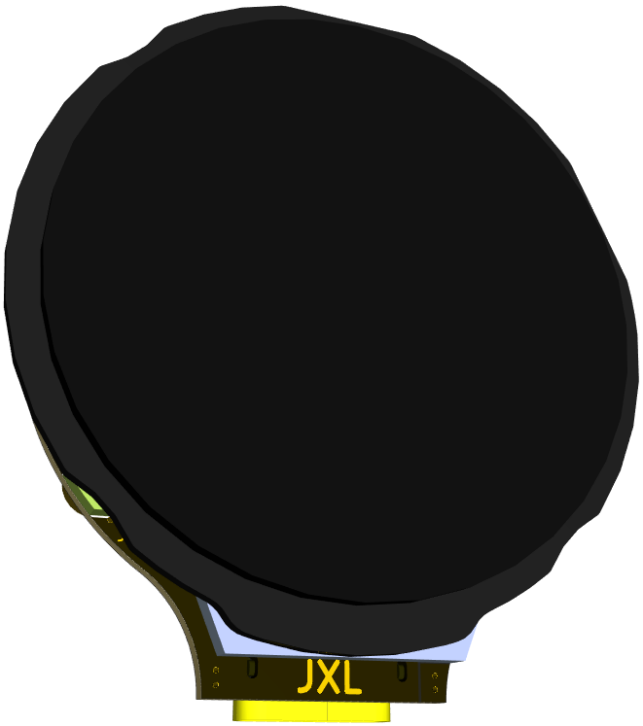
1.28 寸 ESP32S3 圆形 TFT-I80 屏幕开发板

开发参考文档



时间 2025/04/15

版本 V1.0

修订历史

日期	版本	发布说明
2025-04-15	V1.0	● 首次发布

一、 文档说明

开发板主体以 ESP32-S3 为核心，并搭配丰富的外设以及供电等电路设计，开发板通过长按中间按键 3S 实现开关机，本文档展开对开发板各个板块的功能进行分析以便用户进行开发。

文档按照下面几大方面对开发板进行拆解分析，注意代码部分需要配合资料工程 FullFunctionTest 阅读，共同学习进步。

ESP32-S3 核心参数

UPS 供电切换稳压 & 开关机

锂电池充电 & 电量检测

屏幕参数显示 & 电容触摸

TCA6408 扩展 IO

物理按键

QMI8658 陀螺仪

PCF85063 RTC 时钟

TF 卡

IIS 音频输出 & MIC

WS2812 & 扩展 IO

二、 ESP32-S3 核心参数

ESP32-S3 核心部分有 ESP32-S3 芯片，外挂的 FLASH & PSRAM，以及下载模式控制按键与天线几大部分组成。

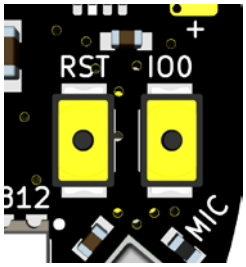
ESP32-S3 芯片运行频率及基本功能等参数不再赘述，可参考官方芯片手册。

本次开发板，外挂有 16MB 的 FLASH,采用四线 SPI 进行连接。

PSRAM 部分为 ESP32-S3 芯片内置，出货有 8MB 和 16MB 两种版本供用户选择,PSRAM 内部为 OPI 连接方式,使用时务必选择 OPI PSRAM，否则将会无法识别。

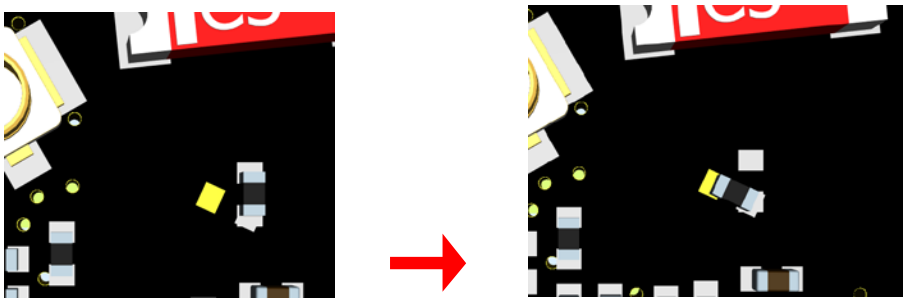
开发板仅有的 USB 口连接至芯片的 IO19 D- 和 IO20 D+ 两个引脚用于下载调试，因此在下载时请选择 USB 下载，而非 UART 下载，而对于想要使用 USB 口进行串口打印的用户，打开 USB CDC On Boot 的功能可以使用 Serial.print 等语法打印调试信息。

对于部分用户会出现烧录时代码选项设置为关闭 USB，这在下载之后将无法再找到 USB 口，电脑也无法再识别到 USB 口，以及关闭 USB 下载功能后识别到 USB 口但无法下载的情况，这种情况需要使用板载的两颗微型物理按键进入下载模式，如下图



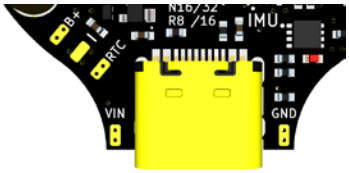
按住 IO0 按键不松，再次按下 RST 按键随后松开 RST 按键，再随后松开 IO0 按键，ESP32-S3 将进入下载模式，值得注意的是此处的 IO0 按键不建议作为常规物理按键去实现功能，因为 IO0 已经作为触摸复位使用。

天线默认采用板载的天线，预留的 IPEX 需要更改电阻切换，外接 IPEX 天线需要将下方 0Ω 电阻更换到另一侧的位置进行焊接。



三、 供电 & 开关机

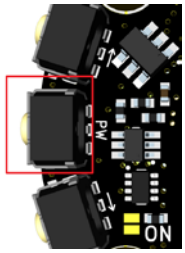
供电部分提供了外部 VIN 输入焊盘，USB，以及 BAT 电池三种供电方式。



其中 VIN 焊盘与 USB 的电源端直接连通，请注意使用，电压请勿超过 5.5V，并且均可为电池进行充电。

USB 供电与电池供电使用两颗理想二极管实现 UPS 切换功能，电流随后经过电源控制开关供电至 TPS63802 升降压芯片，稳压 3.3V 输出给 ESP32-S3 以及其他板块使用。

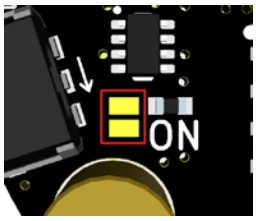
电源控制开关由中间按键进行控制，长按 3 秒以上开机或关机，因此平时可以作为普通按键进行使用。



