EAIDK-610 公开教程

Python 版

2020/07/22

OPEN AI LAB

版本 V0.1.1

目录(catalog)

1 前言

1.1 目的

本文档主要介绍 EAIDK-610 产品基本功能，硬件特性，软件特点及软件调试操作方法，旨在帮助开发人员更快、更好熟悉 EAIDK-610 硬件平台（EAIDK-610）和使用 EAIDK-610 开发套件。

1.2 术语

- **EAIDK**：Embedded AI Development Kit。嵌入式人工智能开发套件。
- **AID**：AID 是 OPEN AI LAB 开发的一个面向嵌入式平台前端智能，跨 SoC 的 AI 核心软件平台。
- **GPIO**：通用型之输入输出。
- **IIC**：Inter-Integrated Circuit（集成电路总线）
- **SPI**：Serial Peripheral Interface（串行外设接口）
- **bme280**：博世气压，温湿度三合一传感器

2 硬件介绍

2.1 硬件总览

EAIDK-610 采用 8 层板设计，沉金工艺。正面如图 2-1 所示，背面如图 2- 2 所示。

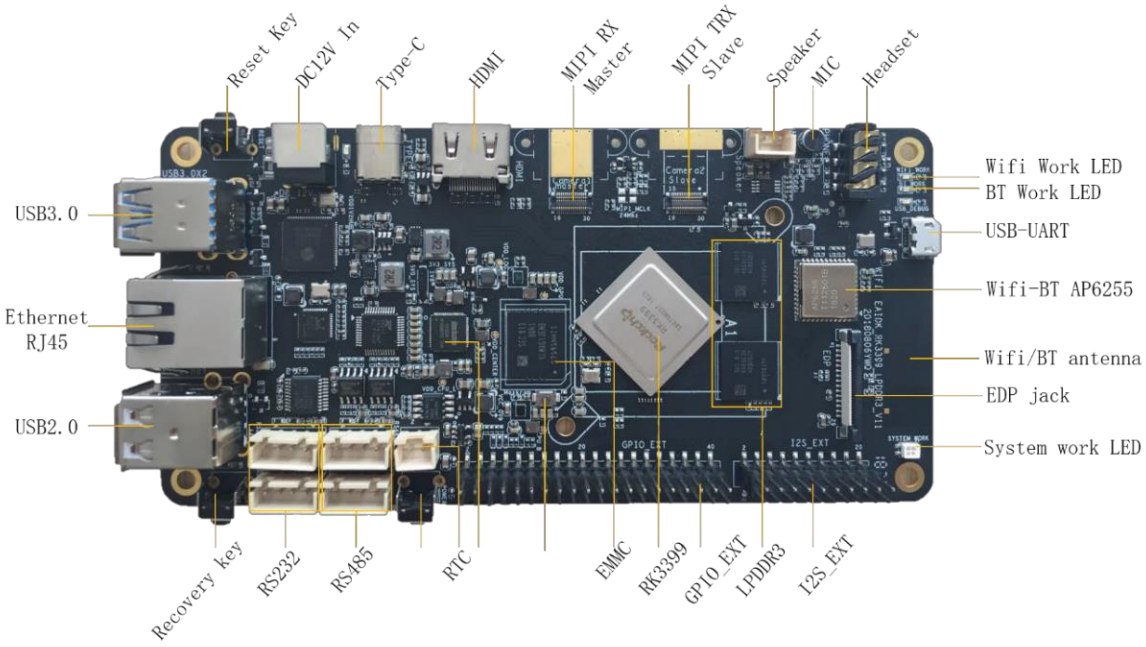

图 2-1 Top Layer 接口图

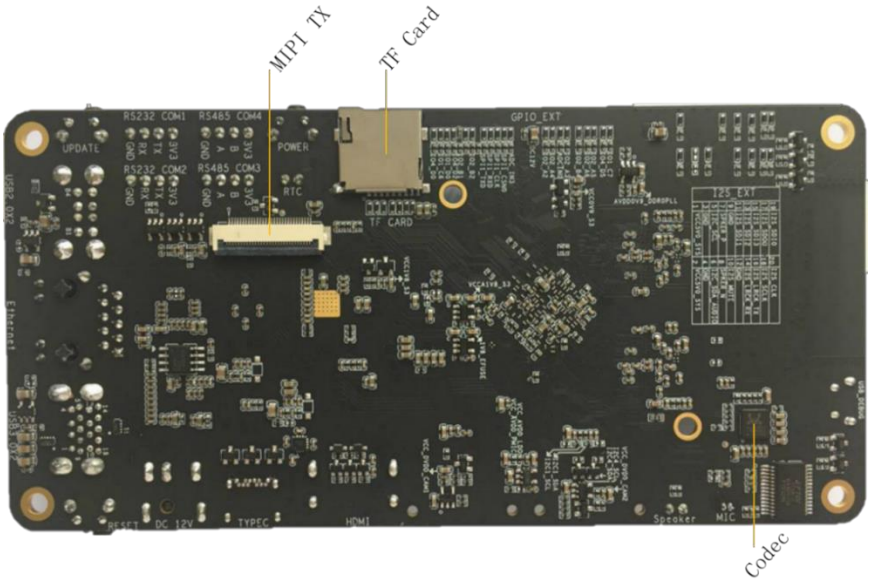

图 2-2 Bottom Layer 接口图

2.2 调试接口

开发板提供调试串口供开发调试使用。调试串口连接主控的 UART2 接口，通过板上集成 FT232RL UART 转 USB 接口转换芯片，外接 Micro USB 座子。用户只需要一个普通 Micro USB 线即可。

