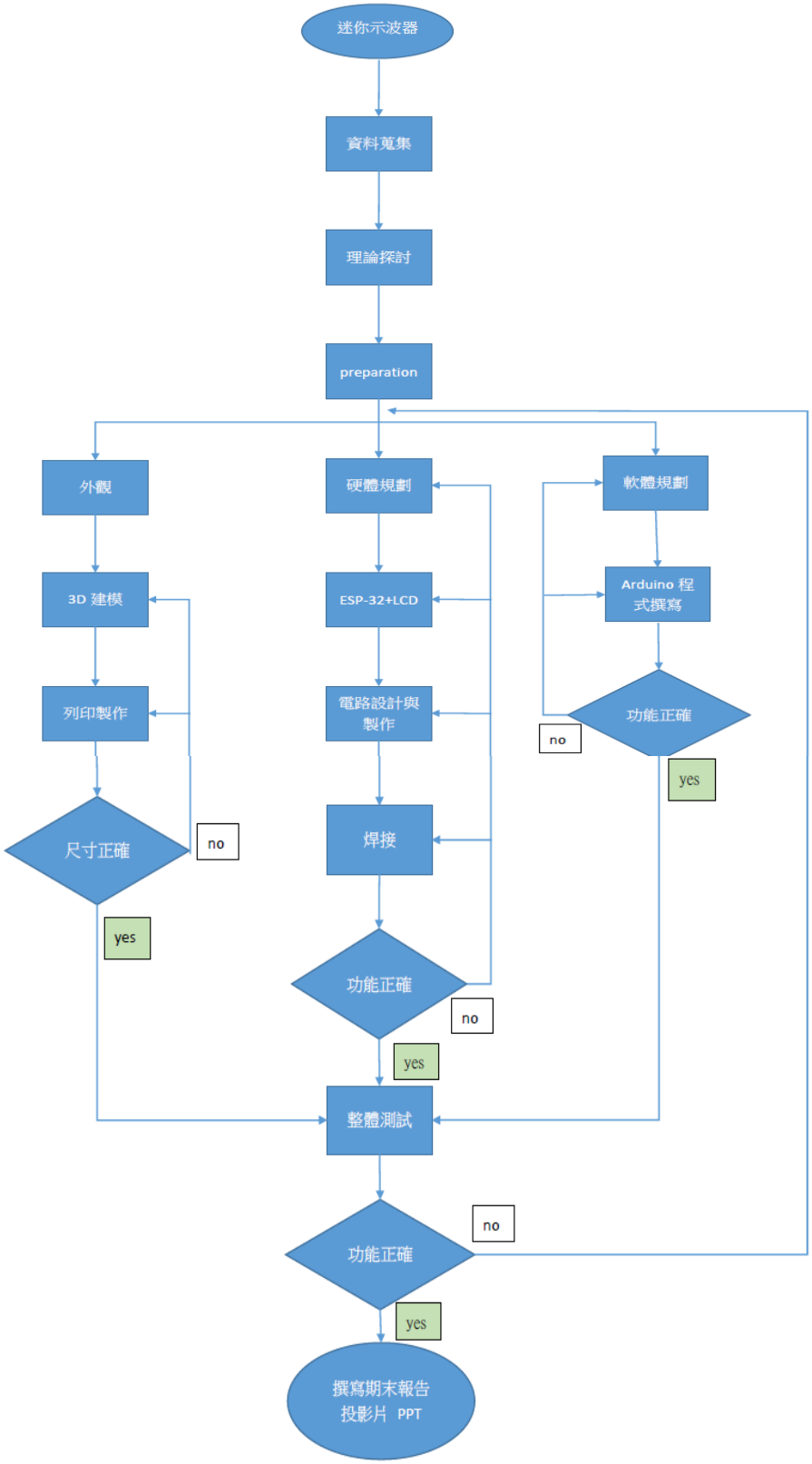


圖3-2 甘特圖

日期 / 工作項目	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	負責成員
雷射切割繪圖						■	■												林峻彣
雷射切割製作								■	■	■	■	■	■						林峻彣,陳致維
程式分析								■	■	■	■	■							陳致維,簡廷諭
電路分析						■	■	■											顏冠穎
電路製作									■	■									顏冠穎
焊接									■	■									陳致維,簡廷諭
功能測試													■	■	■	■			全組
找尋資料	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	全組
報告製作				■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	全組
預定進度	3	5	6	8	10	20	30	35	45	50	75	75	75	80	90	93	95	100	累積百分比%

3-3 流程圖

圖3-3 流程圖

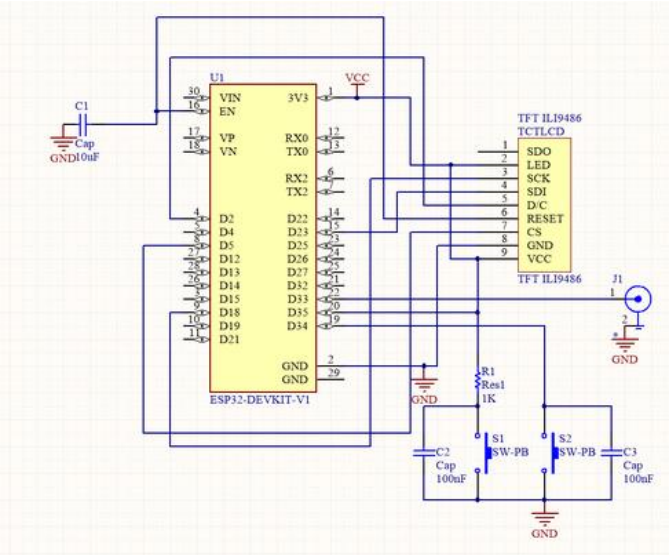


第四章 專題成果

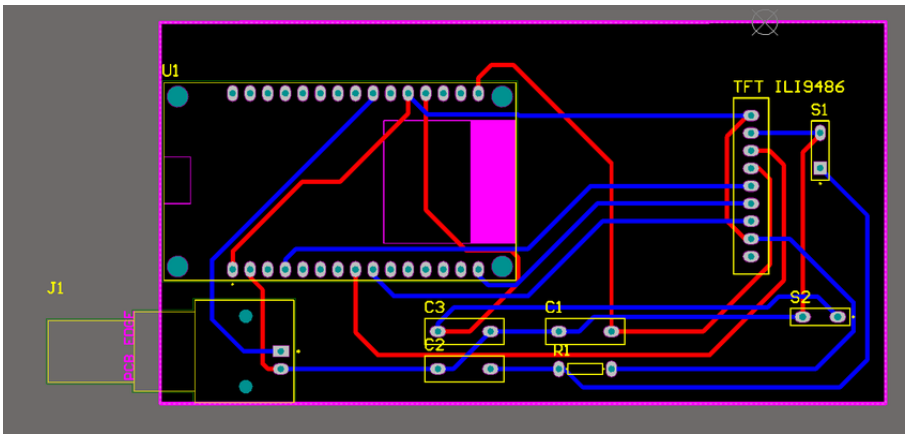
4-1電路設計

我們的電路板是用 Altium Designer 繪製而成，圖3-4為電路設計圖，圖3-5為電路接線設計，我們所考慮到的地方有 ESP32與 LCD 螢幕擺放位置，而考慮原因為避免兩者在組裝時交疊，導致無法組裝等問題。

▼ 圖4-1 電路設計圖



▼ 圖4-2 電路接線圖



4-1 電路設計

接著電路設計完成後我們將電路圖印出，並將電路板切成適當大小及進行曝光。

圖4-3 切割電路板

圖4-4 抽真空

圖4-5 對準曝光位置

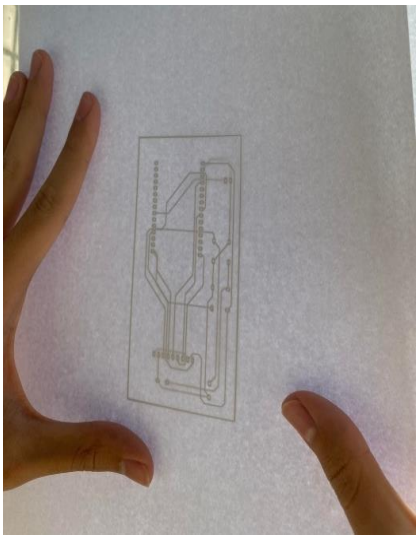


圖4-6 電路曝光

4-1 電路設計

完成曝光後，先將適溫的水與顯像劑倒入盆中攪拌均勻，再將曝光後的電路板放置在水中，經過時間的推移板子上就會出現電路的樣貌，最後將電路板放到蝕刻機內，等待約10~15分鐘，電路板送出後，就可以鑽孔與焊接了。

圖4-7 電路板顯像

圖4-8 鑽孔

圖4-9 將電路板放入蝕刻機內

圖4-10 焊接

4-2 軟體設計

主程式(I2s_adc_esp32)

```
#include <Arduino.h>
#include <driver/i2s.h>
#include <driver/adc.h>
#include <soc/syscon_reg.h>
#include <TFT_eSPI.h>
#include <SPI.h>
#include "esp_adc_cal.h"                    //#define DEBUG_SERIAL
#include "filters.h"                        //#define DEBUG_BUFF
#include <Button2.h>
#define DELAY 1000

#define WIDTH  175 //ttgo:135  ili9341:240 (voltage)
#define HEIGHT 320 //ttog:240  ili9341:320 (time)
#define ADC_CHANNEL   ADC1_CHANNEL_5
#define NUM_SAMPLES   1000
#define I2S_NUM         (0)
#define BUFF_SIZE 50000
#define B_MULT BUFF_SIZE/NUM_SAMPLES
#define BUTTON_1        35                  //選單模式按鈕
#define BUTTON_2        0                   //設定按鈕

TFT_eSPI    tft = TFT_eSPI();

TFT_eSprite spr = TFT_eSprite(&tft);


Button2 button_mode = Button2(BUTTON_1);
Button2 button_set = Button2(BUTTON_2);

esp_adc_cal_characteristics_t adc_chars;

TaskHandle_t task_menu;                     //建立多核工作名稱
TaskHandle_t task_adc;

                                            //3300/4
float v_div = 825;
float s_div = 10;
float offset = 0;
float toffset = 0;
uint8_t current_filter = 1;

enum Option {
  None,
  Autoscale,
  Vdiv,
  Sdiv,
  Offset,
```

```
  TOffset,
  Filter,
  Stop,
  Mode,
  Single,
  Clear,
  Reset,
  Probe,
  UpdateF,
  Cursor1,
  Cursor2
};

int8_t volts_index = 0;

int8_t tscale_index = 0;

uint8_t opt = None;

bool menu = false;
bool info = true;
bool set_value  = false;

float RATE = 1000;                                  //in ksps --> 1000 = 1Msps

bool auto_scale = false;

bool full_pix = true;

bool stop = false;

bool stop_change = false;

uint16_t i2s_buff[BUFF_SIZE];

bool single_trigger = false;
bool data_trigger = false;

bool updating_screen = false;
bool new_data = false;
bool menu_action = false;
uint8_t digital_wave_option = 0;                    //0-auto | 1-analog | 2-digital

void setup() {
  Serial.begin(115200);                             //預設為自動模式 auto

  configure_i2s(1000000);

  setup_screen();
```

```
  button_mode.setClickHandler(click);
  button_mode.setLongClickHandler(click_long);
  button_set.setClickHandler(click);
  button_set.setLongClickHandler(click_long);

  characterize_adc();
#ifdef DEBUG_BUF
  debug_buffer();
#endif




  xTaskCreatePinnedToCore(
    core0_task,
    "menu_handle",
    10000,
    NULL,
    0,
    &task_menu,
    0);

  xTaskCreatePinnedToCore(
    core1_task,
    "adc_handle",
    10000,
    NULL,
    3,
    &task_adc,.
    1);

}


void core0_task( void * pvParameters ) {

  (void) pvParameters;

  for (;;) {
    menu_handler();

    if (new_data || menu_action) {
      new_data = false;
      menu_action = false;

      updating_screen = true;
      update_screen(i2s_buff, RATE);
      updating_screen = false;
      vTaskDelay(pdMS_TO_TICKS(10));
      Serial.println("CORE0");
```

```
//建立工作給指定的 core
//core0:menu
//使用堆疊大小10000
// Task input parameter
// Priority of the task
// Task handle.
// Core where the task should
run


//core1:adc

// Stack size in words
// Task input parameter
// Priority of the task
// Task handle
//雙核心 core0 core1
處理不同事情


//core0執行 menu 副程式
```

```
   }

   vTaskDelay(pdMS_TO_TICKS(10));
  }

}

void core1_task( void * pvParameters ) {

  (void) pvParameters;

  for (;;) {
   if (!single_trigger) {
    while (updating_screen) {
     vTaskDelay(pdMS_TO_TICKS(1));
    }
    if (!stop) {
     if (stop_change) {
      i2s_adc_enable(I2S_NUM_0);
      stop_change = false;
     }
     ADC_Sampling(i2s_buff);
     new_data = true;
    }
    else {
     if (!stop_change) {
      i2s_adc_disable(I2S_NUM_0);
      i2s_zero_dma_buffer(I2S_NUM_0);
      stop_change = true;
     }
    }
    Serial.println("CORE1");
    vTaskDelay(pdMS_TO_TICKS(300));
   }


   else {
    float old_mean = 0;
    while (single_trigger) {
     stop = true;
     ADC_Sampling(i2s_buff);
     float mean = 0;
     float max_v, min_v;
     peak_mean(i2s_buff, BUFF_SIZE, &max_v,
&min_v, &mean);




     if ((old_mean != 0 && fabs(mean - old_mean) >
```