关机后仅开关控制部分以及电池充电部分的电流消耗，实测关机后静态电流约 7ua。

如果需要上电就可以而不是通过按键控制，那么可以短接下方图

示的两个焊点，短接后上电即开机。



四、 锂电池充电 & 电量检测

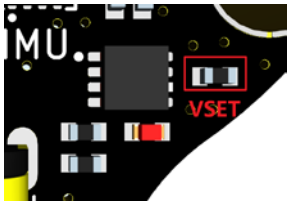
电池可以选择 3.7V 可充电锂电池，充电部分使用 BQ25170 芯片进行充电。

BQ25170 支持以下电池

- 锂离子电池：4.05V、4.1V、4.2V、4.35V、4.4V
- 磷酸铁锂电池：3.5V、3.6V、3.7V

默认设置充电电压为 4.2V，适配标定 3.7V 的锂电池进行使用

依据芯片手册，通过修改下方位置电阻调节

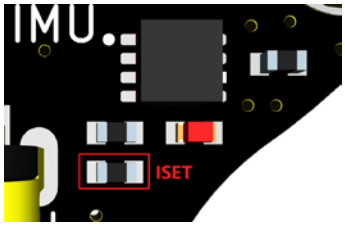


对应关系如下

表 7-1. VSET pin resistor value table

RESISTOR	CHARGE VOLTAGE (V)
> 150 kΩ	No Charge (open-circuit)
100 kΩ	1-cell $LiFePO_4$: 3.50 V
82 kΩ	1-cell $LiFePO_4$: 3.60 V
62 kΩ	1-cell $LiFePO_4$: 3.70 V
47 kΩ	1-cell LiIon: 4.05 V
36 kΩ	1-cell LiIon: 4.10 V
27 kΩ	1-cell LiIon: 4.20 V
24 kΩ	1-cell LiIon: 4.35 V
18 kΩ	1-cell LiIon: 4.40 V
< 3.0 kΩ	No Charge (short-circuit)

最大电流 800ma 可调节，电流默认设置为最大 600ma



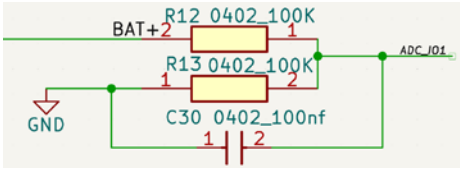
对应关系如下　　$I_{CHG} = K_{ISET}/R_{ISET}$

其中 I_{CHG} 为需要设置的充电电流 K_{ISET} 为增益因子，其值为 270-330 典型值 300，由此可计算出对应充电电流需要使用的电阻 R_{ISET}，默认 499 Ω。

充电指示灯在无电池接入时，接入 USB 口或 VIN 供电指示灯会处于闪烁的状态，接入电池后在充电状态下会常亮，充满后熄灭，充电芯片并非低于 4.2V 就开始充电，而是有一个阈值，低于才开始充电，属于正常现象，具体可参考资料中芯片手册。

值得注意的是 TPS63802 升降压芯片，可以在低至 2V 以下的电压下工作，并且由于没有板载电池过放电路，因为电池选型是最好使用带保护板型号的电池。

电量检测的方式以检测电池电压为主，电池电压通过两个 100K 电阻分压后连接到 ESP32-S3 的 IO1 端口，参考下图即可

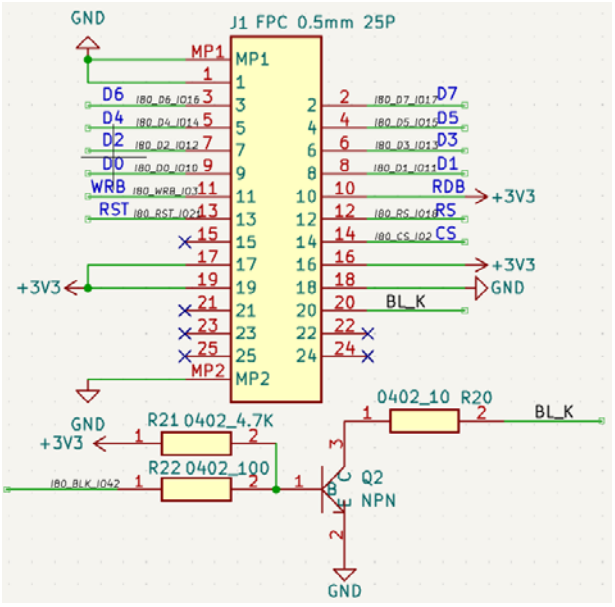


电池未接入时电压会处于浮动状态，上图为参考电路，不会增加开发板静态功耗，静态功耗仍然会在 7ua 左右。

五、 屏幕 & 电容触摸

屏幕为圆形显示区域，1.28in 分辨率 240x240 的 IPS 彩色显示

屏，其驱动为 GC9A01,硬件驱动接口为 I80 接口，部分文档叫 MCU 接口，和 ESP32-S3 的连接可参照下图部分



D0-D7	数据引脚	I010 - 17
RST	复位引脚	I021
WRB	写使能	I03
RS	数据/命令	I018
CS	片选	I02
BLK	背光控制	I042

提供的测试代码使用 Arduino_GFX_Library 的库进行驱动，由于该库存在多个版本，并且不完全兼容，建议使用资料中已经打包好的 1.4.7 版本库，解压后添加到 Arduino IDE 存放库的路径下即可，不做过多赘述

示例代码中

```
#include <Arduino_GFX_Library.h> //添加库文件引用
```

通过下方代码进行初始化

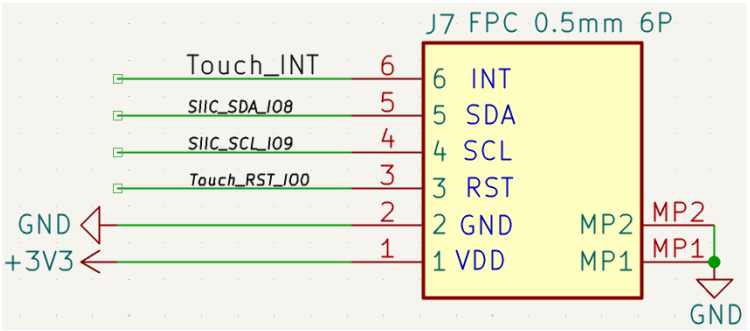
```
#define GFX_BL 42
#define BL_Freq 5000
unsigned int BL_Brightness = 255;
Arduino_DataBus *bus = new Arduino_ESP32LCD8(18 /* DC */, 2 /* CS */, 3 /* WR */, -1 /* RD */,10 /* D0 */, 11 /* D1 */, 12 /* D2 */, 13
/* D3 */, 14 /* D4 */, 15 /* D5 */, 16 /* D6 */, 17 /* D7 */);
Arduino_GFX *gfx = new Arduino_GC9A01(bus, 21 /* RST */, 0 /* rotation */, true /* IPS */);
```

```
gfx ->begin();  //Initialize LCD
gfx->fillScreen(BLACK);  //Background color Black
```