说明：EAIDK-610 的调试串口的波特率为 1500000。

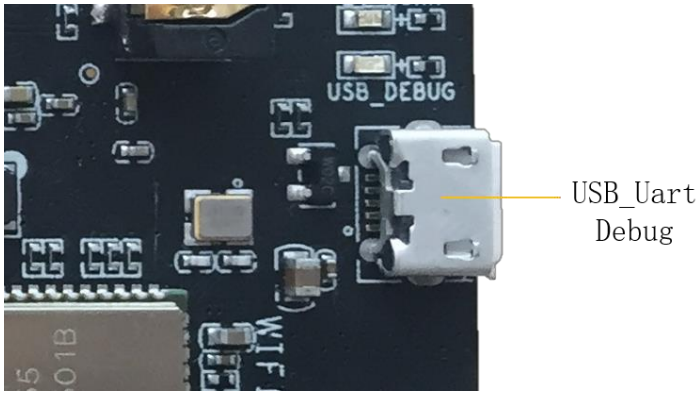

图 2-3 USB UART Debug 接口示意图

2.3 电源模块

EAIDK-610 开发板的电源模块采用 PMIC RK808 为核心芯片，配合外围的 Buck、LDO 组成。

图 2-4 DC 输入接口示意图

2.4 存储模块

2.4.1 内存

EAIDK-610 开发板采用两颗 32bit 2GB LPDDR3 颗粒，构成 64bit 4GB DDR。

图 2-5 LPDDR3 位置示意图

2.4.2 EMMC

EAIDK-610 开发板采用 EMMC 作为系统盘，默认容量 16GB。

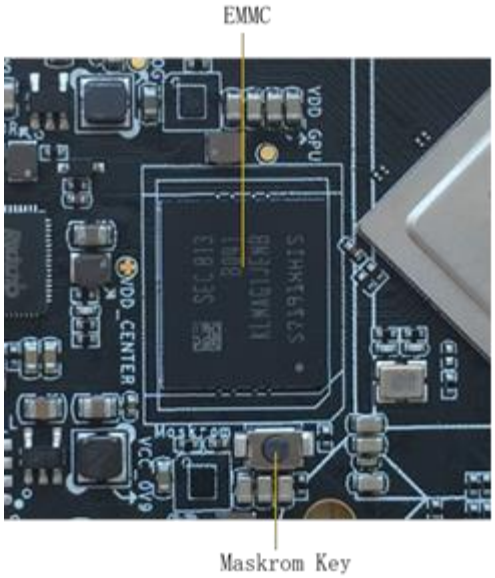

图 2-6 EMMC 位置示意图

2.4.3 TF 卡

EAIDK-610 开发板带有 TF Card 卡座，连接 RK3399 SDMMC0。数据总线宽带为 4bit，支持热插拔。

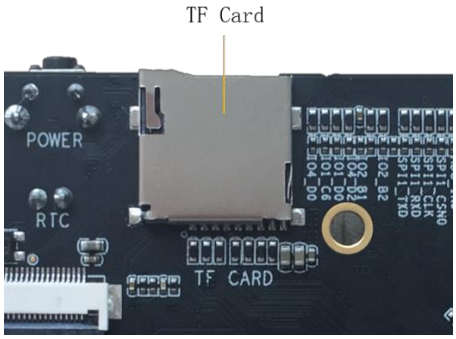

图 2-7 TFcard 位置示意图

2.5 显示模块

2.5.1 MIPI 显示

EAIDK-610 开发板标配显示屏为 5.5 寸 720P MIPI 显示屏，支持 5 点触摸。

图 2-8 MIPI-TX 连接座位置示意图

MIPI 管脚定义如下表所示：

表 2-1 MIPI_TX 管脚定义表

Pin	Name	Pin	Name
1	GND	**16**	GND
2	MIPI_TX0_D0N	**17**	LCD_BL_PWM
3	MIPI_TX0_D0P	**18**	NC
4	GND	**19**	NC
5	MIPI_TX0_D1N	**20**	LCD_RST_H