//core1執行 adc

//單次觸發=0，重複執行持續讀取輸入波形

//單次觸發=1，只執行一次

```
0.2) || to_voltage(max_v) - to_voltage(min_v) > 0.05)
{
        float freq = 0;
        float period = 0;
        uint32_t trigger0 = 0;
        uint32_t trigger1 = 0;



        bool digital_data = !false;
        if (digital_wave_option == 1) {
          trigger_freq_analog(i2s_buff, RATE, mean,
max_v, min_v, &freq, &period, &trigger0, &trigger1);
        }
        else if (digital_wave_option == 0) {
          digital_data = digital_analog(i2s_buff, max_v,
min_v);
          if (!digital_data) {
            trigger_freq_analog(i2s_buff, RATE, mean,
max_v, min_v, &freq, &period, &trigger0, &trigger1);
          }
          else {
            trigger_freq_digital(i2s_buff, RATE, mean,
max_v, min_v, &freq, &period, &trigger0);
          }
        }
        else {
          trigger_freq_digital(i2s_buff, RATE, mean,
max_v, min_v, &freq, &period, &trigger0);
        }

        single_trigger = false;
        new_data = true;
        Serial.println("Single GOT");

       }

       vTaskDelay(pdMS_TO_TICKS(1));

     }
     vTaskDelay(pdMS_TO_TICKS(300));
   }
  }
}

void loop() {}
```

//如果類比模式或自動模式為類比(數位以外為類比)
//option=1為 analog

//option=0為 analog

//option=2為 digital

副程式（screen）

```
void setup_screen() {
  tft.init();
  tft.setRotation(3); //0:0 degree,1:90 degree,2:180 degree,3:270 degree




  spr.createSprite(HEIGHT, WIDTH);
  tft.fillScreen(TFT_BLACK);
}

int data[HEIGHT] = {0};

int step_WIDTH=WIDTH/4;
float to_scale(float reading) {
  float temp = WIDTH -
               (
                 (
                   (
                     (float)((reading - 20480.0) / 4095.0)
                     + (offset / 3.3)
                   )
                   * 3300 /
                   (v_div * 4)
                 )
               )
               * (WIDTH - 1)
               - 1;
  return temp;
}

float to_voltage(float reading) {
  return  (reading - 20480.0) / 4095.0 * 3.3;
}

uint32_t from_voltage(float voltage) {
  return uint32_t(voltage / 3.3 * 4095 + 20480.0);
}
void update_screen(uint16_t *i2s_buff, float sample_rate) {

  float mean = 0;
  float max_v, min_v;

  peak_mean(i2s_buff, BUFF_SIZE, &max_v, &min_v, &mean);
```

```
// Initialise the TFT registers

//螢幕旋轉90度

// 設定長寬
// Clear the TFT screen to blue
//初始化螢幕

//set Voltage division 4
//調整電壓在示波器上顯示格數

//設定在螢幕顯示像素

//第一個陣列顯示數值與名稱
```

```
  float freq = 0;
  float period = 0;
  uint32_t trigger0 = 0;
  uint32_t trigger1 = 0;

  bool digital_data = false;
  if (digital_wave_option == 1) {
    trigger_freq_analog(i2s_buff, sample_rate,
mean, max_v, min_v, &freq, &period, &trigger0,
&trigger1);
  }
  else if (digital_wave_option == 0) {
    digital_data = digital_analog(i2s_buff, max_v,
min_v);
    if (!digital_data) {
      trigger_freq_analog(i2s_buff, sample_rate,
mean, max_v, min_v, &freq, &period, &trigger0,
&trigger1);
    }
    else {
      trigger_freq_digital(i2s_buff, sample_rate,
mean, max_v, min_v, &freq, &period,
&trigger0);
    }
  }
  else {
    trigger_freq_digital(i2s_buff, sample_rate,
mean, max_v, min_v, &freq, &period,
&trigger0);
  }
  draw_sprite(freq, period, mean, max_v, min_v,
trigger0, sample_rate, digital_data, true);
}
void draw_sprite(float freq,                          //第二個陣列做各項數值運算
                 float period,                        及畫出波形
                 float mean,
                 float max_v,
                 float min_v,
                 uint32_t trigger,
                 float sample_rate,
                 bool digital_data,
                 bool new_data
                ) {

  max_v = to_voltage(max_v);
  min_v = to_voltage(min_v);

  String frequency = "";
  if (freq < 1000)
```

```
    frequency = String(freq) + "hz";
  else if (freq < 100000)
    frequency = String(freq / 1000) + "khz";
  else
    frequency = "----";

  String s_mean = "";
  if (mean > 1.0)
    s_mean = "Avg: " + String(mean) + "V";
  else
    s_mean = "Avg: " + String(mean * 1000.0) +
"mV";

  String str_filter = "";
  if (current_filter == 0)
    str_filter = "None";
  else if (current_filter == 1)
    str_filter = "Pixel";
  else if (current_filter == 2)
    str_filter = "Mean-5";
  else if (current_filter == 3)
    str_filter = "Lpass9";

  String str_stop = "";
  if (!stop)
    str_stop = "RUNNING";
  else
    str_stop = "STOPPED";

  String wave_option = "";
  if (digital_wave_option == 0)
    if (digital_data )
      wave_option = "AUTO:Dig./data";
    else
      wave_option = "AUTO:Analog";
  else if (digital_wave_option == 1)
    wave_option = "MODE:Analog";
  else
    wave_option = "MODE:Dig./data";

  if (new_data) {

    spr.fillSprite(TFT_BLACK);

    draw_grid();
    if (auto_scale) {
      auto_scale = false;
      v_div = 1000.0 * max_v / 4.0;
      s_div = period / 3.5;
      if (s_div > 7000 || s_div <= 0)
```

//頻率若大於1000hz 則換算成 khz

//平均值若大於1 則顯示平均值＋v；否則將平均值＊1000+mv

//stop=0顯示 running

//stop=1顯示 stopped

//自動調整螢幕波形

```
      s_div = 7000;
    if (v_div <= 0)
      v_div = 825;
  }

  if (!(digital_wave_option == 2 && trigger ==
0))
    draw_channel1(trigger, 0, i2s_buff,
sample_rate);

 }

 int shift = 150;                                    //開啟選單後，將原名稱及數值的位置做移位後，顯示選單
 if (menu) {
  spr.drawLine( 0, int(WIDTH/2), HEIGHT,
int(WIDTH/2), TFT_WHITE); //center line
  spr.fillRect(shift, 0, HEIGHT, WIDTH,
TFT_BLACK);
  spr.drawRect(shift, 0, HEIGHT, WIDTH,
TFT_WHITE);
  spr.fillRect(shift + 1, 3 + 10 * (opt - 1), 100,
11, TFT_RED);

  spr.drawString("AUTOSCALE",  shift + 5, 5);
  spr.drawString("V: " + String(int(v_div)) +
"mV/div",  shift + 5, 15);
  spr.drawString("H: " + String(int(s_div)) +
"uS/div",  shift + 5, 25);
  spr.drawString("Offset: " + String(offset) +
"V",  shift + 5, 35);
  spr.drawString("T-Off: " +
String((uint32_t)toffset) + "uS",  shift + 5, 45);
  spr.drawString("Filter: " + str_filter, shift + 5,
55);
  spr.drawString(str_stop, shift + 5, 65);
  spr.drawString(wave_option, shift + 5, 75);
  spr.drawString("Single " +
String(single_trigger ? "ON" : "OFF"), shift + 5,
85);

  spr.drawLine(shift, 103, shift + 100, 103,
TFT_WHITE);

  spr.drawString("Vmax: " + String(max_v) +
"V",  shift + 5, 105);
  spr.drawString("Vmin: " + String(min_v) +
"V",  shift + 5, 115);
  spr.drawString(s_mean,  shift + 5, 125);

  shift -= 80;
```

```
   spr.drawRect(shift, 0, 80, 30, TFT_WHITE);
   spr.drawString("Vp-p: " + String(max_v -
min_v) + "V",  shift + 5, 5);
   spr.drawString("f: " + frequency,  shift + 5,
15);
   String offset_line = String((2.0 * v_div) /
1000.0 - offset) + "V";
   spr.drawString(offset_line,  shift + 50, 75);

   if (set_value) {
     spr.fillRect(HEIGHT-11, 0, 11, 11,
TFT_BLUE);
     spr.drawRect(HEIGHT-11, 0, 11, 11,
TFT_WHITE);
     spr.drawLine(HEIGHT-9, 5, HEIGHT-2 , 5,
TFT_WHITE);
     spr.drawLine(HEIGHT-6, 2, HEIGHT-6, 8,
TFT_WHITE);
     spr.fillRect(HEIGHT-11, WIDTH-11, 11, 11,
TFT_BLUE);
     spr.drawRect(HEIGHT-11, WIDTH-11, 11,
11, TFT_WHITE);
     spr.drawLine(HEIGHT-9, WIDTH-6,
HEIGHT-2, WIDTH-6, TFT_WHITE);
   }
  }

  else if (info) {
   spr.drawLine( 0, int(WIDTH/2), HEIGHT,
int(WIDTH/2), TFT_WHITE);
   spr.drawRect(shift + 10, 0, HEIGHT - shift -
10, 30, TFT_WHITE);
   spr.drawString("Vp-p: " + String(max_v -
min_v) + "V",  shift + 15, 5);
   spr.drawString("f: " + frequency,  shift + 15,
15);
   String offset_line = String((2.0 * v_div) /
1000.0 - offset) + "V";
   spr.drawString(offset_line,  shift + 70, 75);
  }

  spr.pushSprite(0, 0);

  yield();
}

void draw_grid() {
  for (int i = 0; i < (HEIGHT/10); i++) {
    spr.drawPixel(i * 10, step_WIDTH,
```

```
//spr.fillRect(shift, 0, 70, 30,
TFT_BLACK);

//change 229 to HEIGHT-11

//change 229 to HEIGHT-11

//change 231 to HEIGHT-9 238
to HEIGHT-2
//change 234 to HEIGHT-6 234
to HEIGHT-6
//change 229 to HEIGHT-11
124 to WIDTH-11
//change 229 to HEIGHT-11
124 to WIDTH-11
//change 231 to HEIGHT-9 238
to HEIGHT-2 129 to WIDTH-6

//顯示頻率、Vp-p
//center line

//spr.drawString("taivs",  10,
170);
//push the drawed sprite to the
screen
// Stop watchdog reset

//設定示波器的格子、像素
//24 modify to HEIGHT/10
//33 modify to 60 (240/4)
```

```
TFT_WHITE);
   spr.drawPixel(i * 10, step_WIDTH*2,
TFT_WHITE);
   spr.drawPixel(i * 10, step_WIDTH*3,
TFT_WHITE);
  }
  for (int i = 0; i < WIDTH; i += 10) {
   for (int j = 0; j < HEIGHT; j += step_WIDTH)
{
    spr.drawPixel(j, i, TFT_WHITE);
   }
  }
}

void draw_channel1(uint32_t trigger0, uint32_t
trigger1, uint16_t *i2s_buff, float sample_rate) {

  data[0] = to_scale(i2s_buff[trigger0]);
  low_pass filter(0.99);
  mean_filter mfilter(5);
  mfilter.init(i2s_buff[trigger0]);
  filter._value = i2s_buff[trigger0];
  float data_per_pixel = (s_div / step_WIDTH) /
(sample_rate / 1000);



  uint32_t index_offset = (uint32_t)(toffset /
data_per_pixel);
  trigger0 += index_offset;
  uint32_t old_index = trigger0;
  float n_data = 0, o_data =
to_scale(i2s_buff[trigger0]);
  for (uint32_t i = 1; i < HEIGHT; i++) {
   uint32_t index = trigger0 + (uint32_t)((i + 1) *
data_per_pixel);
   if (index < BUFF_SIZE) {
    if (full_pix && s_div > step_WIDTH &&
current_filter == 0) {
      uint32_t max_val = i2s_buff[old_index];
      uint32_t min_val = i2s_buff[old_index];
      for (int j = old_index; j < index; j++) {

        if (i2s_buff[j] > max_val)
          max_val = i2s_buff[j];
        else if (i2s_buff[j] < min_val)
          min_val = i2s_buff[j];

      }
      spr.drawLine(i, to_scale(min_val), i,
```

```
//67 modify to 120 (240/4)

//101 modify to 180 (240/4)


//135 modify to WIDTH
//240 modify to HEIGHT,34
modify to step_WIDTH


//畫波形
//screen wave drawing


//34 modify to step_WIDTH
(240/4)
//  uint32_t cursor = (trigger1-
trigger0)/data_per_pixel;
  //  spr.drawLine(cursor, 0,
cursor, 135, TFT_RED);


//240 modify to HEIGHT


//34 modify to step_WIDTH
(240/4)


//draw lines for all this data
points on pixel i
```

```
to_scale(max_val), TFT_YELLOW);
      }
      else {
        if (current_filter == 2)
          n_data =
to_scale(mfilter.filter((float)i2s_buff[index]));
        else if (current_filter == 3)
          n_data =
to_scale(filter.filter((float)i2s_buff[index]));
        else
          n_data = to_scale(i2s_buff[index]);

        spr.drawLine(i - 1, o_data, i, n_data,
TFT_YELLOW);
        o_data = n_data;
      }

    }
    else {
      break;
    }
    old_index = index;
  }
}
```