随后即可调用 ARDUINO GFX 库中所提供的函数进行显示驱动

可参照 `gfx->fillScreen(BLACK);` 设置背景色为黑色等功能，具体可查看库作者的 github：https://github.com/moononournation/Arduino_GFX

电容触摸使用 CST816 触摸专用芯片，采用 IIC 驱动，带复位及中断引脚，IIC 器件地址 0x15，与 ESP32-S3 的硬件连接如下图所示



SDA	数据引脚	IO8
SCL	时钟引脚	IO9
RST	复位引脚，低电平有效	IO0
INT	中断引脚	TCA6408 - P0

值得注意 RST 与 INT 引脚，RST 引脚使用 IO0，因此前文 IO0 按键无法当普通按键使用，否则会影响到触摸功能。

中断引脚连接至 TCA6408 扩展芯片的 P0 端口，检测触摸时应该检测 TCA6408 的中断引脚作为中断事件触发，并通过 IIC 检测 P0 端口的

状态来判断是否存在触摸。

示例代码中下方位置定义了 IIC 使用的引脚

```
1  //IIC
2  #define SCL 9
3  #define SDA 8
```

以下代码定义了复位引脚，IIC 器件地址 0x15 以及 ID 寄存器地址与寄存器中值的含义

```
01  //CST816
02  #define TouchRST 0
03  #define TouchI2CAddr 0x15
04
05  #define ChipIdRegister 0xA7
06  #define CST716ChipId 0X20
07  #define CST816SChipId 0XB4
08  #define CST816TChipId 0XB5
09  #define CST816DChipId 0XB6
10  #define CST826ChipId 0X11
11  #define CST830ChipId 0X12
12  #define CST836UChipId 0X13
```

以下代码复位触摸芯片并检测芯片 ID 判断触摸芯片具体型号

```
01      pinMode(TouchRST, OUTPUT);
02      digitalWrite(TouchRST, LOW);
03      delay(10);
04      digitalWrite(TouchRST, HIGH);
05      delay(50);
06
07      Wire.beginTransmission(TouchI2CAddr);
08      Wire.write(ChipIdRegister);
09      Wire.endTransmission(false);
10      Wire.requestFrom(TouchI2CAddr, 1 ,true);
11      ChipID = Wire.read();
```

```
        Serial.printf("\r\nTouchChipID: 0x%02X",ChipID);
        if(ChipID == CST716ChipId) Serial.println(",Touch chip model :CST716");
        else if(ChipID == CST816SChipId) Serial.println(",Touch chip model :CST816S");
        else if(ChipID == CST816TChipId) Serial.println(",Touch chip model :CST816T");
        else if(ChipID == CST816DChipId) Serial.println(",Touch chip model :CST816D");
        else if(ChipID == CST826ChipId) Serial.println(",Touch chip model :CST826");
        else if(ChipID == CST830ChipId) Serial.println(",Touch chip model :CST830");
        else if(ChipID == CST836UChipId) Serial.println(",Touch chip model :CST836U");
        else Serial.println(",error!");
```

以下代码开启 TCA6408 中断事件检测

```
pinMode(TCA6408Int,INPUT_PULLUP);
//Registering interrupt service function
attachInterrupt(digitalPinToInterrupt(TCA6408Int), TCA6408HandleInterrupt, FALLING);
```

中断检测后的执行函数及触摸检测的函数

```
void TCA6408HandleInterrupt(void)
{
 TCA6408EventFlag = true;
 }
```

```
void my_AllInt()
{
 if(TCA6408EventFlag){
  int TCA6408IntValue = 0;
  Wire.beginTransmission(TCA6408I2CAddr);
  Wire.write(TCA6408InputPortReg);
  Wire.endTransmission(false);
  Wire.requestFrom(TCA6408I2CAddr, 1 ,true);
  TCA6408IntValue = Wire.read();

  if((TCA6408IntValue & 0x01) == 0x00) {TouchEventFlag = true; Serial.print("\r\nTouch Int");}
  if((TCA6408IntValue & 0x02) == 0x00) {Serial.print("\r\nIMU Int1");}
  if((TCA6408IntValue & 0x04) == 0x00) {Serial.print("\r\nIMU Int2");}
  if((TCA6408IntValue & 0x08) == 0x00) {Serial.print("\r\nSW UP Int");}
  if((TCA6408IntValue & 0x10) == 0x00) {Serial.print("\r\nSW PW Int");}
  if((TCA6408IntValue & 0x20) == 0x00) {Serial.print("\r\nSW Down Int");}
  if((TCA6408IntValue & 0x40) == 0x00) {Serial.print("\r\nCharging...");}
  if((TCA6408IntValue & 0x80) == 0x00) {Serial.print("\r\nRTC Int");}

  TCA6408EventFlag = false;
  }
 }
```