6	MIPI_TX0_D1P	**21**	GND
7	GND	**22**	LCD_EN_H
8	MIPI_TX0_CLKN	**23**	I2C_SCL_TP
9	MIPI_TX0_CLKP	**24**	I2C_SDA_TP
10	GND	**25**	TOUCH_INT_L
11	MIPI_TX0_D2P	**26**	TOUCH_RST_L
12	MIPI_TX0_D2N	**27**	GND
13	GND	**28**	VCC5V0_SYS
14	MIPI_TX0_D3N	**29**	VCC5V0_SYS
15	MIPI_TX0_D3P	**30**	VCC5V0_SYS

2.5.2 eDP 显示

EAIDK-610 开发板可选配件为 7.85 寸 2K eDP 显示屏，支持 5 点触摸。

图 2-9 EDP 连接示意图

eDP 管脚定义如下表所示：

表 2-2 EDP 管脚定义表

Pin	Name	Pin	Name
1	GND	**16**	GND
2	EDP_TX0N	**17**	LCD_BL_PWM
3	EDP_TX0P	**18**	GND
4	GND	**19**	VCC3V3_S0
5	EDP_TX1N	**20**	LCD_RST_H
6	EDP_TX1P	**21**	NC
7	GND	**22**	LCD_EN_H
8	EDP_AUXN	**23**	I2C_SCL_TP
9	EDP_AUXP	**24**	I2C_SDA_TP
10	GND	**25**	TOUCH_INT_L
11	EDP_TX2N	**26**	TOUCH_RST_L
12	EDP_TX2P	**27**	GND
13	GND	**28**	VCC5V0_SYS
14	EDP_TX3N	**29**	VCC5V0_SYS
15	EDP_TX3P	**30**	VCC5V0_SYS