副程式（options_handler）

```
int voltage_division[6] = {

  825,
  750,
  500,
  250,
  100,
  50
};


float time_division[9] = {
  10,
  25,
  50,
  100,
  250,
  500,
  1000,
  2500,
  5000
};
};

void menu_handler() {
  button_mode.loop();
  button_set.loop();
}

void click_long(Button2& btn) {
  menu_action = true;
  if (btn == button_mode) {
    uint32_t pressed =
btn.wasPressedFor();
    if (pressed > 1000) {
      opt = None;
      hide_menu();
      set_value = false;
    }
  }


  else {
    uint32_t pressed =
btn.wasPressedFor();
    if (pressed > 1000) {
      if (set_value) {
        set_value = false;
```

```
//screen has 4 divisions, 31 pixels each
(125 pixels of height)
//fullscreen 3.3V peak-peak
//電壓分配 設定每格的電壓值



/*each sample represents 1us (1Msps),
   thus, the time division is the number
   of samples per screen division
*/
//screen has 4 divisions, 60 pixel each
(240 pixel of width)
//時基 設定每格的時間
//,  //1Mhz    35ms of data (of 50ms
possible)
//  10000,  //100khz  70ms/500ms
//  25000,  //100khz  175ms/500ms of
data
//  50000,  //100khz  350ms/500ms of
data
//  100000  //50khz   700ms/1000ms of
data



//當按鈕2長按 顯示選單
// 如果按超過一秒 隱藏選單
```

```
    }
    else {
      switch (opt) {
        case Vdiv:
          v_div = 825;
          volts_index = 0;
          break;

        case Sdiv:
          s_div = 10;
          tscale_index = 0;
          break;

        case Offset:
          offset = 0;
          break;

        case TOffset:
          toffset = 0;
          break;

        case Filter:
          current_filter = 1;
          break;

        case None:
          info = !info;
          break;

        default:
          break;
      }
    }
  }
 }
}

void click(Button2& btn) {
 menu_action = true;

 if (set_value) {
  switch (opt) {
    case Vdiv:
      if (btn == button_set) {
        volts_index++;
        if (volts_index >=
sizeof(voltage_division) /
sizeof(*voltage_division)) {
          volts_index = 0;
        }
```

//電壓初始值

//時間初始值

//直流準位初始值

//週期初始值

//濾波方式選擇

//若每格電壓值在50時，按減號會回到825；若每格電壓值在825時，按加號會回到50

```
      }
      else {
        volts_index--;
        if (volts_index < 0) {
          volts_index =
sizeof(voltage_division) /
sizeof(*voltage_division) - 1;
        }
      }

      v_div =
voltage_division[volts_index];
      break;

    case Sdiv:
      if (btn == button_mode) {
        tscale_index++;
        if (tscale_index >=
sizeof(time_division) /
sizeof(*time_division)) {
          tscale_index = 0;
        }
      }
      else {
        tscale_index--;
        if (tscale_index < 0) {
          tscale_index =
sizeof(time_division) /
sizeof(*time_division) - 1;
        }
      }

      s_div = time_division[tscale_index];
      break;

    case Offset:
      if (btn == button_mode){
        offset += 0.1 * (v_div * 4) / 3300;
      }
      else{
        offset -= 0.1 * (v_div * 4) / 3300;
      }

      if (offset > 3.3)
        offset = 3.3;
      if (offset < -3.3)
        offset = -3.3;

      break;
```

```
//若每格時間在10時，按加號會回到5000；若每格時間在5000時，按減號會回到10

//調整波形準位的上下偏移量
```

case TOffset:
if (btn == button_mode)
toffset += 0.1 * s_div;
else
toffset -= 0.1 * s_div;
 	
break;
 	
default:
break;
}
}
else {
if (btn == button_mode) {
opt++;
if (opt > Single)
opt = None;
if (opt == None)
hide_menu();
else
show_menu();
 	
}
else if (btn == button_set) {
switch (opt) {
case Autoscale:
auto_scale = !auto_scale;
break;
 	
case Vdiv:
set_value = true;
break;
 	
case Sdiv:
set_value = true;
break;
 	
case Offset:
set_value = true;
break;
 	
case Stop:
stop = !stop;
break;
 	
case TOffset:
set_value = true;
break;
 	
case Single:
	//調整波形左右移動

```
        single_trigger = true;
        break;

      case Reset:                                   //reset 重置
        offset = 0;
        v_div = 825;
        s_div = 10;
        tscale_index = 0;
        volts_index = 0;
        break;

      case Probe:                                   //探測
        break;

      case Mode:
        digital_wave_option++;
        if (digital_wave_option > 2)
          digital_wave_option = 0;
        break;

      case Filter:
        current_filter++;
        if (current_filter > 3)
          current_filter = 0;
        break;

      default:
        break;

    }
   }
  }
}

void hide_menu() {
  menu = false;
}

void hide_all() {                               //選擇按鈕選項
  menu = false;
  info = false;
}

void show_menu() {
  menu = true;
}

String strings_vdiv() {
  return "";
}
```

```
String strings_sdiv() {
  return "";
}

String strings_offset() {
  return "";
}

String strings_toffset() {
  return "";
}

String strings_freq() {
  return "";
}

String strings_peak() {
  return "";
}

String strings_vmax() {
  return "";
}

String strings_vmin() {
  return "";
}

String strings_filter() {
  return "";
}
```

副程式 (data_analysis)

```
void peak_mean(uint16_t *i2s_buffer,
uint32_t len, float * max_value, float *
min_value, float *pt_mean) {
  max_value[0] = i2s_buffer[0];
  min_value[0] = i2s_buffer[0];
  mean_filter filter(5);
  filter.init(i2s_buffer[0]);

  float mean = 0;
  for (uint32_t i = 1; i < len; i++) {

    float value =
filter.filter((float)i2s_buffer[i]);
    if (value > max_value[0])
      max_value[0] = value;
```

```
   if (value < min_value[0])
     min_value[0] = value;

   mean += i2s_buffer[i];
  }
  mean /= float(BUFF_SIZE);
  mean = to_voltage(mean);
  pt_mean[0] = mean;
}
bool digital_analog(uint16_t
*i2s_buffer, uint32_t max_v, uint32_t
min_v) {
  uint32_t upper_threshold = max_v -
0.05 * (max_v - min_v);

  uint32_t lower_threshold = min_v +
0.05 * (max_v - min_v);
  uint32_t digital_data = 0;
  uint32_t analog_data = 0;
  for (uint32_t i = 0; i < BUFF_SIZE; i++) {
   if (i2s_buffer[i] > lower_threshold) {
    if (i2s_buffer[i] > upper_threshold) {

      digital_data++;
    }
    else {

      analog_data++;
    }
   }
   else {

    digital_data++;
   }
  }

  if (analog_data < digital_data)
   return true;
  return false;
}

void trigger_freq_analog(
uint16_t*i2s_buffer,
float sample_rate,
float mean,uint32_t max_v,
uint32_t min_v,
float *pt_freq,
```

```
//正負峰值、平均值
//true if digital/ false if analog 判斷數
位 or 類比訊號

//計算上臨界電壓 +Vp-Vp-p*5%

//計算下臨界電壓 –Vp+Vp-p*5%

//HIGH DIGITAL

//ANALOG/TRANSITION

//LOW DIGITAL

//more than 50% of data is analog
//數位
//類比

//類比頻率計算
```

```
float *pt_period,
uint32_t *pt_trigger0,
uint32_t *pt_trigger1) {

 float freq = 0;
 float period = 0;
 bool signal_side = false;
 uint32_t trigger_count = 0;
 uint32_t trigger_num = 10;
 uint32_t trigger_temp[trigger_num] =
{0};
 uint32_t trigger_index = 0;


 if (to_voltage(i2s_buffer[0]) > mean) {



  signal_side = true;
 }



 uint32_t wave_center = (max_v +
min_v) / 2;
 for (uint32_t i = 1 ; i < BUFF_SIZE; i++) {
  if (signal_side && i2s_buffer[i] <
wave_center - (wave_center - min_v) *
0.2) {
    signal_side = false;
  }
  else if (!signal_side && i2s_buffer[i] >
wave_center + (max_v - wave_center) *
0.2) {
    freq++;
    if (trigger_count < trigger_num) {
     trigger_temp[trigger_count] = i;
     trigger_count++;
    }
    signal_side = true;
  }
 }


 if (trigger_count < 2) {
  trigger_temp[0] = 0;
  trigger_index = 0;
```

```
//計數示觸發器
//觸發次數 10個正負半週=5週期


//get initial signal relative to the mean
//與平均值比較    >mean 正半週
                  <mean 負半週


//waveform repetitions calculation + get
triggers time
//電壓中點=(最大+最小)/2


//負半週


//正半週


//frequency calculation 頻率計算初始
化
```

```
  freq = 0;
  period = 0;
 }
 else {

  freq = freq * 1000 / 50;
  period = (float)(sample_rate * 1000.0)
/ freq;

  if (freq < 2000 && freq > 80) {
    period = 0;
    for (uint32_t i = 1; i < trigger_count;
i++) {
      period += trigger_temp[i] -
trigger_temp[i - 1];
    }
    period /= (trigger_count - 1);
    freq = sample_rate * 1000 / period;
  }


  else if (trigger_count > 1 && freq <=
80) {
    period = trigger_temp[1] -
trigger_temp[0];
    freq = sample_rate * 1000 / period;
  }
 }



 uint32_t trigger2 = 0;
 if (trigger_temp[0] - period * 0.05 > 0
&& trigger_count > 1) {
  trigger_index = trigger_temp[0] -
period * 0.05;
  trigger2 = trigger_temp[1] - period *
0.05;
 }
 else if (trigger_count > 2) {
  trigger_index = trigger_temp[1] -
period * 0.05;
  if (trigger_count > 2)
    trigger2 = trigger_temp[2] - period *
0.05;
 }
 pt_trigger0[0] = trigger_index;
```

```
//simple frequency calculation fair
enough for frequencies over 2khz (20hz
resolution)      1KHz/50=20Hz
//f=2KHz
//from 2000 to 80 hz -> uses mean of
the periods for precision
      (介於2000~800Hz 之間 使用週期
的平均值提高準確度)
//80Hz~2KHz



//under 80hz, single period for
frequency calculation
(低於80Hz 頻率計算單個週期)



//setting triggers offset and getting
second trigger for debug cursor on
drawn_channel1
/*
   The trigger function uses a rise
porcentage (5%) obove the mean, thus,
   the real waveform starting point is
some datapoints back.
   The resulting trigger gets a negative
offset of 5% of the calculated period
 */
```

```
  pt_trigger1[0] = trigger2;
  pt_freq[0] = freq;
  pt_period[0] = period;

}

void trigger_freq_digital(
uint16_t *i2s_buffer,
float sample_rate,
float mean,
uint32_t max_v,
uint32_t min_v,
float *pt_freq,
float *pt_period,
uint32_t *pt_trigger0) {
  float freq = 0;
  float period = 0;
  bool signal_side = false;
  uint32_t trigger_count = 0;
  uint32_t trigger_num = 10;
  uint32_t trigger_temp[trigger_num] =
{0};
  uint32_t trigger_index = 0;

  if (to_voltage(i2s_buffer[0]) > mean) {
    signal_side = true;
  }

  uint32_t wave_center = (max_v +
min_v) / 2;
  bool normal_high = (mean >
to_voltage(wave_center)) ? true : false;
  if (max_v - min_v > 4095 * (0.4 / 3.3)) {

    for (uint32_t i = 1 ; i < BUFF_SIZE; i++)
{
     if (signal_side && i2s_buffer[i] <
wave_center - (wave_center - min_v) *
0.2) {

       if (trigger_count < trigger_num &&
normal_high) {
         trigger_temp[trigger_count] = i;
         trigger_count++;
}
       signal_side = false;
     }
```

```
//數位頻率計算

//計數式觸發器
//觸發次數 10個(正負半周)=5週期

//get initial signal relative to the mean
//waveform repetitions calculation + get
triggers time
//電壓中點=最大+最小/2

//判斷平均值否大於電壓中點
//0.4/3.3=x/4095(比例)

//signal was high, fell -> trigger if
normal high
//計數值+1

//負緣觸發
```

```
    else if (!signal_side && i2s_buffer[i]
> wave_center + (max_v - wave_center)
* 0.2) {
      freq++;

      //signal was low, rose -> trigger if
normal low
      if (trigger_count < trigger_num
&& !normal_high) {
        trigger_temp[trigger_count] = i;
        trigger_count++;                    //計數值+1
      }
      signal_side = true;                   //正緣觸發
    }
   }
   freq = freq * 1000 / 50;
   period = (float)(sample_rate * 1000.0)
/ freq;
   if (trigger_count > 1) {

    if (freq < 2000 && freq > 80) {         //from 2000 to 80 hz -> uses mean of
      period = 0;                           the periods for precision
      for (uint32_t i = 1; i < trigger_count;  (介於2000~800Hz 之間 使用週期的平
i++) {                                      均值提高準確度)
        period += trigger_temp[i] -
trigger_temp[i - 1];
      }
      period /= (trigger_count - 1);
      freq = sample_rate * 1000 / period;   //頻率=取樣率*1000/週期
    }
    else if (trigger_count > 1 && freq <=   //低於80Hz 單個頻率計算週期
80) {
      period = trigger_temp[1] -
trigger_temp[0];
      freq = sample_rate * 1000 / period;
    }
   }
   trigger_index = trigger_temp[0];
   if (trigger_index > 10)
    trigger_index -= 10;
   else
    trigger_index = 0;
  }
  pt_trigger0[0] = trigger_index;
  pt_freq[0] = freq;
  pt_period[0] = period;
}
```