坐标检测函数

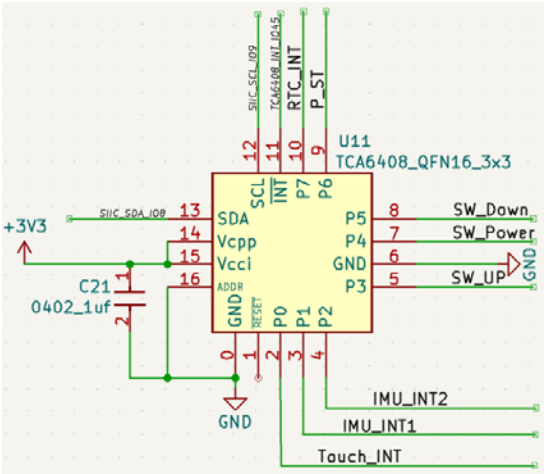
```
void my_touch_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data){
 //Store the pressed coordinates and status
 lv_coord_t last_x = 0;
 lv_coord_t last_y = 0;
 unsigned int X_H4 = 0;
 unsigned int X_L8 = 0;
 unsigned int Y_H4 = 0;
 unsigned int Y_L8 = 0;
 //True if there is touch, false otherwise
 if(TouchEventFlag) {
   Wire.beginTransmission(TouchI2CAddr);
   Wire.write(0x03);
   Wire.endTransmission(false);
   //Wire.beginTransmission(TouchI2CAddr);
   Wire.requestFrom(TouchI2CAddr, 1 ,true);
   X_H4 = Wire.read();

   Wire.beginTransmission(TouchI2CAddr);
   Wire.write(0x04);
   Wire.endTransmission(false);
   //Wire.beginTransmission(TouchI2CAddr);
   Wire.requestFrom(TouchI2CAddr, 1 ,true);
   X_L8 = Wire.read();

   Wire.beginTransmission(TouchI2CAddr);
   Wire.write(0x05);
   Wire.endTransmission(false);
   //Wire.beginTransmission(TouchI2CAddr);
   Wire.requestFrom(TouchI2CAddr, 1 ,true);
   Y_H4 = Wire.read();

   Wire.beginTransmission(TouchI2CAddr);
   Wire.write(0x06);
   Wire.endTransmission(false);
   //Wire.beginTransmission(TouchI2CAddr);
   Wire.requestFrom(TouchI2CAddr, 1 ,true);
   Y_L8 = Wire.read();
```

```
    last_x = 0xFF - (X_H4 << 8 | X_L8)&0X0FFF;
    last_y = (Y_H4 << 8 | Y_L8)&0X0FFF;
    data->point.x = last_x;
    data->point.y = last_y;

    TouchEventFlag = false;
    //Serial.printf("Touch Point:%02X , %02X \r\n",last_x,last_y);
    data->state = LV_INDEV_STATE_PR;
  }
  else {
    data->state = LV_INDEV_STATE_REL;
  }
}
```

注意以上代码需要搭配完整代码使用，并非直接复制使用的，仅作为代码分析学习共同进步的作用

六、 TCA6408 扩展 IO

TCA6408 扩展芯片可通过 IIC 通信扩展出 8 个支持输入检测和输出功能的 IO 口，并且带中断触发功能，在开发板上 8 个 IO 均作为输入检测的功能，IIC 器件地址 0x20，与 ESP32-S3 以及各端口功能参考下表或图。



SCL	时钟引脚	IO9
SDA	数据引脚	IO8

INT	中断引脚，触发后低电平脉冲	IO45
P0	屏幕触摸中断，已上拉	--
P1	QMI8658-IMU 中断 1，已上拉	--
P2	QMI8658-IMU 中断 1，已上拉	--
P3	侧面上方按键，已上拉	--
P4	侧面中间按键，已上拉	--
P5	侧面下方按键，已上拉	--
P6	USB 插入/充电检测，已上拉	--
P7	RTC 时钟中断，已上拉	--

TCA6408 芯片由 4 个寄存器控制，如下所示

寄存器	寄存器地址	说明
输入端口	0x00	用于读取端口电平
输出端口	0x01	控制端口输出电平
极性反转	0x02	设置 1 输出电平反转
配置	0x03	1 高阻抗输入启用 0 输出启用

开发板上，我们仅需使用输入端口寄存器进行状态读取即可，配置寄存器默认为 1，即输入检测的状态可设置可不设置。

P0-P7 端口所连接的输入设备/端口均已上拉电阻，默认为高电平，读取为 0xFF，当触发后为低电平，同时 INT 将由高电平转为低电平输出，直到清除中断

基础驱动思路如下

注册中断事件

中断事件触发

查询端口状态

判断具体触发事件

示例中下方代码段定义了中断/IIC 引脚，TCA6408 的 IIC 器件地址，以及寄存器和标志位相关信息

```
//IIC
#define SCL 9
#define SDA 8
```

```
//TCA6408
#define TCA6408Int 45
#define TCA6408I2CAddr 0x20

#define TCA6408ConfigurationReg 0x03
#define TCA6408ConfigurationData 0xFF
#define TCA6408InputPortReg 0x00

Ticker TCA6408InterruptTicker;
volatile bool TCA6408EventFlag = false;
volatile bool TouchEventFlag = false;
```

下方代码段为初始化 TCA6408 内容

```
void TCA6408Init(){
 int TCA6408TempData = 0;

 Wire.beginTransmission(TCA6408I2CAddr);
 Wire.write(TCA6408ConfigurationReg);
 Wire.write(0x55);
 Wire.endTransmission(true);
 delay(10);
 Wire.beginTransmission(TCA6408I2CAddr);
 Wire.write(TCA6408ConfigurationReg);
 Wire.endTransmission(false);
 Wire.requestFrom(TCA6408I2CAddr, 1 ,true);
 TCA6408TempData = Wire.read();

 if(0x55 == TCA6408TempData) Serial.print("\r\nTCA6408 pass!");
 else Serial.print("\r\nTCA6408 fail!");
 delay(10);
 Wire.beginTransmission(TCA6408I2CAddr);
 Wire.write(TCA6408ConfigurationReg);
 Wire.write(TCA6408ConfigurationData);
 Wire.endTransmission(true);

 delay(10);
 }
```

中断事件注册

```
pinMode(TCA6408Int,INPUT_PULLUP);
//Registering interrupt service function
attachInterrupt(digitalPinToInterrupt(TCA6408Int), TCA6408HandleInterrupt, FALLING);
```

```
void TCA6408HandleInterrupt(void)
{
 TCA6408EventFlag = true;
 }
```

查询触发事件

```
void my_AllInt()
{
 if(TCA6408EventFlag)
 {
  int TCA6408IntValue = 0;
  Wire.beginTransmission(TCA6408I2CAddr);
  Wire.write(TCA6408InputPortReg);
  Wire.endTransmission(false);
  Wire.requestFrom(TCA6408I2CAddr, 1 ,true);
  TCA6408IntValue = Wire.read();

  if((TCA6408IntValue & 0x01) == 0x00) {TouchEventFlag = true; Serial.print("\r\nTouch Int");}
  if((TCA6408IntValue & 0x02) == 0x00) {Serial.print("\r\nIMU Int1");}
  if((TCA6408IntValue & 0x04) == 0x00) {Serial.print("\r\nIMU Int2");}
  if((TCA6408IntValue & 0x08) == 0x00) {Serial.print("\r\nSW UP Int");}
  if((TCA6408IntValue & 0x10) == 0x00) {Serial.print("\r\nSW PW Int");}
  if((TCA6408IntValue & 0x20) == 0x00) {Serial.print("\r\nSW Down Int");}
  if((TCA6408IntValue & 0x40) == 0x00) {Serial.print("\r\nCharging...");}
  if((TCA6408IntValue & 0x80) == 0x00) {Serial.print("\r\nRTC Int");}

  TCA6408EventFlag = false;
 }
}
```