2.5.3 HDMI 显示

EAIDK-610 开发板支持 HDMI 显示，采用 A 型接口，可以同其他显示接口组成双屏 显示：双屏同显和双屏异显。

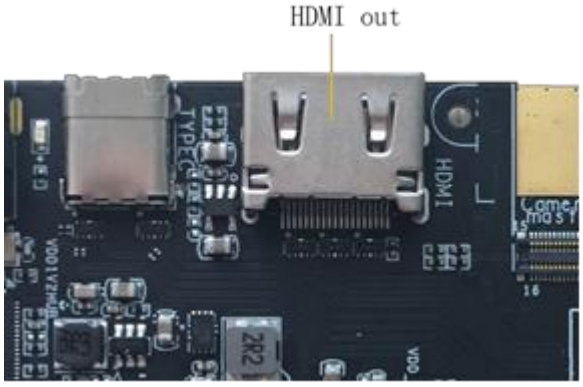

图 2-10 HDMI 位置示意图

2.6 MIPI 相机接口

EAIDK-610 开发板拥有 2 路 MIPI Camera 接口，可外接 2 个 OV9750 摄像头组成双 MIPI Camera 同步显示和前后摄像模式；也可外接 1 路 IMX258 实现 4K 高清摄像。

开发板上 2 路 MIPI 接口采用兼容设计。用户只需要设计简单的电源转换电路即可匹配其他 Camera 模组。

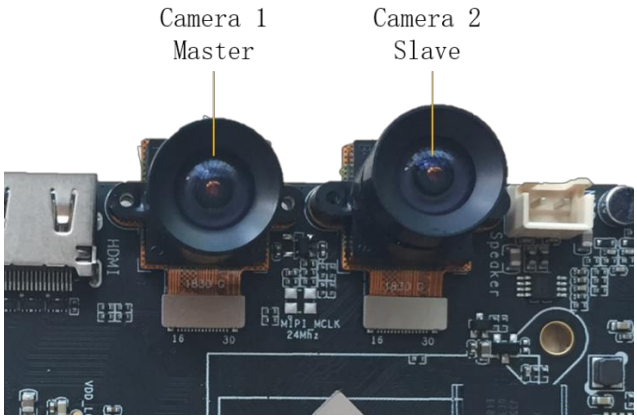

图 2-11 Camera 位置示意图

表 2-3 MIPI Rx0 管脚定义表

Pin	Name	Pin	Name
1	GND	**16**	GND
2	MIPI_RX0_D0P	**17**	VCC_AVDD
3	MIPI_RX0_D0N	**18**	VCC_AVDD
4	GND	**19**	GND
5	MIPI_RX0_D2P	**20**	I2C_SCL_1V8_CAM1
6	MIPI_RX0_D2N	**21**	I2C_SDA_1V8_CAM1
7	GND	**22**	VCC_DVDD_CAM11
8	MIPI_RX0_D3P	**23**	GND
9	MIPI_RX0_D3N	**24**	VCC1V8_DVP
10	GND	**25**	GND
11	MIPI_MCLK_CAM1	**26**	MIPI_RX0_D1N
12	MIPI_RST_CAM1J	**27**	MIPI_RX0_D1P
13	GND	**28**	GND
14	MIPI_PDN_CAM1J	**29**	MIPI_RX0_CLKP
15	FSIN/VSYNC1	**30**	MIPI_RX0_CLKN

2.7 音频模块

EAIDK-610 开发板集成 Realtek ALC5651 Codec 芯片，内置 Charge Pump，板载 MIC。支持立体声耳机无电容耦合输出和耳麦输入；支持麦克风差分输入，Speaker 输出。