4-3 外殼設計

我們的外殼是以雷射切割的方式製作，而運用的程式是 Laserbox，此程式能讓我們更方便製作外殼，像利用此程式我們可以直接在變腦上調整列印的位置，以避免有超出或浪費材料之問題。

圖4-11 Laserbox 圖示

在設計外殼時我們需考慮到的地方有按鈕、螢幕、BNC 接頭與充電孔的大小和位置。

4-3 外殼設計

圖4-12 外殼設計

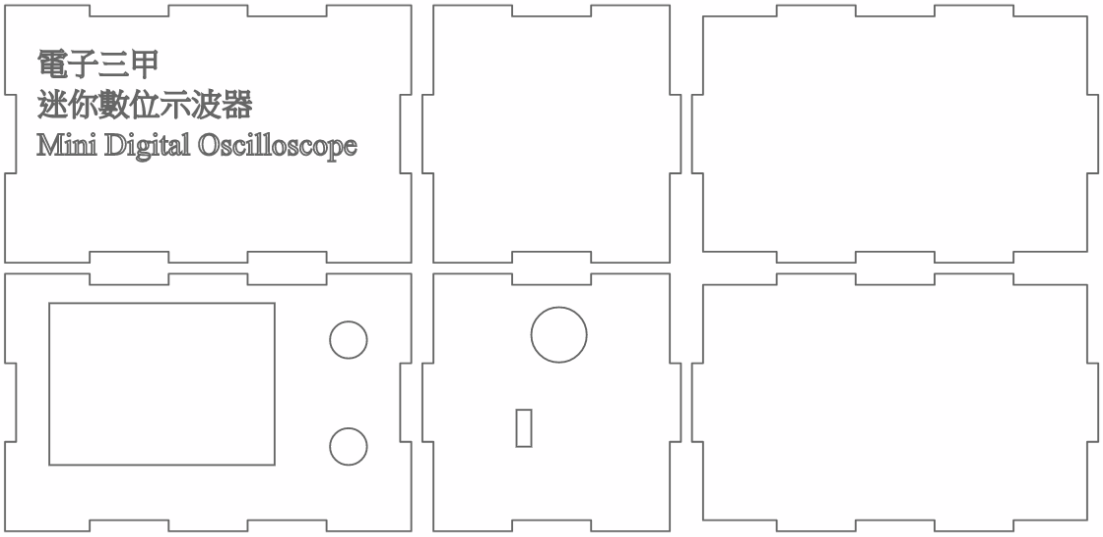
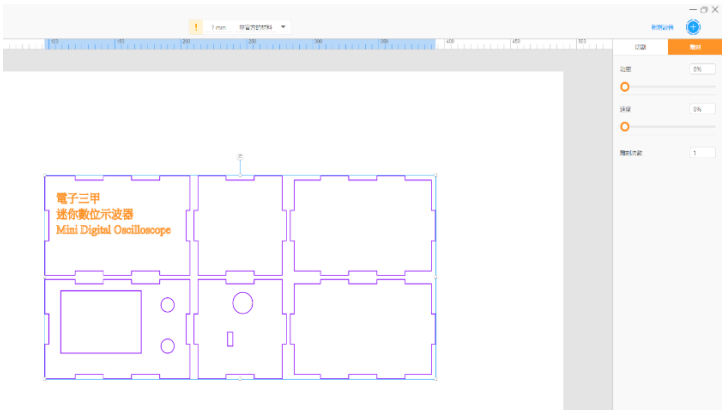


圖4-13 雷射切割製作



將設計好的外殼模板與雷射切割機連線（如圖4-14）放入雷射切割機切割（如圖4-15）切割完後進行組裝（如圖4-16）,完成後如圖4-17

圖4-14 雷射切割機連線

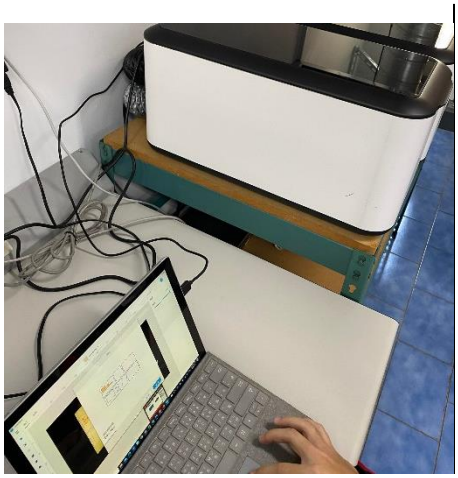


圖4-15 雷射切割機切割

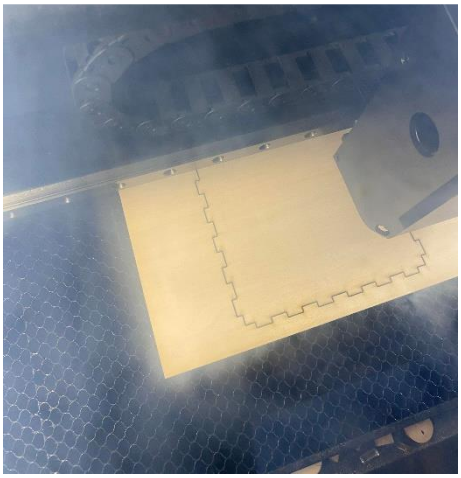


圖4-16 組裝

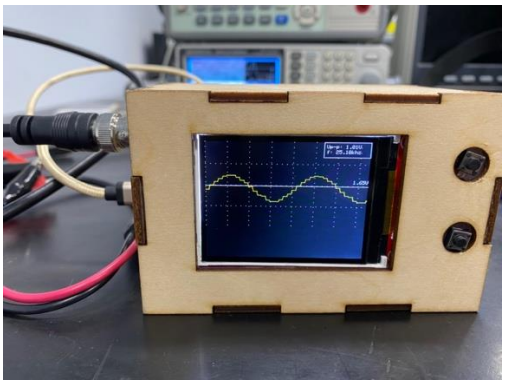
圖4-17 外殼完成照

4-4 成果

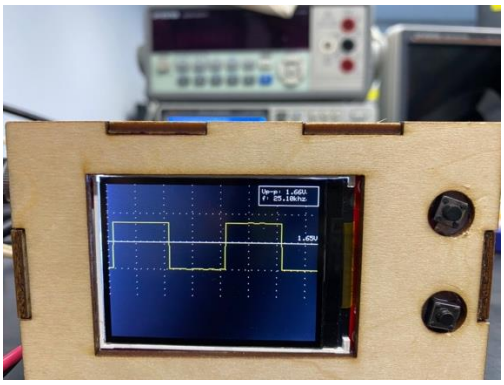
我們這次測試了三種不同波型,分別是正弦波,三角波與方波

最佳的量測頻率為20HZ-100KHZ

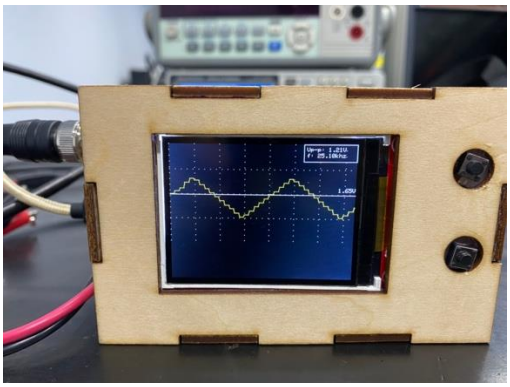
▼ 圖4-18　正弦波量測



▼ 圖4-19　方波量測

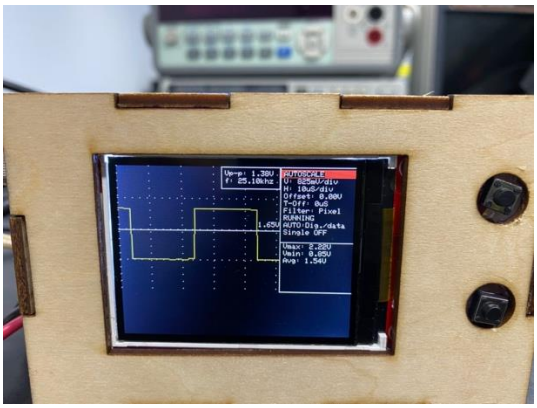


▼ 圖4-20 三角波量測



另外還測試了表單功能（如圖4-21）

▼ 圖4-21 表單測試



4-5 問題與解決

遭遇問題一:電路圖設計錯誤

第一次洗板子時發現電路線寬過窄(如圖4-22),

可能導致兩焊點未成功連接(如圖4-23)

圖4-22 電路圖

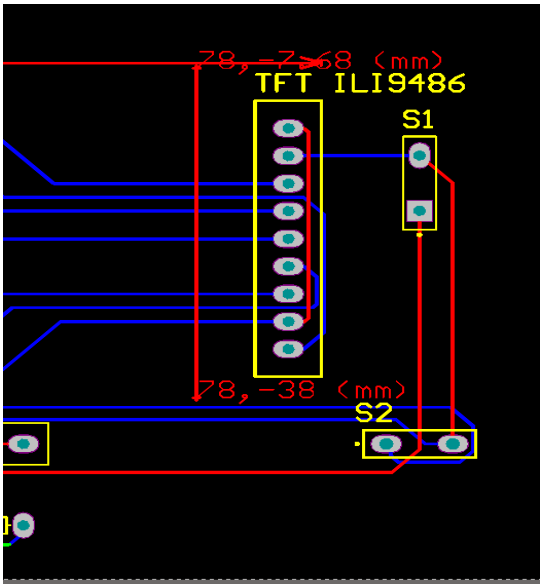
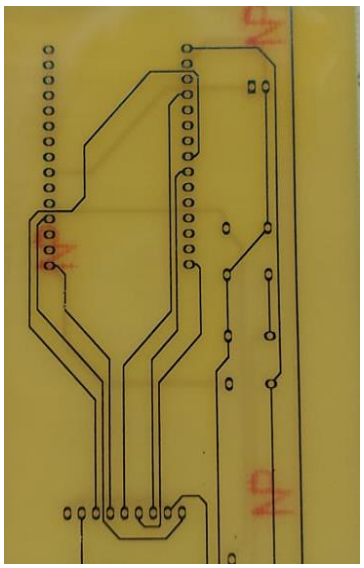


圖4-23電路板



解決方法:

點 AD 右下角的 PCB 選單，再按左鍵點開後選擇 PCB Filter(如圖4-24)，接著輸入你要獨立選取的內容(如圖4-25)。本專題內輸入”Iswire”來選取線段後(如圖4-18) ctrl+A 全選線段，即可更改設定。

圖4-24 PCB 選單

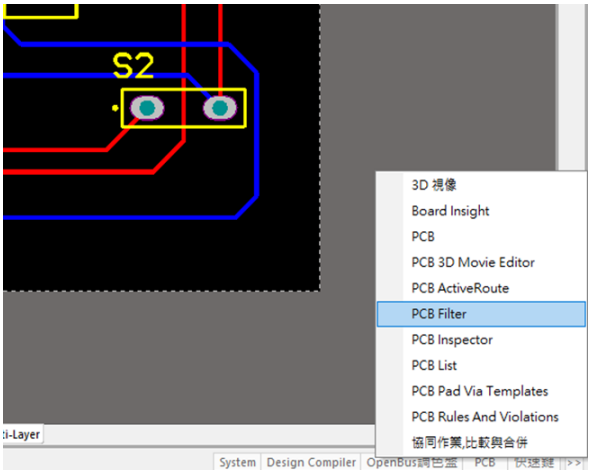


圖4-25 PCB Filter

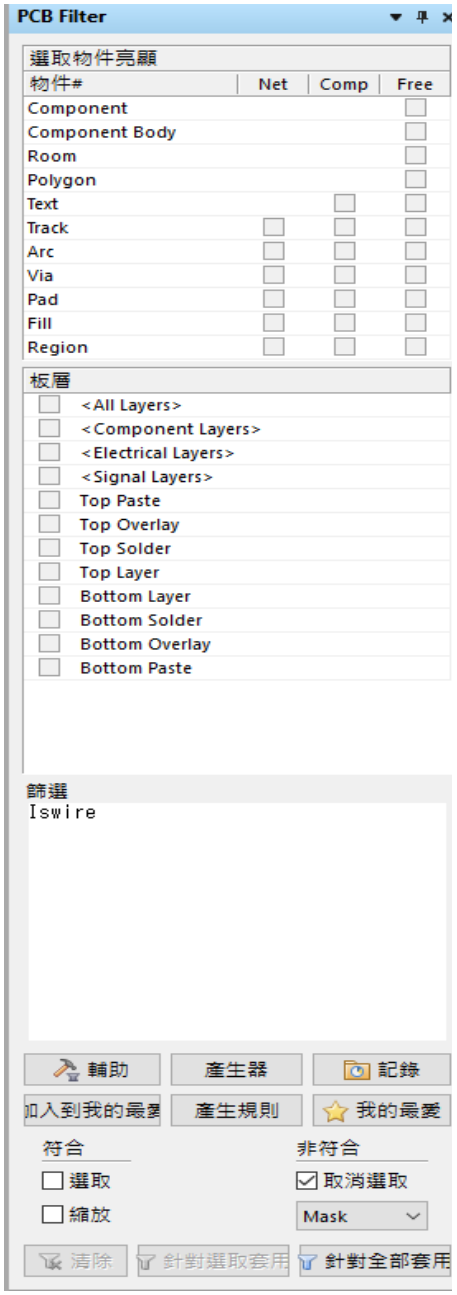
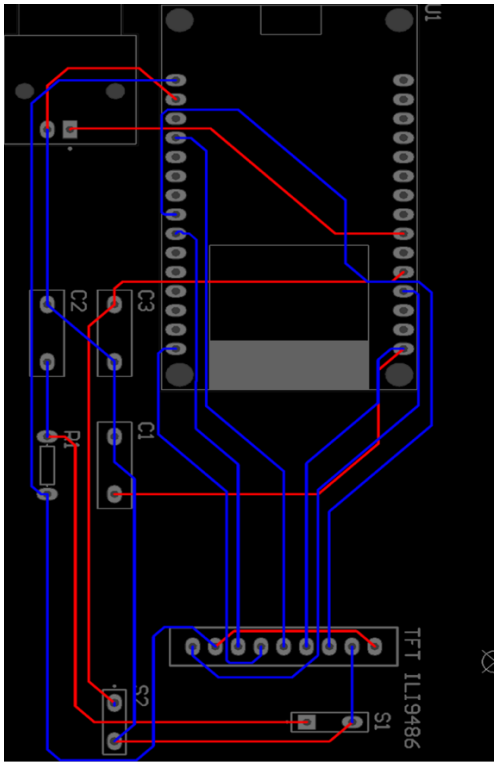