通过上面扩展的 IO 口，可以通过 TCA6408 芯片做到以下的几大功能

屏幕触摸的中断触发事件

惯性测量单元的中断触发事件

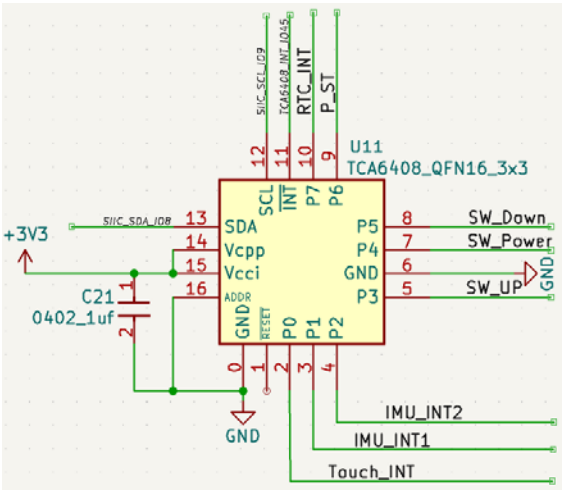
3 个物理按键中断触发事件

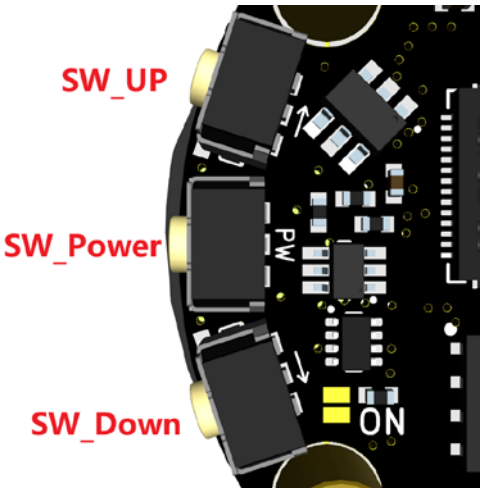
USB 插入或电池是否处于充电中的状态检测

RTC 时钟芯片的中断触发以实现低功耗定时唤醒的功能

七、 物理按键

开发板板载有 5 个物理按键，其中两个模式控制及复位不作为普通按键使用，三个侧面按键连接至 TCA6408 的 P3　P4　P5,按键连接参考与位置如图所示





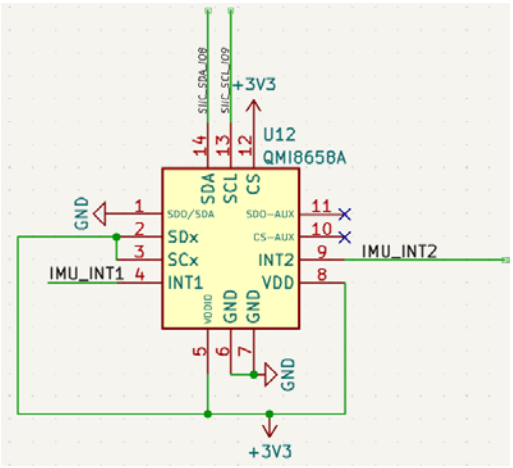
值得一说的是中间的按键不仅作为普通按键使用还承担了电源开关的功能，长按 3 秒将会让开发板开机或关机，当然，再次说明可以短接下方 ON 位置焊盘来取消开关机的功能。

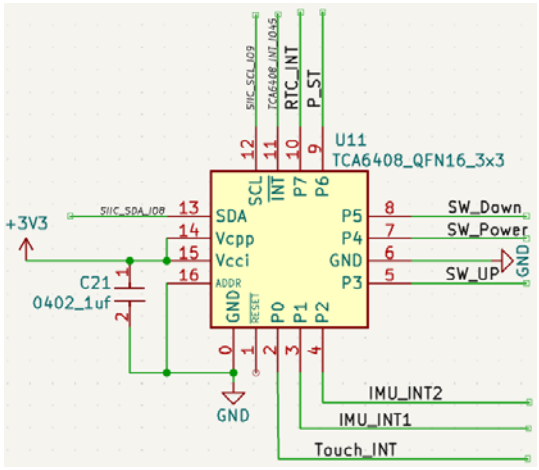
对于 3 个按键的使用，可以参考前文 TCA6408 扩展 IO 中的内容，因为三个按键均连接至 TCA6408，因此对 3 个按键的应用开发基本等同于对 TCA6408 的应用。

八、 QMI8658 陀螺仪

QMI8658 是一颗六轴惯性测量单元芯片，具有三轴陀螺仪、三轴加速度计，足够进行姿态以及运动状态检测，通过 IIC 与 ESP32-S3 进行通信，带两个中断引脚，分别对应陀螺仪的中断与加速度计的中断供用户使用。

其 IIC 器件地址为 0x6B，开发板上连接可参照下图或表





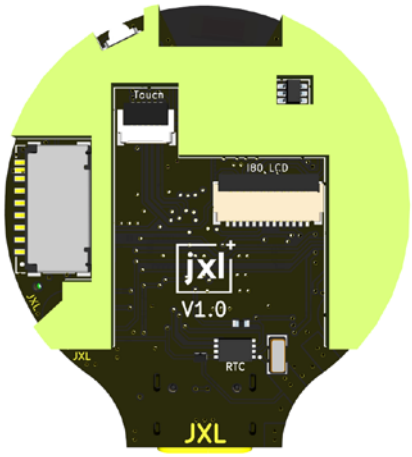
SCL	时钟引脚	IO9
SDA	数据引脚	IO8
IMU_INT1	中断引脚 1	连接至 TCA6408 - P1
IMU_INT2	中断引脚 2	连接至 TCA6408 - P2

驱动代码可参考资料工程 FullFunctionTest 以及芯片手册进行阅读，足够了解该类型芯片的用户可以自行编码驱动，或者选型移植示例中的代码，本代码中 IMU 部分同样移植自微雪的示例，节省大量时间，感谢微雪电子，同时也建议移植已验证代码去使用。

九、 PCF85063 RTC 时钟

板载的 RTC 时钟电路没有与 ESP32-S3 在同一面，因此不必为没有在开发板上看到相关电路而困惑。

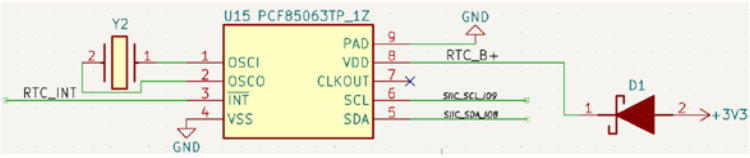
如下图所示为 PCB 背面没有屏幕时的状态



PCF85063 这颗 RTC 时钟芯片同样采用 IIC 进行通信，得益于 IIC 总线的特性，他与开发板上其他 IIC 器件一样均共用一组 IIC 引脚，用于设置或读取时间等参数。