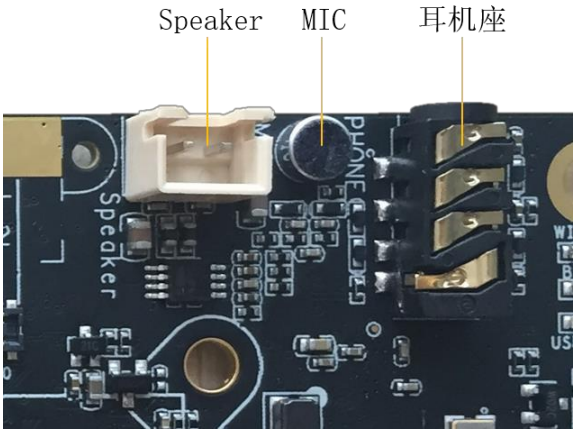

图 2-12 音频接口位置示意图

2.8 USB 模块

2.8.1 USB Host

EAIDK-610 开发板集成 2 路 USB2.0 Host 和 2 路 USB3.0 Host。外接 USB 鼠标、键盘 和 U 盘等多种的人机交互方式。

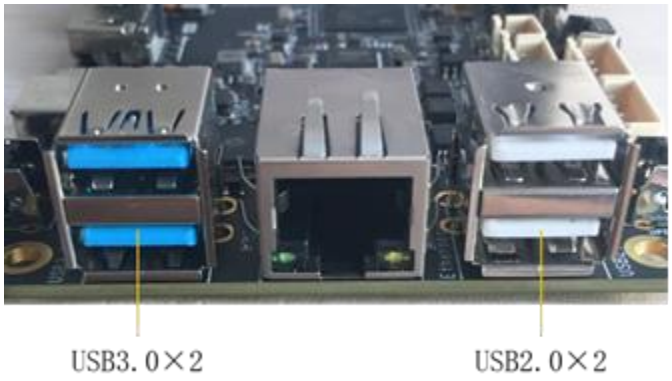

图 2-13 USB 位置示意图

2.8.2 Type-C

EAIDK-610 集成 Type-C 接口，支持 USB OTG 功能。可作为 android 的 adb device；当外接 USB 鼠标、键盘和 U 盘等多种的人机交互方式时，自动切换到 Host 模式。

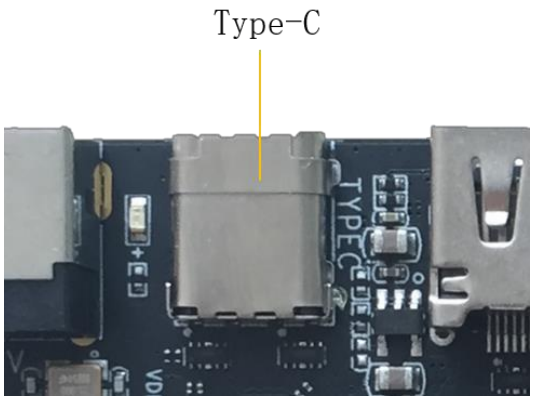

图 2-14 Type-C 位置示意图

2.9 网络通讯

2.9.1 以太网

EAIDK-610 开发板支持 RJ45 接口，可提供千兆以太网连接功能，选用 PHY 为 RTL8211E-VB-CG，其特性如下：

- 兼容 IEEE802.3 标准，支持全双工和半双工操作，支持交叉检测和自适应
- 支持 10/100/1000M 数据速率。
- 接口采用具有指示灯和隔离变压器的 RJ45 接口。