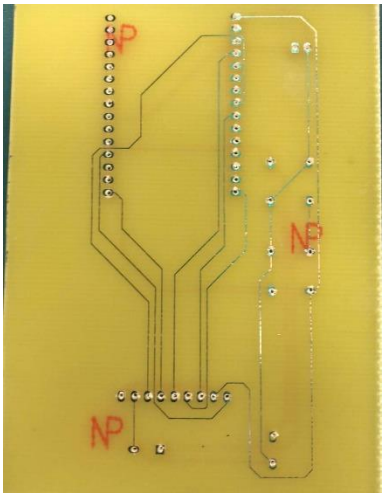
圖4-26電路板



遭遇問題二:電路板銅線脫落與焊點空焊

因為裁切電路板時沒有架高,導致電路板上的銅箔被磨掉了不少(如圖4-27)

▼ 圖4-27 電路板

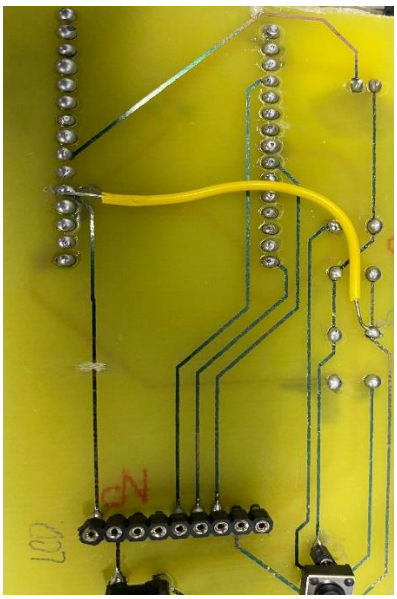


解決方法:

將兩點之間用裸銅線進行跳線(如圖4-28,圖4-29)

▼ 圖4-28 外殼與電路板

▼ 圖4-29 外殼與電路板



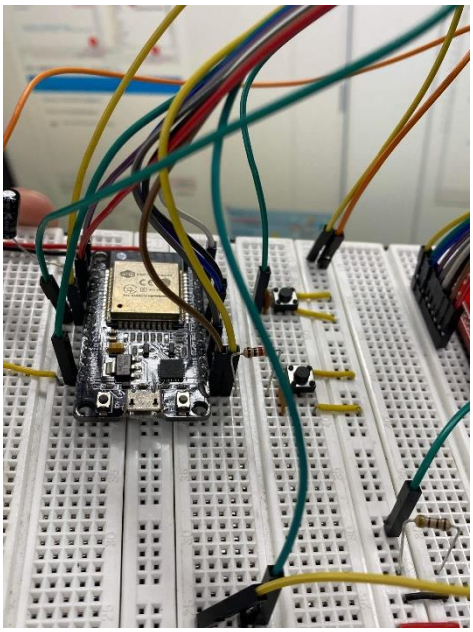
之後我們焊接完時,又發現功能不正常

於是我們把材料複製到麵包板測試一次(如圖4-)

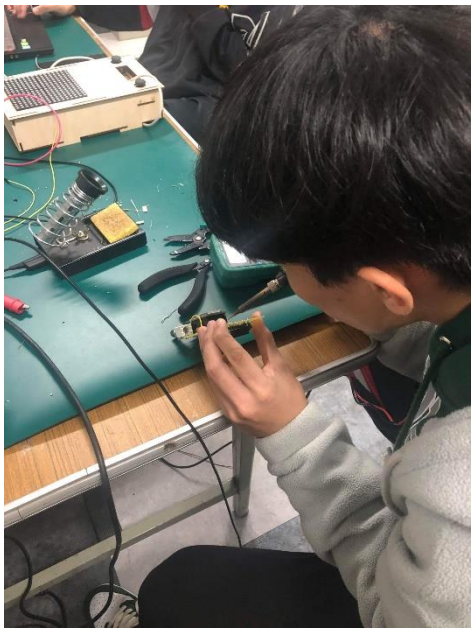
並且將電路板靜態測試完成

補完沒焊好的焊點即可!(如圖4-31)

▼ 圖4-30 麵包板



▼ 圖4-31 外殼與電路板

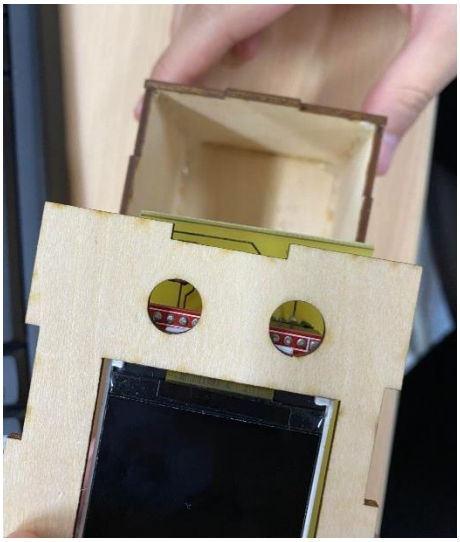


遭遇問題三:雷射切割設計錯誤

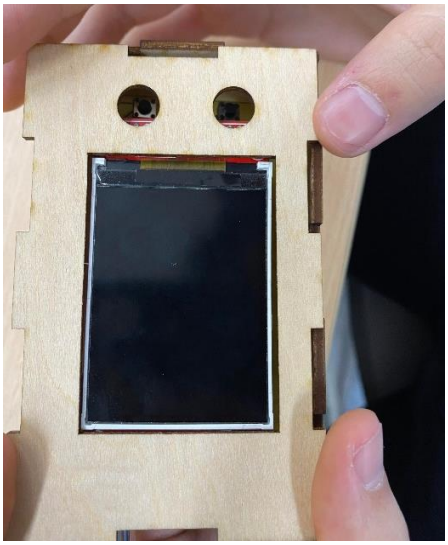
因為沒有確切考慮到電路板與外觀之間的距離

所以導致外殼無法組裝

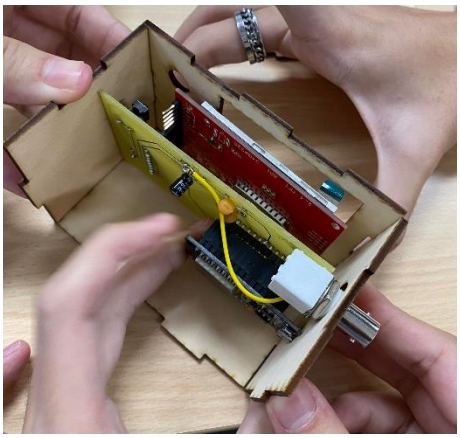
▼ 圖4-32 外殼與電路板



▼ 圖4-33 外殼與電路板



▼ 圖4-34 外殼與電路板



解決方法:

將螢幕與邊框的最短距離量好(如圖4-35)

以先放得進去外殼為主

再去固定螢幕位置

才能確保按鈕和螢幕的相對位置不會改變

圖4-35 雷射切割模板

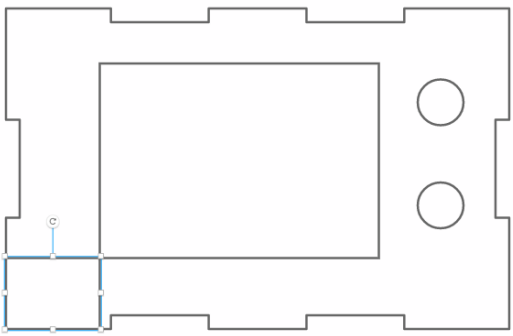


圖4-36 成品

遭遇問題四:程式無法燒錄到 ESP32裡

解決方法:

因為 ESP32需要設定特定的環境

所以必須自行安裝開發板

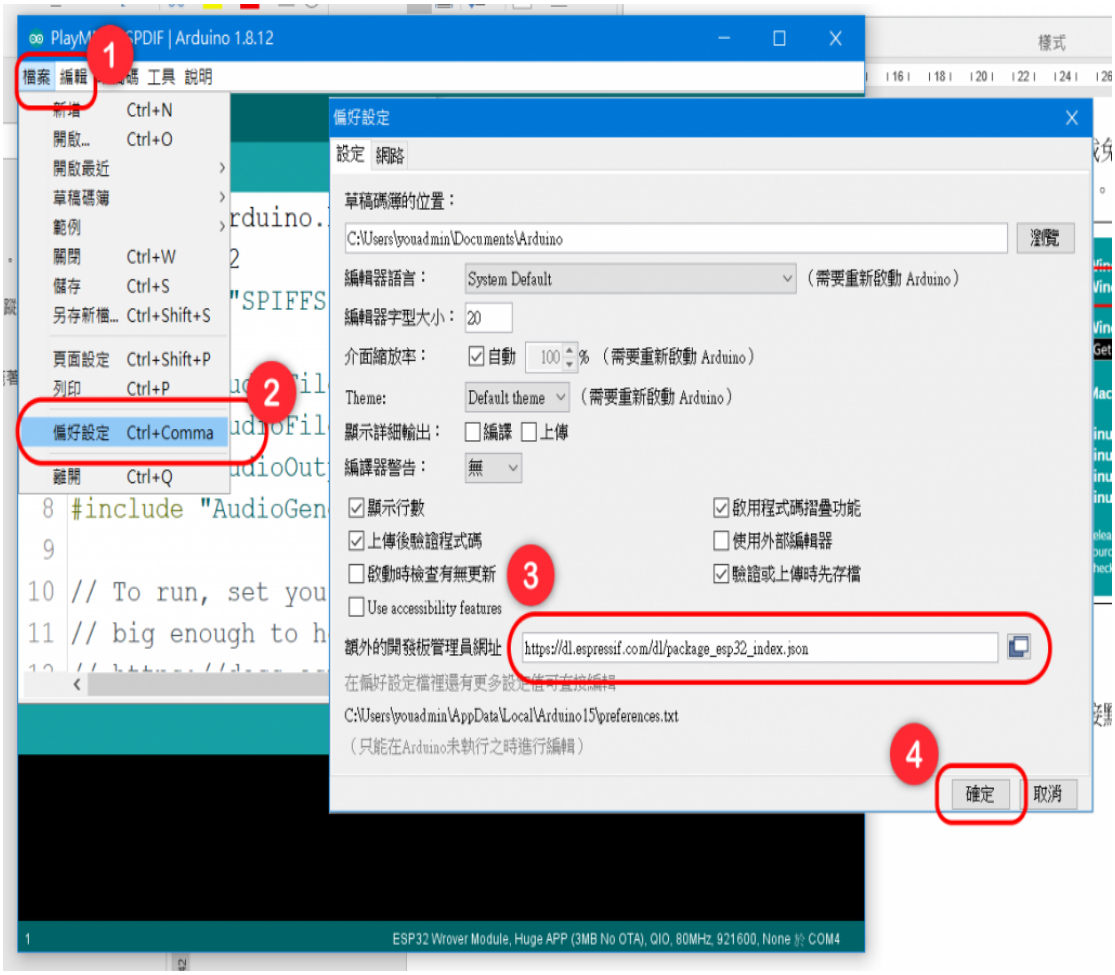
資訊來源為 https://makerpro.cc/2020/06/how-to-install-and-configure-esp32-development-environment/