除此之外这颗芯片拥有 INT 引脚，我们将他连接到了 TCA6408 扩展芯片的 P7 引脚，INT 引脚设置半分钟或者一分钟中断，可以在 ESP32-S3 进入低功耗的状态下通过这个中断的状态进行唤醒，当然也可以用于其他的有趣的方案。

芯片于开发板其他部件的连接可以参考下图或表



SCL	时钟引脚	IO9
SDA	数据引脚	IO8
INT	RTC 中断引脚	连接至 TCA6408 - P7

对于电源部分，RTC 的电源显然是非常重要的，板载有 RTC 电池的连接端口，推荐使用 3V 电压的可充电电池，在 RTC 电池未接入时芯片通过板载的 3.3V 稳压获取电源，RTC 电池接入后 3.3V 将会给 RTC 电池进行充电,在开发板关机时芯片将由 RTC 电池进行续电工作，

静态电流可以参考芯片手册。

PCF85063 拥有 11 个寄存器，每个寄存器有 8 个位，可以分为两类，一类为配置参数，另一类为时间参数。

Address	Register name	Bit								Reference
		7	6	5	4	3	2	1	0	
Control and status registers										
00h	Control_1	EXT_TEST	-	STOP	SR	-	CIE	12_24	CAP_SEL	Section 8.2.1
01h	Control_2	-	-	MI	HMI	TF	COF[2:0]			Section 8.2.2
02h	Offset	MODE	OFFSET[6:0]							Section 8.2.3
03h	RAM_byte	B[7:0]								Section 8.2.4
Time and date registers										
04h	Seconds	OS	SECONDS (0 to 59)							Section 8.3.1
05h	Minutes	-	MINUTES (0 to 59)							Section 8.3.2
06h	Hours	-	-	AMPM	HOURS (1 to 12) in 12 hour mode					Section 8.3.3
				HOURS (0 to 23) in 24 hour mode						
07h	Days	-	-	DAYS (1 to 31)						Section 8.3.4
08h	Weekdays	-	-	-	-	-	WEEKDAYS (0 to 6)			Section 8.3.5
09h	Months	-	-	-	MONTHS (1 to 12)					Section 8.3.6
0Ah	Years	YEARS (0 to 99)								Section 8.3.7

0x00 - 0x03 寄存器储存了配置参数，在需要设置的位中

0x00 寄存器需要关注位 1，用于设置 12 小时或 24 小时制，写入 0 为 24 小时制，1 对应 12 小时制，默认为 0。

Bit	Symbol	Value	Description	Reference
7	EXT_TEST		**external clock test mode**	Section 8.2.1.1
		0[1]	normal mode	
		1	external clock test mode	
6	-	0	unused	-
5	STOP		**STOP bit**	Section 8.2.1.2
		0[1]	RTC clock runs	
		1	RTC clock is stopped; all RTC divider chain flip-flops are asynchronously set logic 0	
4	SR		**software reset**	Section 8.2.1.3
		0[1]	no software reset	
		1	initiate software reset[2]; this bit always returns a 0 when read	
3	-	0	unused	-
2	CIE		**correction interrupt enable**	Section 8.2.3
		0[1]	no correction interrupt generated	
		1	interrupt pulses are generated at every correction cycle	
1	12_24		**12 or 24 hour mode**	Section 8.3.3
		0[1]	24 hour mode is selected	
		1	12 hour mode is selected	
0	CAP_SEL		**internal oscillator capacitor selection** for quartz crystals with a corresponding load capacitance	-
		0[1]	7 pF	
		1	12.5 pF	

0x01 寄存器用于设置分钟/半分钟中断，其中位 4 对应半分钟中断，默认为 0 关闭，位 5 对应分钟中断，默认为 0 关闭。

Bit	Symbol	Value	Description
7 to 6	-	00	unused
5	MI		**minute interrupt**
		0[1]	disabled
		1	enabled
4	HMI		**half minute interrupt**
		0[1]	disabled
		1	enabled
3	TF		**timer flag**
		0[1]	no timer interrupt generated
		1	flag set when timer interrupt generated
2 to 0	COF[2:0]	see Table 11	**CLKOUT control**

0X02 寄存器用于设置偏移量，在运行中芯片会由于晶振精度，负载电容精度，老化以及温漂等因素难以做到 0 偏差，此寄存器用于纠正此偏差以达到更精准的计时。

寄存器位 0 到位 6 存储偏移的值，位 7 - MODE 为偏移模式设置，

Bit	Symbol	Value	Description
7	MODE		**offset mode**
		0[1]	normal mode: offset is made once every two hours
		1	course mode: offset is made every 4 minutes
6 to 0	OFFSET[6:0]	see Table 13	**offset value**

当 MODE 设置为 0 时为正常模式，每两小时调整一次偏移量，此时存储的偏移量为 1 个单位 4.34ppm，也就是说具体的偏移量需要将位 0-6 里面存储的值*4.34，单位 ppm。

如果 MODE 设置为了 1，那么芯片将会每 4 分钟调整一次偏移，此时的偏移量为 1 个单位 4.069ppm。

位 1 到 6 中存储的值为补码值，需要进行换算，可以以参照下表或芯片手册查看更详细信息。

OFFSET[6:0]	Offset value in decimal	Offset value in ppm	
		Normal mode MODE = 0	Fast mode MODE = 1
0111111	+63	+273.420	+256.347
0111110	+62	+269.080	+252.278
:	:	:	:
0000010	+2	+8.680	+8.138
0000001	+1	+4.340	+4.069
0000000[1]	0	0[1]	0[1]
1111111	−1	−4.340	−4.069
1111110	−2	−8.680	−8.138
:	:	:	:
1000001	−63	−273.420	−256.347
1000000	−64	−277.760	−260.416

0x03 寄存器为芯片预留的空闲字节，有一个 8 个位的空间，可以用于存储部分例如状态等信息的数据，当接入 RTC 电池时可实现开发板掉电保存数据。

Bit	Symbol	Value	Description
7 to 0	B[7:0]	00000000[1] to 11111111	**RAM content**