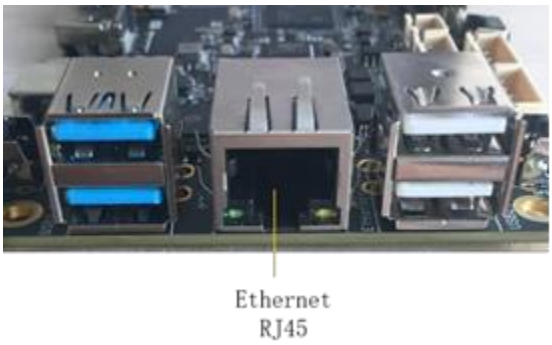

图 2-15 RJ45 位置示意图

2.9.2 WIFI/BT

开发板上 WIFI+BT 模组采用台湾正基的 AP6255，其特性如下：

- 支持 WIFI 2.4G 和 5G，802.11 ac，采用 4bits SDIO 通讯。
- 支持 BT4.1 功能，采用 UART 通讯。

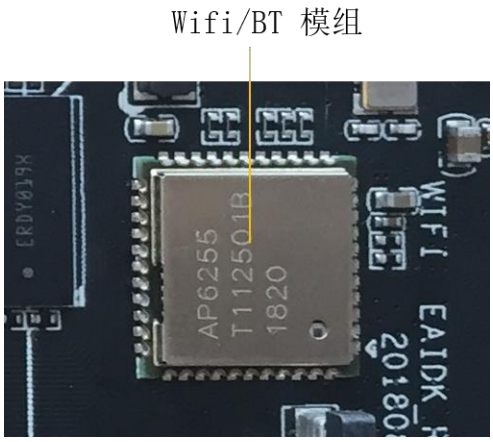

图 2-16 WIFI/BT 模组示意图

2.10 低速 IO 接口

EAIDK-610 集成 40 Pins IO 扩展接口和 1 路 8 通道 I2S 接口。其配套低速 IO 配件，支 持 I2C/SPI/ADC/GPIO 教学实验和 6 路麦克风阵列+ADC 回采。

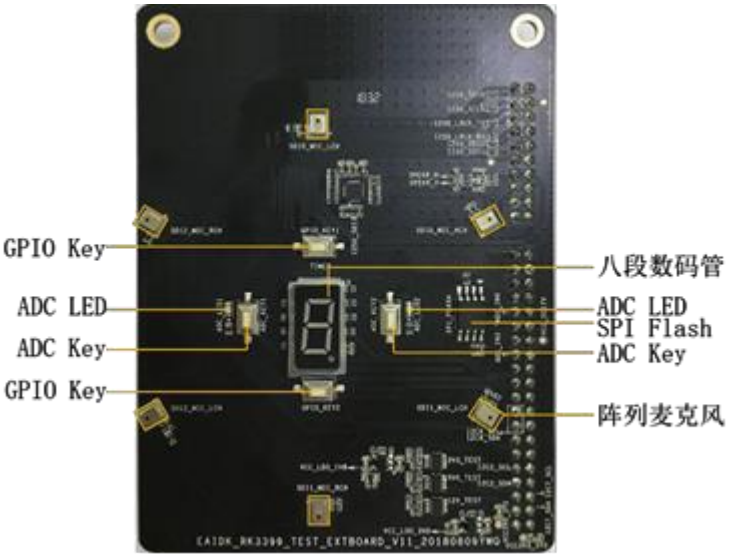

图 2-17 低速 IO 板示意图

低速接口 GPIO_EXT 的管脚定义如表 3-4，I2S 接口 I2S_EXT 的管脚定义如表 3-5

表 2-4 GPIO_EXT 管脚定义表

Pin	Name	Pin	Name
1	VCC3V3_SYS	2	VCC5V0_SYS
3	GPIO2_A7/I2C7_SDA	4	VCC5V0_SYS
5	GPIO2_B0/I2C7_SCL	6	GND
7	GPIO4_D0	8	GPIO2_A0/I2C2_SDA
9	GND	10	GPIO2_A1/I2C2_SCL
11	GPIO1_C6	12	
13	GPIO1_D0	14	GND
15	GPIO4_D2	16	GPIO2_B1/I2C6_SDA
17	VCC3V3_SYS	18	GPIO2_B2/I2C6_SCL
19	SPI1_TXD	20	GND
21	SPI1_RXD	22	
23	SPI1_CLK	24	SPI1_CSn0
25	GND	26	ADC_IN3
27	VCC_DC12V	28	VCC_DC12V
29	GPIO2_A2	30	GND
31	GPIO2_A4	32	ADC_IN0
33	GPIO2_A3	34	GND
35	GPIO2_B4	36	GPIO2_A6
37	GPIO4_D5	38	GPIO2_A5
39	GND	40	GPIO1_C7

表 2-5 I2S_EXT 管脚定义表

Pin	Name	Pin	Name
1	VCC5V0_SYS	**11**	I2S0_SDI1
2	VCC5V0_SYS	**12**	GND
3	GND	**13**	I2S0_SDI2
4	GND	**14**	I2S0_LRCK_RX
5	I2C_SCL_AUDIO	**15**	I2S0_SDI3
6	I2C_SDA_AUDIO	**16**	I2S0_LRCK_TX
7	SPKER_P	**17**	I2S0_SDO0
8	SPKER_N	**18**	I2S0_SCLK