開啟主程式後，選擇功能表的檔案/偏好設定，開啟偏好設定視窗

輸入偏好設定視窗中下方的額外開發板管理員網址（圖4-37）

https://dl.espressif.com/dl/package_esp32_index.json 後按確認

▼ 圖4-37 arduino 視窗圖

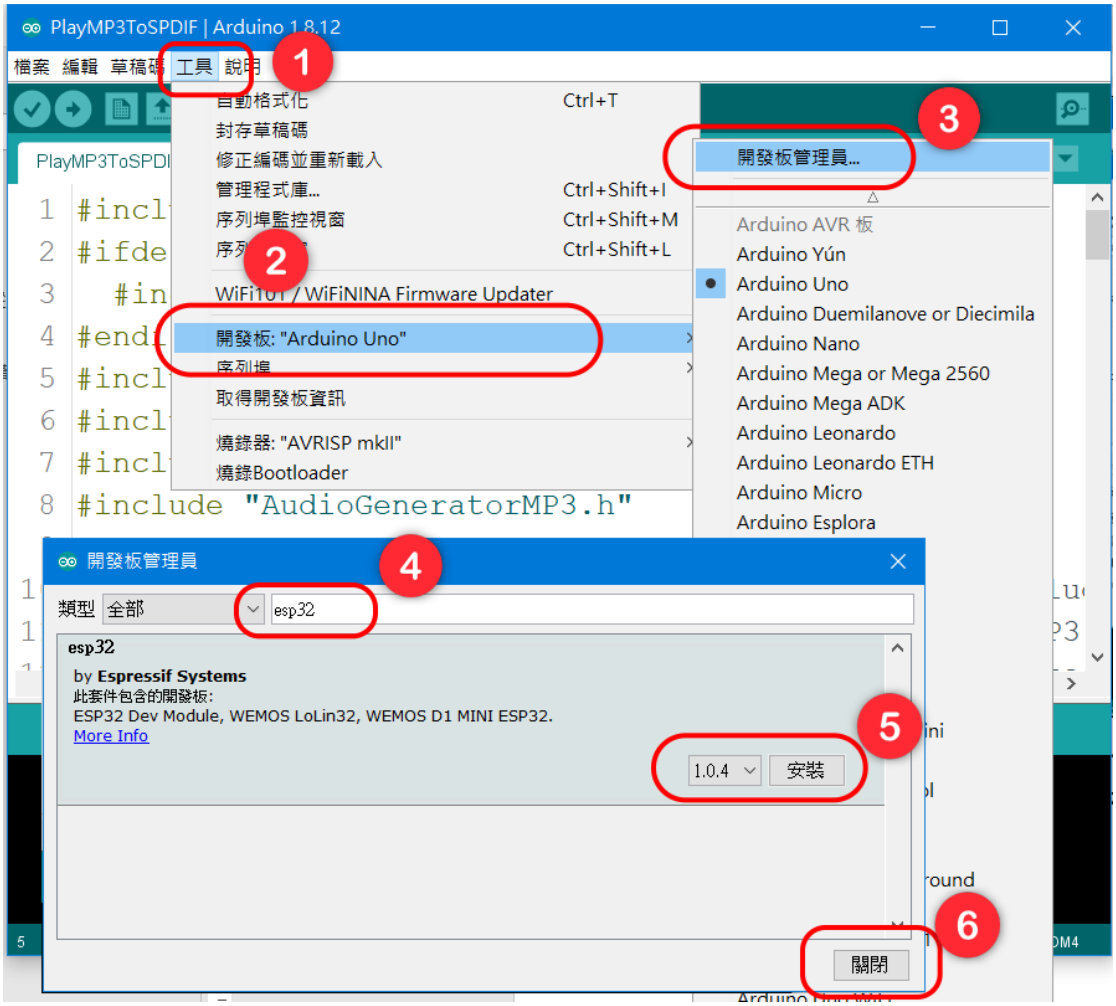


接著點選工具/開發板/開發板管理員，會出現開發板管理員視窗，

在開發板管理員視窗中，輸入關鍵字 ESP32後，選擇 ESP32核心套件，

下載核心套件並完成安裝，最後再關閉開發板管理員視窗（圖4-38）

圖4-38 arduino 視窗圖



遭遇問題五:ESP32連接 LCD 時沒有示波器畫面

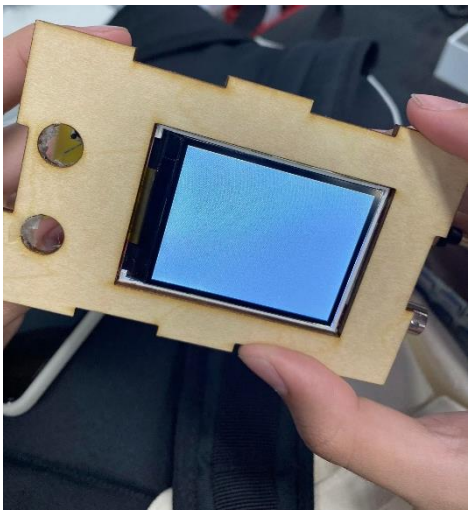
組完外殼後做最後一次 demo 時,

發現 ESP32與 LCD 的接觸時常不良,常常出現黑屏或白屏,(如圖4-39 ,4- 40)

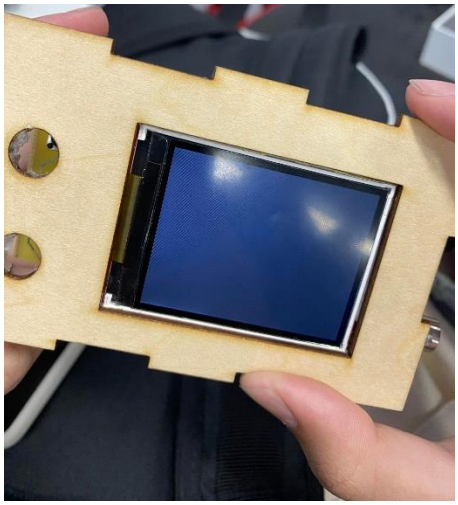
因為 ESP32的 RESET 在外殼裡面,不可能時常把外殼拆掉進去裡面按

所以必須解決

▼ 圖4-39 LCD 白屏



▼ 圖4-40 LCD 黑屏



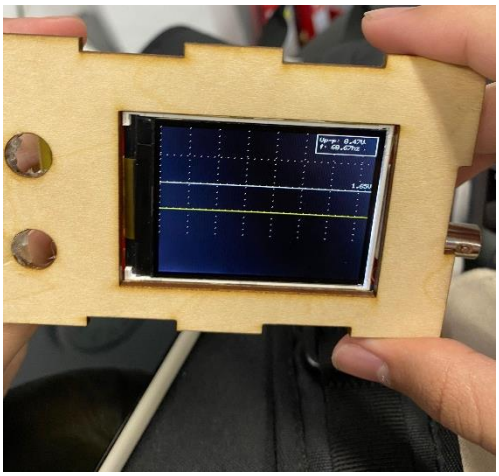
解決方法:

因為電路在麵包板上 demo 過是沒問題的,

所以大機率應該是連接 LCD 的接腳接觸不良

後來發現確實是這樣,把接觸不良的點解焊再重新焊一次就好了

▼ 圖4-41 LCD 顯示示波器



第五章 結論與建議

5-1 結論

經過這次的專題製作訓練，我們這組從中學到了很多，像運用 ESP32和 LCD 螢幕的使用方式、也更熟練地運用軟體製作電路板和外殼，更重要的是我們還學到了團隊合作，俗話說：「 三個臭皮匠，勝過一個諸葛亮。」一個人無法完成的事情，大家合作後，一定會有辦法的。

5-2 建議

製作示波器的過程要注意的事項有很多!

在製作專題的過程中常常會遇到困難，想不出來時最後還是要問老師，但是過程中已經浪費了很多時間，所以我們的建議是如果你覺得遇到的問題已經超出你的理解範圍，不要拖時間直接問老師了。

附錄

附錄一 設備與材料清單

▼ 表 設備清單

類別	設備 軟體名稱	應用說明
設備	電腦	1. 撰寫程式語言 2. 製作企畫書 3. 製作簡報 4. 查詢相關資料
設備	手機	1. 紀錄製作過程 2. 查詢相關資料
軟體	Laserbox	1.雷射切割繪圖
軟體	Arduino	1.程式設計
軟體	Altium Designer	1.電路板繪圖
工具	雷射切割機	1.雷射切割製作

▼ 表 材料清單

類別名稱	材 料 名 稱	單位	數量	應 用 說 明
外殼	雷射切割	片	6	製作示波器外殼
開發版	ESP32-WROOM	個	1	儲存信號
液晶顯示器	TFT ILI9486	個	1	顯示波形
工具	麵包版	個	1	測試電路
元件	電容	個	3	製作電路板
元件	按鈕	個	2	製作電路板
元件	電阻	個	1	製作電路板
元件	BNC 接頭	個	1	製作電路板
電路板	感光電路板	片	1	製作電路板

附錄二 成員簡歷

▼ 表 成員簡歷–林峻彣

姓　　名	林峻彣	班　　級	電子三甲	
曾修習專業科目	基礎電子實習 電子學實習 電腦輔助設計實習 基本電學實習 程式設計實習 行動裝置應用實習 單晶片微處理機實習 可程式邏輯設計實習 微電腦應用實習 程式設計實習 電子電路實習 電子學 基本電學 數位邏輯 微處理機			
參與專題工作項目	1. 雷射切割繪圖 2. 雷射切割製作 3. 報告書製作 4. 電路板鑽孔 5. 外殼組裝			
經歷簡介	擔任108學年度班級幹部-輔導幹事 擔任108學年度班級幹部-內掃幹事 擔任109學年度班級幹部-環保衛生幹事 擔任109學年度班級幹部-內掃幹事 擔任110學年度班級幹部-體育幹事 工業電子丙級證照			

▼ 表 成員簡歷-陳致維

姓　　名	陳致維	班　　級	電子三甲	
曾修習 專業科目	基礎電子實習 電子學實習 電腦輔助設計實習 基本電學實習 程式設計實習 行動裝置應用實習 單晶片微處理機實習 可程式邏輯設計實習 微電腦應用實習 程式設計實習 電子電路實習 電子學 基本電學 數位邏輯 微處理機			
參與專題 工作項目	1. 雷射切割製作 2. 程式分析 3. 焊接 4. 功能測試 5. 找尋資料 6. 報告製作			
經歷簡介	擔任108學年度班級幹部-環保衛生幹事 擔任108學年度班級幹部-輔導幹事 擔任109學年度班級幹部-環保衛生幹事 擔任109學年度班級幹部-體育幹事 擔任110學年度班級幹部-輔導幹事 工業電子丙級證照 專業英文詞彙能力國際認證 PVQC 證照			

▼ 表 成員簡歷-簡廷諭

姓　　名	簡廷諭	班　　級	電子三甲	
曾修習 專業科目	基礎電子實習 電子學實習 電腦輔助設計實習 基本電學實習 程式設計實習 行動裝置應用實習 單晶片微處理機實習 可程式邏輯設計實習 微電腦應用實習 程式設計實習 電子電路實習 電子學 基本電學 數位邏輯 微處理機			
參與專題 工作項目	1. 程式分析 2. 焊接 3. 功能測試 4. 找尋資料 5. 報告製作			
經歷簡介	擔任109學年度班級幹部-圖書幹事 工業電子丙級證照 專業英文詞彙能力國際認證 PVQC 證照			

▼ 表 成員簡歷-顏冠穎

姓　　名	顏冠穎	班　　級	電子三甲	
曾修習專業科目	基礎電子實習 電子學實習 電腦輔助設計實習 基本電學實習 程式設計實習 行動裝置應用實習 單晶片微處理機實習 可程式邏輯設計實習 微電腦應用實習 程式設計實習 汽車電子應用實習 電子學 基本電學 數位邏輯 微處理機			
參與專題工作項目	1. 電路分析 2. 電路製作 3. 功能測試 4. 找尋資料 5. 報告製作			
經歷簡介	擔任109學年度班級幹部-輔導幹事 工業電子丙級證照 專業英文詞彙能力國際認證 PVQC 證照			

附錄三 程式碼

▼ 表 adc 程式碼

```
void characterize_adc() {
  esp_adc_cal_characterize(
    (adc_unit_t)ADC_UNIT_1,
    (adc_atten_t)ADC_CHANNEL,
    (adc_bits_width_t)ADC_WIDTH_BIT_12,
    1100,
    &adc_chars);
}
```