0x04-0x0A 寄存器用于储存时间相关的数据，详细信息可参考芯片数据手册，简介如下。

0x04 寄存器存储秒钟数据。

Bit	Symbol	Value	Place value	Description
7	OS			**oscillator stop**
		0	-	clock integrity is guaranteed
		1[1]	-	clock integrity is not guaranteed; oscillator has stopped or has been interrupted
6 to 4	SECONDS	0[1] to 5	ten's place	**actual seconds** coded in BCD format, see Table 20
3 to 0		0[1] to 9	unit place	

0x05 寄存器存储分钟数据。

Bit	Symbol	Value	Place value	Description
7	-	0	-	unused
6 to 4	MINUTES	0[1] to 5	ten's place	**actual minutes** coded in BCD format
3 to 0		0[1] to 9	unit place	

0x06 寄存器存储小时数据，分为 12 小时制和 24 小时制，存在两种数据获取方法，在 0x00 寄存器中设置小时制。

Bit	Symbol	Value	Place value	Description
7 to 6	-	00	-	unused
12 hour mode[1]				
5	AMPM			**AM/PM indicator**
		0[2]	-	AM
		1	-	PM
4	HOURS	0[2] to 1	ten's place	**actual hours in 12 hour mode** coded in BCD format
3 to 0		0[2] to 9	unit place	
24 hour mode[1]				
5 to 4	HOURS	0[2] to 2	ten's place	**actual hours in 24 hour mode** coded in BCD format
3 to 0		0[2] to 9	unit place	

0x07 寄存器存储日期/天数据，注意设置月和年，此处并非按照一月 30 天计算，而是区分了 30/31 天的大小月，并且包括对闰年自动计算 2 月的天数。

Bit	Symbol	Value	Place value	Description
7 to 6	-	00	-	unused
5 to 4	DAYS[1]	0[2] to 3	ten's place	**actual day** coded in BCD format
3 to 0		0[3] to 9	unit place	

0x08 寄存器存储周/星期数据，注意数据 0 为周日/天，星期七并非对应数据 7

Bit	Symbol	Value	Description
7 to 3	-	00000	unused
2 to 0	WEEKDAYS	0 to 6	**actual weekday** values, see Table 25

0x09 寄存器存储月数据

Bit	Symbol	Value	Place value	Description
7 to 5	-	000	-	unused
4	MONTHS	0 to 1	ten's place	**actual month** coded in BCD format, see Table 27
3 to 0		0 to 9	unit place	

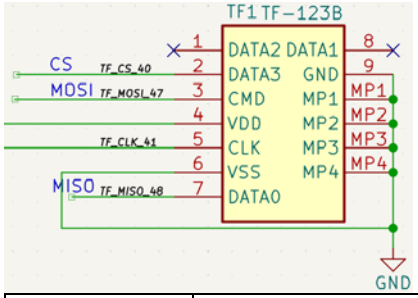
0x0A 寄存器存储年数据

Bit	Symbol	Value	Place value	Description
7 to 4	YEARS	0[1] to 9	ten's place	**actual year** coded in BCD format
3 to 0		0[1] to 9	unit place	

对于 RTC 的使用，注意设置年月信息，即使用不上，对于偏移量的设置，需要运行固定时间时候对比标准时间的偏差才能计算出需要设置的偏移量，并不能在未测试的情况下直接设置，具体可参考芯片手册更加详细说明。

十、 TF 卡

TF 卡座采用 SPI 的连接方式和 ESP32-S3 进行连接，因此需要使用支持 SPI 协议的 TF 卡，市面上绝大部分卡都支持，但也买到过部分低价不支持的 SPI 协议的 TF 卡，IO 连接可参考下图或表



CS	SPI 片选引脚	IO40
CLK	SPI 数据引脚	IO41
MOSI	SPI 数据引脚	IO47
MISO	SPI 数据引脚	IO48

TF 卡可以借助 #include <SD.h> 的库文件 以及 File vFile; 文件系统进行操作，可以参考示例代码中播放 TF 卡视频的部分。

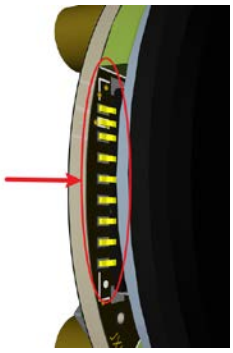
初始化并判断 TF 卡是否存在

```
pinMode(SD_CS,INPUT_PULLUP);
delay(10);
SPI.begin(SCK, MISO, MOSI, SD_CS);
delay(10);
if (!SD.begin(SD_CS, SPI, 80000000))  //1-bit SD bus mode
```

播放视频

```
{
 uint8_t *mjpeg_buf = (uint8_t *)ps_malloc(MJPEG_BUFFER_SIZE);
 Start:
 vFile = SD.open(MJPEG_FILENAME);
 if (!vFile || vFile.isDirectory())
 {
  Serial.println(F("ERROR: Failed to open " MJPEG_FILENAME " file for reading"));
  vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(1000));
  esp_restart();
  }
 else
 {
  mjpeg.setup(&vFile, mjpeg_buf, displayBack , true , 0 , 0 , gfx->width() /* widthLimit */, gfx->height());
  Serial.println(F("MJPEG video start"));
  while (vFile.available() && mjpeg.readMjpegBuf())
  {
   // Play video
   mjpeg.drawJpg();
   //Serial.printf("\r\nmainTask: %d", uxTaskGetStackHighWaterMark(NULL));
   vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(33));
   }
  vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(10));
  Serial.println(F("MJPEG video end"));
  vFile.close();
  goto Start;
  }
}
```

TF 卡位置处于 PCB 板与屏幕之间，设计外壳需要注意开孔，因为不在 PCB 正面容易观察到，设计时容易将其遗漏。



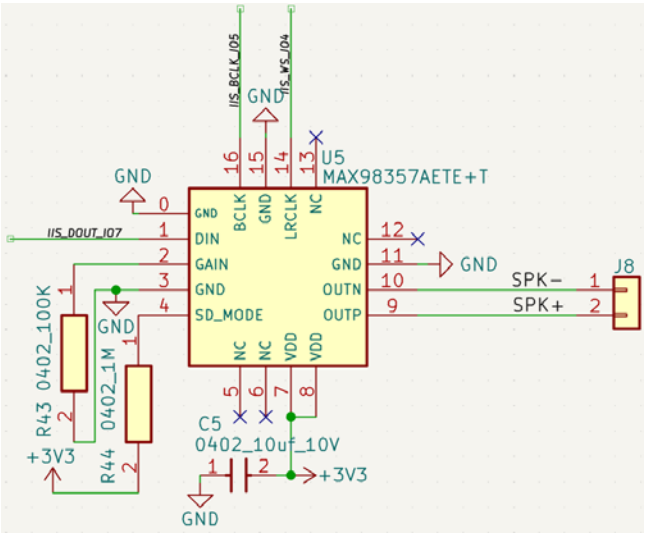
TF 卡安装时有金属触点一点朝向底部(屏幕显示的相反面)。

十一、 IIS 音频输出 & MIC

音频输出与 MIC 的音频采集，均采用 IIS 进行驱动连接，采用 Duplex I2S 模式与 ESP32 进行连接，即共用 IIS 的时钟和左右声道引脚。

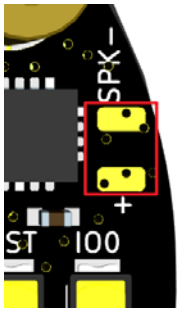
IIS 音频输出采用 MAX98357 芯片，最大输出功率 3W，按实际使用的效果来看，建议使用 8Ω 的喇叭。