表 I2s_adc_esp32程式碼

```
#include <Arduino.h>
#include <driver/i2s.h>
#include <driver/adc.h>
#include <soc/syscon_reg.h>
#include <TFT_eSPI.h>
#include <SPI.h>
#include "esp_adc_cal.h"
#include "filters.h"
#include <Button2.h>
#define DELAY 1000

#define WIDTH  175 //ttgo:135  ili9341:240 (voltage)
#define HEIGHT 320 //ttog:240  ili9341:320 (time)
#define ADC_CHANNEL   ADC1_CHANNEL_5
#define NUM_SAMPLES   1000
#define I2S_NUM         (0)
#define BUFF_SIZE 50000
#define B_MULT BUFF_SIZE/NUM_SAMPLES
#define BUTTON_1        35
#define BUTTON_2        0

TFT_eSPI    tft = TFT_eSPI();

TFT_eSprite spr = TFT_eSprite(&tft);


Button2 button_mode = Button2(BUTTON_1);
Button2 button_set = Button2(BUTTON_2);

esp_adc_cal_characteristics_t adc_chars;

TaskHandle_t task_menu;
TaskHandle_t task_adc;

float v_div = 825;
float s_div = 10;
float offset = 0;
float toffset = 0;
uint8_t current_filter = 1;

enum Option {
  None,
  Autoscale,
  Vdiv,
  Sdiv,
  Offset,
  TOffset,
  Filter,
```

```
  Stop,
  Mode,
  Single,
  Clear,
  Reset,
  Probe,
  UpdateF,
  Cursor1,
  Cursor2
};

int8_t volts_index = 0;

int8_t tscale_index = 0;

uint8_t opt = None;

bool menu = false;
bool info = true;
bool set_value  = false;

float RATE = 1000;

bool auto_scale = false;

bool full_pix = true;

bool stop = false;

bool stop_change = false;

uint16_t i2s_buff[BUFF_SIZE];

bool single_trigger = false;
bool data_trigger = false;

bool updating_screen = false;
bool new_data = false;
bool menu_action = false;
uint8_t digital_wave_option = 0;

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

  configure_i2s(1000000);

  setup_screen();

  button_mode.setClickHandler(click);
  button_mode.setLongClickHandler(click_long);
```

```
  button_set.setClickHandler(click);
  button_set.setLongClickHandler(click_long);

  characterize_adc();
#ifdef DEBUG_BUF
  debug_buffer();
#endif




  xTaskCreatePinnedToCore(
    core0_task,
    "menu_handle",
    10000,
    NULL,
    0,
    &task_menu,
    0);

  xTaskCreatePinnedToCore(
    core1_task,
    "adc_handle",
    10000,
    NULL,
    3,
    &task_adc,.
    1);

}


void core0_task( void * pvParameters ) {

  (void) pvParameters;

  for (;;) {
    menu_handler();

    if (new_data || menu_action) {
      new_data = false;
      menu_action = false;

      updating_screen = true;
      update_screen(i2s_buff, RATE);
      updating_screen = false;
      vTaskDelay(pdMS_TO_TICKS(10));
      Serial.println("CORE0");
    }
```

```
    vTaskDelay(pdMS_TO_TICKS(10));
  }

}

void core1_task( void * pvParameters ) {

  (void) pvParameters;

  for (;;) {
    if (!single_trigger) {
      while (updating_screen) {
        vTaskDelay(pdMS_TO_TICKS(1));
      }
      if (!stop) {
        if (stop_change) {
          i2s_adc_enable(I2S_NUM_0);
          stop_change = false;
        }
        ADC_Sampling(i2s_buff);
        new_data = true;
      }
      else {
        if (!stop_change) {
          i2s_adc_disable(I2S_NUM_0);
          i2s_zero_dma_buffer(I2S_NUM_0);
          stop_change = true;
        }
      }
      Serial.println("CORE1");
      vTaskDelay(pdMS_TO_TICKS(300));
    }


    else {
      float old_mean = 0;
      while (single_trigger) {
        stop = true;
        ADC_Sampling(i2s_buff);
        float mean = 0;
        float max_v, min_v;
        peak_mean(i2s_buff, BUFF_SIZE, &max_v, &min_v, &mean);




        if ((old_mean != 0 && fabs(mean - old_mean) > 0.2) || to_voltage(max_v) -
to_voltage(min_v) > 0.05) {
          float freq = 0;
          float period = 0;
```

```
        uint32_t trigger0 = 0;
        uint32_t trigger1 = 0;



        bool digital_data = !false;
        if (digital_wave_option == 1) {
          trigger_freq_analog(i2s_buff, RATE, mean, max_v, min_v, &freq, &period,
&trigger0, &trigger1);
        }
        else if (digital_wave_option == 0) {
          digital_data = digital_analog(i2s_buff, max_v, min_v);
          if (!digital_data) {
            trigger_freq_analog(i2s_buff, RATE, mean, max_v, min_v, &freq,
&period, &trigger0, &trigger1);
          }
          else {
            trigger_freq_digital(i2s_buff, RATE, mean, max_v, min_v, &freq,
&period, &trigger0);
          }
        }
        else {
          trigger_freq_digital(i2s_buff, RATE, mean, max_v, min_v, &freq, &period,
&trigger0);
        }

        single_trigger = false;
        new_data = true;
        Serial.println("Single GOT");

       }

       vTaskDelay(pdMS_TO_TICKS(1));

     }
     vTaskDelay(pdMS_TO_TICKS(300));
   }
  }
}

void loop() {}
```

▼ 表 options_handler 程式碼

```
int voltage_division[6] = {

  825,
  750,
  500,
  250,
  100,
  50
};


float time_division[9] = {
  10,
  25,
  50,
  100,
  250,
  500,
  1000,
  2500,
  5000
};
};

void menu_handler() {
  button_mode.loop();
  button_set.loop();
}

void click_long(Button2& btn) {
  menu_action = true;
  if (btn == button_mode) {
    uint32_t pressed = btn.wasPressedFor();
    if (pressed > 1000) {
      opt = None;
      hide_menu();
      set_value = false;
    }
  }
  else {
    uint32_t pressed = btn.wasPressedFor();
    if (pressed > 1000) {
      if (set_value) {
        set_value = false;
      }
      else {
        switch (opt) {
          case Vdiv:
```

```
          v_div = 825;
          volts_index = 0;
          break;

        case Sdiv:
          s_div = 10;
          tscale_index = 0;
          break;

        case Offset:
          offset = 0;
          break;

        case TOffset:
          toffset = 0;
          break;
        case Filter:
          current_filter = 1;
          break;

        case None:
          info = !info;
          break;

        default:
          break;
      }
     }
    }
  }
}

void click(Button2& btn) {
 menu_action = true;

 if (set_value) {
  switch (opt) {
    case Vdiv:
      if (btn == button_set) {
        volts_index++;
        if (volts_index >= sizeof(voltage_division) / sizeof(*voltage_division)) {
          volts_index = 0;
        }
      }
      else {
        volts_index--;
        if (volts_index < 0) {
          volts_index = sizeof(voltage_division) / sizeof(*voltage_division) - 1;
        }
      }
```

```
      v_div = voltage_division[volts_index];
      break;

    case Sdiv:
      if (btn == button_mode) {
        tscale_index++;
        if (tscale_index >= sizeof(time_division) / sizeof(*time_division)) {
          tscale_index = 0;
        }
      }
      else {
        tscale_index--;
        if (tscale_index < 0) {
          tscale_index = sizeof(time_division) / sizeof(*time_division) - 1;
        }
      }

      s_div = time_division[tscale_index];
      break;

    case Offset:
      if (btn == button_mode){
        offset += 0.1 * (v_div * 4) / 3300;
      }
      else{
        offset -= 0.1 * (v_div * 4) / 3300;
      }

      if (offset > 3.3)
        offset = 3.3;
      if (offset < -3.3)
        offset = -3.3;

      break;

    case TOffset:
      if (btn == button_mode)
        toffset += 0.1 * s_div;
      else
        toffset -= 0.1 * s_div;

      break;

    default:
      break;
  }
}
else {
  if (btn == button_mode) {
```

```
    opt++;
    if (opt > Single)
      opt = None;
    if (opt == None)
      hide_menu();
    else
      show_menu();

  }
  else if (btn == button_set) {
    switch (opt) {
      case Autoscale:
        auto_scale = !auto_scale;
        break;

      case Vdiv:
        set_value = true;
        break;

      case Sdiv:
        set_value = true;
        break;

      case Offset:
        set_value = true;
        break;

      case Stop:
        stop = !stop;
        break;

      case TOffset:
        set_value = true;
        break;

      case Single:
        single_trigger = true;
        break;

      case Reset:
        offset = 0;
        v_div = 825;
        s_div = 10;
        tscale_index = 0;
        volts_index = 0;
        break;

      case Probe:
        break;
```

```
        case Mode:
          digital_wave_option++;
          if (digital_wave_option > 2)
            digital_wave_option = 0;
          break;

        case Filter:
          current_filter++;
          if (current_filter > 3)
            current_filter = 0;
          break;

        default:
          break;

      }
    }
  }
}

void hide_menu() {
  menu = false;
}

void hide_all() {
  menu = false;
  info = false;
}

void show_menu() {
  menu = true;
}

String strings_vdiv() {
  return "";
}

String strings_sdiv() {
  return "";
}

String strings_offset() {
  return "";
}

String strings_toffset() {
  return "";
}

String strings_freq() {
```

```
  return "";
}

String strings_peak() {
  return "";
}

String strings_vmax() {
  return "";
}

String strings_vmin() {
  return "";
}

String strings_filter() {
  return "";
}
```

▼ 表 screen 程式碼

```
 void setup_screen() {
 tft.init();
 tft.setRotation(3); //0:0 degree,1:90 degree,2:180 degree,3:270 degree



 spr.createSprite(HEIGHT, WIDTH);
 tft.fillScreen(TFT_BLACK);
}

int data[HEIGHT] = {0};

int step_WIDTH=WIDTH/4;
float to_scale(float reading) {
 float temp = WIDTH -
              (
                (
                  (
                    (float)((reading - 20480.0) / 4095.0)
                    + (offset / 3.3)
                  )
                  * 3300 /
                  (v_div * 4)
                )
              )
              * (WIDTH - 1)
              - 1;
 return temp;
}

float to_voltage(float reading) {
 return  (reading - 20480.0) / 4095.0 * 3.3;
}

uint32_t from_voltage(float voltage) {
 return uint32_t(voltage / 3.3 * 4095 + 20480.0);
}
void update_screen(uint16_t *i2s_buff, float sample_rate) {

 float mean = 0;
 float max_v, min_v;

 peak_mean(i2s_buff, BUFF_SIZE, &max_v, &min_v, &mean);

 float freq = 0;
 float period = 0;
 uint32_t trigger0 = 0;
 uint32_t trigger1 = 0;
```

```
  bool digital_data = false;
  if (digital_wave_option == 1) {
    trigger_freq_analog(i2s_buff, sample_rate, mean, max_v, min_v, &freq,
&period, &trigger0, &trigger1);
  }
  else if (digital_wave_option == 0) {
    digital_data = digital_analog(i2s_buff, max_v, min_v);
    if (!digital_data) {
      trigger_freq_analog(i2s_buff, sample_rate, mean, max_v, min_v, &freq,
&period, &trigger0, &trigger1);
    }
    else {
      trigger_freq_digital(i2s_buff, sample_rate, mean, max_v, min_v, &freq,
&period, &trigger0);
    }
  }
  else {
    trigger_freq_digital(i2s_buff, sample_rate, mean, max_v, min_v, &freq, &period,
&trigger0);
  }
  draw_sprite(freq, period, mean, max_v, min_v, trigger0, sample_rate, digital_data,
true);
}
void draw_sprite(float freq,
                 float period,
                 float mean,
                 float max_v,
                 float min_v,
                 uint32_t trigger,
                 float sample_rate,
                 bool digital_data,
                 bool new_data
                ) {

  max_v = to_voltage(max_v);
  min_v = to_voltage(min_v);

  String frequency = "";
  if (freq < 1000)
    frequency = String(freq) + "hz";
  else if (freq < 100000)
    frequency = String(freq / 1000) + "khz";
  else
    frequency = "----";

  String s_mean = "";
  if (mean > 1.0)
    s_mean = "Avg: " + String(mean) + "V";
  else
```

```
  s_mean = "Avg: " + String(mean * 1000.0) + "mV";

String str_filter = "";
if (current_filter == 0)
  str_filter = "None";
else if (current_filter == 1)
  str_filter = "Pixel";
else if (current_filter == 2)
  str_filter = "Mean-5";
else if (current_filter == 3)
  str_filter = "Lpass9";

String str_stop = "";
if (!stop)
  str_stop = "RUNNING";
else
  str_stop = "STOPPED";

String wave_option = "";
if (digital_wave_option == 0)
  if (digital_data )
    wave_option = "AUTO:Dig./data";
  else
    wave_option = "AUTO:Analog";
else if (digital_wave_option == 1)
  wave_option = "MODE:Analog";
else
  wave_option = "MODE:Dig./data";

if (new_data) {

  spr.fillSprite(TFT_BLACK);

  draw_grid();
  if (auto_scale) {
    auto_scale = false;
    v_div = 1000.0 * max_v / 4.0;
    s_div = period / 3.5;
    if (s_div > 7000 || s_div <= 0)
      s_div = 7000;
    if (v_div <= 0)
      v_div = 825;
  }

  if (!(digital_wave_option == 2 && trigger == 0))
    draw_channel1(trigger, 0, i2s_buff, sample_rate);

}

int shift = 220;
```

```
  if (menu) {
    spr.drawLine( 0, int(WIDTH/2), HEIGHT, int(WIDTH/2), TFT_WHITE);
//center line
    spr.fillRect(shift, 0, HEIGHT, WIDTH, TFT_BLACK);
    spr.drawRect(shift, 0, HEIGHT, WIDTH, TFT_WHITE);
    spr.fillRect(shift + 1, 3 + 10 * (opt - 1), 100, 11, TFT_RED);

    spr.drawString("AUTOSCALE",  shift + 5, 5);
    spr.drawString("V: " + String(int(v_div)) + "mV/div",  shift + 5, 15);
    spr.drawString("H: " + String(int(s_div)) + "uS/div",  shift + 5, 25);
    spr.drawString("Offset: " + String(offset) + "V",  shift + 5, 35);
    spr.drawString("T-Off: " + String((uint32_t)toffset) + "uS",  shift + 5, 45);
    spr.drawString("Filter: " + str_filter, shift + 5, 55);
    spr.drawString(str_stop, shift + 5, 65);
    spr.drawString(wave_option, shift + 5, 75);
    spr.drawString("Single " + String(single_trigger ? "ON" : "OFF"), shift + 5, 85);

    spr.drawLine(shift, 103, shift + 100, 103, TFT_WHITE);

    spr.drawString("Vmax: " + String(max_v) + "V",  shift + 5, 105);
    spr.drawString("Vmin: " + String(min_v) + "V",  shift + 5, 115);
    spr.drawString(s_mean,  shift + 5, 125);

    shift -= 80;

    spr.drawRect(shift, 0, 80, 30, TFT_WHITE);
    spr.drawString("Vp-p: " + String(max_v - min_v) + "V",  shift + 5, 5);
    spr.drawString("f: " + frequency,  shift + 5, 15);
    String offset_line = String((2.0 * v_div) / 1000.0 - offset) + "V";
    spr.drawString(offset_line,  shift + 50, 75);

    if (set_value) {
      spr.fillRect(HEIGHT-11, 0, 11, 11, TFT_BLUE);
      spr.drawRect(HEIGHT-11, 0, 11, 11, TFT_WHITE);
      spr.drawLine(HEIGHT-9, 5, HEIGHT-2 , 5, TFT_WHITE);
      spr.drawLine(HEIGHT-6, 2, HEIGHT-6, 8, TFT_WHITE);
      spr.fillRect(HEIGHT-11, WIDTH-11, 11, 11, TFT_BLUE);
      spr.drawRect(HEIGHT-11, WIDTH-11, 11, 11, TFT_WHITE);
      spr.drawLine(HEIGHT-9, WIDTH-6, HEIGHT-2, WIDTH-6, TFT_WHITE);
    }
  }

  else if (info) {
    spr.drawLine( 0, int(WIDTH/2), HEIGHT, int(WIDTH/2), TFT_WHITE);
    spr.drawRect(shift + 10, 0, HEIGHT - shift - 10, 30, TFT_WHITE);
    spr.drawString("Vp-p: " + String(max_v - min_v) + "V",  shift + 15, 5);
    spr.drawString("f: " + frequency,  shift + 15, 15);
    String offset_line = String((2.0 * v_div) / 1000.0 - offset) + "V";
    spr.drawString(offset_line,  shift + 70, 75);
  }
```

```
  spr.pushSprite(0, 0);

  yield();
}

void draw_grid() {
  for (int i = 0; i < (HEIGHT/10); i++) {
    spr.drawPixel(i * 10, step_WIDTH, TFT_WHITE);
    spr.drawPixel(i * 10, step_WIDTH*2, TFT_WHITE);
    spr.drawPixel(i * 10, step_WIDTH*3, TFT_WHITE);
  }
  for (int i = 0; i < WIDTH; i += 10) {
    for (int j = 0; j < HEIGHT; j += step_WIDTH) {
      spr.drawPixel(j, i, TFT_WHITE);
    }
  }
}

void draw_channel1(uint32_t trigger0, uint32_t trigger1, uint16_t *i2s_buff, float
sample_rate) {

  data[0] = to_scale(i2s_buff[trigger0]);
  low_pass filter(0.99);
  mean_filter mfilter(5);
  mfilter.init(i2s_buff[trigger0]);
  filter._value = i2s_buff[trigger0];
  float data_per_pixel = (s_div / step_WIDTH) / (sample_rate / 1000);



  uint32_t index_offset = (uint32_t)(toffset / data_per_pixel);
  trigger0 += index_offset;
  uint32_t old_index = trigger0;
  float n_data = 0, o_data = to_scale(i2s_buff[trigger0]);
  for (uint32_t i = 1; i < HEIGHT; i++) {
    uint32_t index = trigger0 + (uint32_t)((i + 1) * data_per_pixel);
    if (index < BUFF_SIZE) {
      if (full_pix && s_div > step_WIDTH && current_filter == 0) {
        uint32_t max_val = i2s_buff[old_index];
        uint32_t min_val = i2s_buff[old_index];
        for (int j = old_index; j < index; j++) {

          if (i2s_buff[j] > max_val)
            max_val = i2s_buff[j];
          else if (i2s_buff[j] < min_val)
            min_val = i2s_buff[j];

        }
        spr.drawLine(i, to_scale(min_val), i, to_scale(max_val), TFT_YELLOW);
```

```
      }
      else {
        if (current_filter == 2)
          n_data = to_scale(mfilter.filter((float)i2s_buff[index]));
        else if (current_filter == 3)
          n_data = to_scale(filter.filter((float)i2s_buff[index]));
        else
          n_data = to_scale(i2s_buff[index]);

        spr.drawLine(i - 1, o_data, i, n_data, TFT_YELLOW);
        o_data = n_data;
      }

    }
    else {
      break;
    }
    old_index = index;
  }
}
```