与 ESP32-S3 连接可参考下图或表



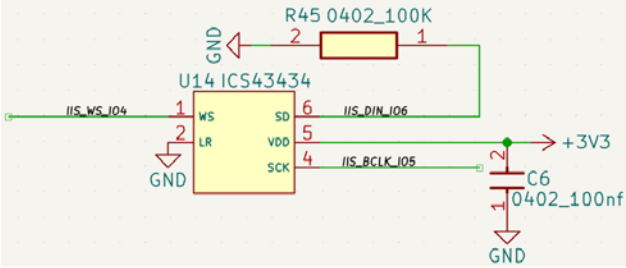
WS	左右声道	IO4
BCLK	时钟引脚	IO5
DIN	数据引脚	IO7

喇叭连接位置如下图所示，需要将喇叭线焊接至对应焊盘。



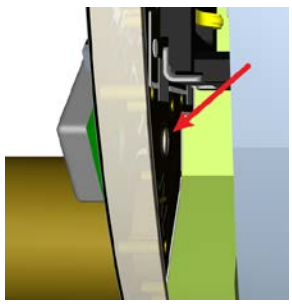
如果需要保持小体积，建议使用手机以及手表上使用的喇叭型号，配合外壳做成腔体的情况下可以实现较大音量的输出，需要注意做成腔体很重要，将会在很大程度上影响输出声音的大小。

音频采集使用的 ICS43434，一颗常用的 IIS 麦克风，与 ESP32-S3 连接可参考下图或表



WS	左右声道	IO4
BCLK	时钟引脚	IO5
DOUT	数据引脚	IO6

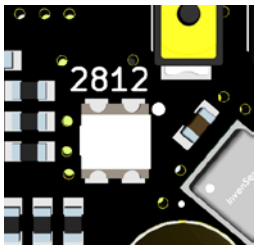
在开发板侧面有一个缺口用于采集声音，请勿将这个缺口堵塞，即使制作外壳也需要对这个位置开孔。



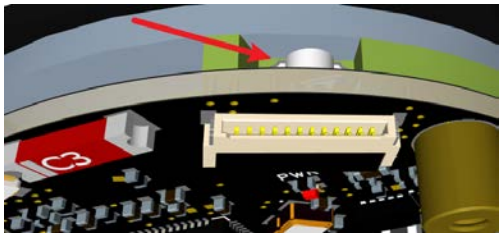
对于 IIS 音频的采集与输出，示例代码以播放 MJPEG 文件为主，没有音频信号，目前测试使用 github 上开源小智 AI 代码移植后进行测试，也可以直接烧录我们移植好的，感谢小智 AI 作者开源如此优质的项目，开源地址如下 https://github.com/78/xiaozhi-esp32

十二、 WS2812 & 扩展 IO

开发板板载有两颗 WS2812-RGB 灯，通过单总线连接，通过 ESP32-S3 的 46 号引脚进行驱动，其中第一颗在麦克风边上，位置如下



第二颗与第一颗串联连接，位置在 PCB 板与屏幕之间(PCB 板的反面)，如下图所示



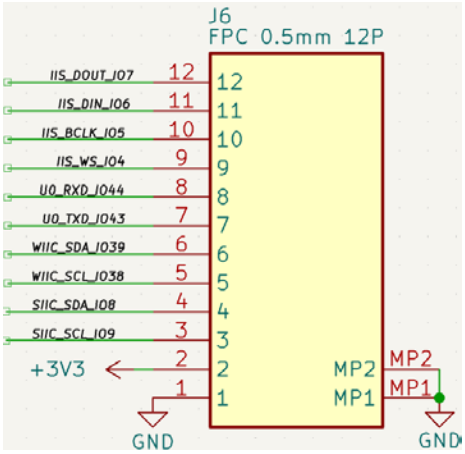
示例代码中使用 #include <Adafruit_NeoPixel.h> 库进行驱动，下面代码添加头文件并设置 WS2812 使用引脚以及数量

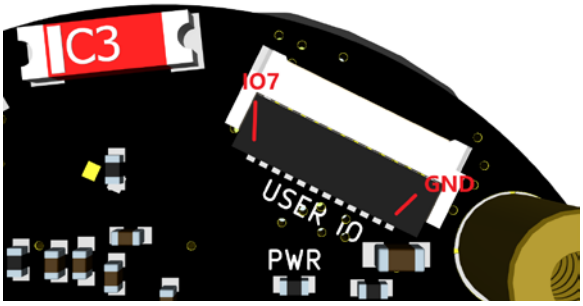
```
#include <Adafruit_NeoPixel.h>


#define WS2812Pin 46
#define WS2812_Count 2
Adafruit_NeoPixel strip(WS2812_Count, WS2812Pin, NEO_GRB + NEO_KHZ800);
```

初始化并设置初始颜色

```
void WS2812Init(){
 strip.begin();  //Initializing the WS2812
 strip.clear();  //Clear All LEDs
 strip.show();  //Turn off all lights
 //Set the first light to red (R, G, B)
 strip.setPixelColor(0, strip.Color(20, 0, 0));
 //Set the second light to green
 strip.setPixelColor(1, strip.Color(0, 20, 0));
 strip.show();  //Send data to update the lamp beads
 }
```

开发板板载有扩展接口，引脚顺序如下所示





参考规划中如下

IO7/6/5/4	IIS 使用	不可用或无需音频功能可当 IO 口使用
IO44/43	UART0	44 - RXD　　43-TXD　　外接 GPS/4G 等
IO39/38	IIC2/IO*2	普通 IO 或第二组 IIC 功能的启用
IO8/9	IIC1	8 - SDA　　9-SCL 与开发板传感器共同 注意地址冲突

至此，开发板基本功能简介均在以上文档中，欢迎向我们提交 BUG 以便我们修复或将产品做到更好。