▼ 表 data_analysis 程式碼

```
void peak_mean(uint16_t *i2s_buffer, uint32_t len, float * max_value, float *
min_value, float *pt_mean) {
  max_value[0] = i2s_buffer[0];
  min_value[0] = i2s_buffer[0];
  mean_filter filter(5);
  filter.init(i2s_buffer[0]);

  float mean = 0;
  for (uint32_t i = 1; i < len; i++) {

    float value = filter.filter((float)i2s_buffer[i]);
    if (value > max_value[0])
      max_value[0] = value;
    if (value < min_value[0])
      min_value[0] = value;

    mean += i2s_buffer[i];
  }
  mean /= float(BUFF_SIZE);
  mean = to_voltage(mean);
  pt_mean[0] = mean;
}


//true if digital/ false if analog
bool digital_analog(uint16_t *i2s_buffer, uint32_t max_v, uint32_t min_v) {
  uint32_t upper_threshold = max_v - 0.05 * (max_v - min_v);
  uint32_t lower_threshold = min_v + 0.05 * (max_v - min_v);
  uint32_t digital_data = 0;
  uint32_t analog_data = 0;
  for (uint32_t i = 0; i < BUFF_SIZE; i++) {
    if (i2s_buffer[i] > lower_threshold) {
      if (i2s_buffer[i] > upper_threshold) {
        //HIGH DIGITAL
        digital_data++;
      }
      else {
        //ANALOG/TRANSITION
        analog_data++;
      }
    }
    else {
      //LOW DIGITAL
      digital_data++;
    }
  }

  //more than 50% of data is analog
```

```
  if (analog_data < digital_data)
    return true;

  return false;
}

void trigger_freq_analog(uint16_t *i2s_buffer,
                         float sample_rate,
                         float mean,
                         uint32_t max_v,
                         uint32_t min_v,
                         float *pt_freq,
                         float *pt_period,
                         uint32_t *pt_trigger0,
                         uint32_t *pt_trigger1) {
  float freq = 0;
  float period = 0;
  bool signal_side = false;
  uint32_t trigger_count = 0;
  uint32_t trigger_num = 10;
  uint32_t trigger_temp[trigger_num] = {0};
  uint32_t trigger_index = 0;

  //get initial signal relative to the mean
  if (to_voltage(i2s_buffer[0]) > mean) {
    signal_side = true;
  }


  //waveform repetitions calculation + get triggers time
  uint32_t wave_center = (max_v + min_v) / 2;
  for (uint32_t i = 1 ; i < BUFF_SIZE; i++) {
    if (signal_side && i2s_buffer[i] < wave_center - (wave_center - min_v) * 0.2) {
      signal_side = false;
    }
    else if (!signal_side && i2s_buffer[i] > wave_center + (max_v - wave_center) *
0.2) {
      freq++;
      if (trigger_count < trigger_num) {
        trigger_temp[trigger_count] = i;
        trigger_count++;
      }
      signal_side = true;
    }
  }

  //frequency calculation
  if (trigger_count < 2) {
    trigger_temp[0] = 0;
    trigger_index = 0;
```

```
    freq = 0;
    period = 0;
  }
  else {

    //simple frequency calculation fair enough for frequencies over 2khz (20hz
resolution)
    freq = freq * 1000 / 50;
    period = (float)(sample_rate * 1000.0) / freq; //us

    //from 2000 to 80 hz -> uses mean of the periods for precision
    if (freq < 2000 && freq > 80) {
      period = 0;
      for (uint32_t i = 1; i < trigger_count; i++) {
        period += trigger_temp[i] - trigger_temp[i - 1];
      }
      period /= (trigger_count - 1);
      freq = sample_rate * 1000 / period;
    }

    //under 80hz, single period for frequency calculation
    else if (trigger_count > 1 && freq <= 80) {
      period = trigger_temp[1] - trigger_temp[0];
      freq = sample_rate * 1000 / period;
    }
  }

  //setting triggers offset and getting second trigger for debug cursor on
drawn_channel1
  /*
     The trigger function uses a rise porcentage (5%) above the mean, thus,
     the real waveform starting point is some datapoints back.
     The resulting trigger gets a negative offset of 5% of the calculated period
  */
  uint32_t trigger2 = 0;
  if (trigger_temp[0] - period * 0.05 > 0 && trigger_count > 1) {
    trigger_index = trigger_temp[0] - period * 0.05;
    trigger2 = trigger_temp[1] - period * 0.05;
  }
  else if (trigger_count > 2) {
    trigger_index = trigger_temp[1] - period * 0.05;
    if (trigger_count > 2)
      trigger2 = trigger_temp[2] - period * 0.05;
  }


  pt_trigger0[0] = trigger_index;
  pt_trigger1[0] = trigger2;
  pt_freq[0] = freq;
  pt_period[0] = period;
```

```
}


void trigger_freq_digital(uint16_t *i2s_buffer,
                          float sample_rate,
                          float mean,
                          uint32_t max_v,
                          uint32_t min_v,
                          float *pt_freq,
                          float *pt_period,
                          uint32_t *pt_trigger0) {

  float freq = 0;
  float period = 0;
  bool signal_side = false;
  uint32_t trigger_count = 0;
  uint32_t trigger_num = 10;
  uint32_t trigger_temp[trigger_num] = {0};
  uint32_t trigger_index = 0;

  //get initial signal relative to the mean
  if (to_voltage(i2s_buffer[0]) > mean) {
    signal_side = true;
  }

  //waveform repetitions calculation + get triggers time
  uint32_t wave_center = (max_v + min_v) / 2;
  bool normal_high = (mean > to_voltage(wave_center)) ? true : false;
  if (max_v - min_v > 4095 * (0.4 / 3.3)) {
    for (uint32_t i = 1 ; i < BUFF_SIZE; i++) {
      if (signal_side && i2s_buffer[i] < wave_center - (wave_center - min_v) * 0.2) {

        //signal was high, fell -> trigger if normal high
        if (trigger_count < trigger_num && normal_high) {
          trigger_temp[trigger_count] = i;
          trigger_count++;
        }

        signal_side = false;

      }
      else if (!signal_side && i2s_buffer[i] > wave_center + (max_v - wave_center) *
0.2) {
        freq++;

        //signal was low, rose -> trigger if normal low
        if (trigger_count < trigger_num && !normal_high) {
          trigger_temp[trigger_count] = i;
          trigger_count++;
```

```
      }

      signal_side = true;
    }
  }

  freq = freq * 1000 / 50;
  period = (float)(sample_rate * 1000.0) / freq; //us

  if (trigger_count > 1) {
    //from 2000 to 80 hz -> uses mean of the periods for precision
    if (freq < 2000 && freq > 80) {
      period = 0;
      for (uint32_t i = 1; i < trigger_count; i++) {
        period += trigger_temp[i] - trigger_temp[i - 1];
      }
      period /= (trigger_count - 1);
      freq = sample_rate * 1000 / period;
    }

    //under 80hz, single period for frequency calculation
    else if (trigger_count > 1 && freq <= 80) {
      period = trigger_temp[1] - trigger_temp[0];
      freq = sample_rate * 1000 / period;
    }
  }

  trigger_index = trigger_temp[0];

  if (trigger_index > 10)
    trigger_index -= 10;
  else
    trigger_index = 0;
 }



 pt_trigger0[0] = trigger_index;
 pt_freq[0] = freq;
 pt_period[0] = period;

}
```

▼ 表 debug_routines 程式碼

```
#ifdef DEBUG_BUF
void debug_buffer() {
  ADC_Sampling();
  i2s_adc_disable(I2S_NUM_0);
  delay(1000);
  for (uint32_t i = 0; i < B_MULT * NUM_SAMPLES; i++) {
    for (int j = 0; j < 1; j++) {
      Serial.println(i2s_buff[i]);
    }
    delay(5);
  }
  i2s_zero_dma_buffer(I2S_NUM_0);
  i2s_adc_enable(I2S_NUM_0);
  while (1);
}
#endif
```

▼ 表 i2s 程式碼

```
 void configure_i2s(int rate) {
 /*keep in mind:
   dma_buf_len * dma_buf_count * bits_per_sample/8 > 4096
 */
 i2s_config_t i2s_config =
 {
  .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX |
I2S_MODE_ADC_BUILT_IN),  // I2S receive mode with ADC
  .sample_rate = rate,                                    // sample rate
  .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT,                    // 16
bit I2S
  .channel_format = I2S_CHANNEL_FMT_ALL_LEFT,                      //
only the left channel
  .communication_format = (i2s_comm_format_t)(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),   // I2S format
  .intr_alloc_flags = 1,                                  // none
  .dma_buf_count = 2,                                     // number of DMA
buffers
  .dma_buf_len = NUM_SAMPLES,                                  // number of
samples
  .use_apll = 0,                                  // no Audio PLL
 };
 adc1_config_channel_atten(ADC_CHANNEL, ADC_ATTEN_11db);
 adc1_config_width(ADC_WIDTH_12Bit);
 i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);

 i2s_set_adc_mode(ADC_UNIT_1, ADC_CHANNEL);
 SET_PERI_REG_MASK(SYSCON_SARADC_CTRL2_REG,
SYSCON_SARADC_SAR1_INV);
 i2s_adc_enable(I2S_NUM_0);
}

void ADC_Sampling(uint16_t *i2s_buff){
 for (int i = 0; i < B_MULT; i++) {
  //TODO i2s_read_bytes is deprecated, replace with new function
  i2s_read_bytes(I2S_NUM_0, (char*)&i2s_buff[i * NUM_SAMPLES],
NUM_SAMPLES * sizeof(uint16_t), portMAX_DELAY);
 }
}

void set_sample_rate(uint32_t rate) {
 i2s_driver_uninstall(I2S_NUM_0);
 configure_i2s(rate);
}
```