9	GND	**19**	I2S0_SDI0
10	I2S_MUTE	**20**	I2S_CLK

2.11 UART 接口

2.11.1 RS232

EAIDK-610 集成 2 路 RS232 接口，支持双工通讯，支持软件标准 UART 编程。

2.11.2 RS485

EAIDK-610 集成 2 路 RS485 接口，支持半双工通讯，支持软件标准 UART 编程。

3 连接外部设备

3.1 登录

1. 连接电源，启动 EAIDK
2. 连接鼠标键盘，输入用户名密码 openailab/oal20200230（如果需要输入用户名密码）,登录 EAIDK

   连接线和输入界面如下图所示：

图 3-1 连接线正面照片

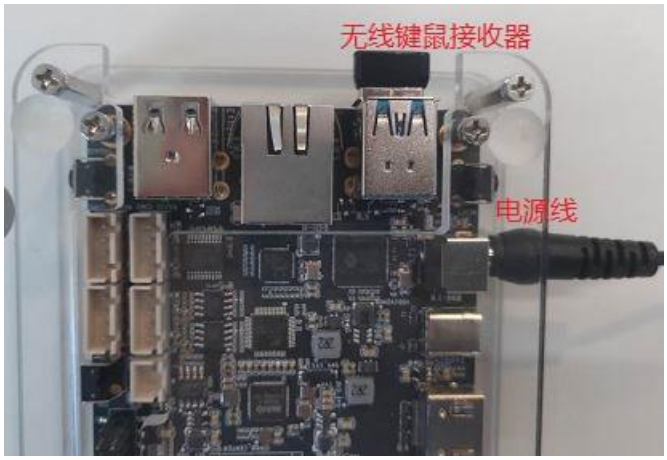

图 3-2 连接线背面照片

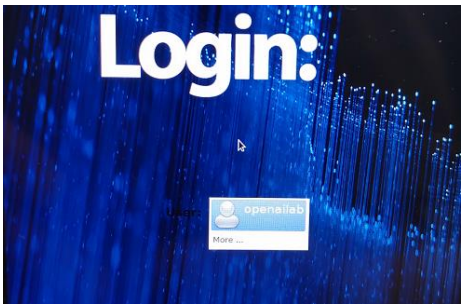

图 3-3 登录界面

3.2 网络配置

3.2.1 连接有线网络(以 IPv4 为例)

1. 使用网线连接 EAIDK-610 与交换机。
2. 右键点击屏幕右下角，网络连接图标，选择 Edit Connections。

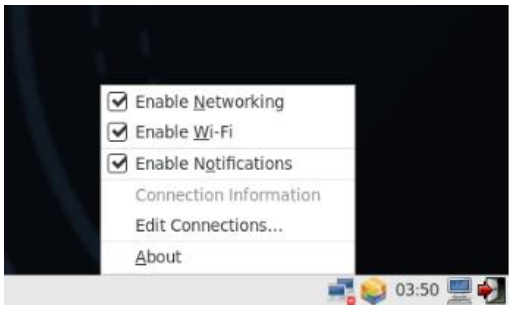

图 3-4 编辑网络设置

3. 双击 Wired connection 1,选择 IPv4 Settings。

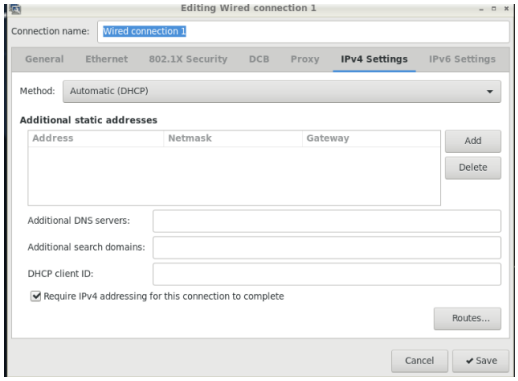

图 3-5 有线设置

4. 如果使用 DHCP，删除已经配置好的静态 ip，method 选择 Automatic(DHCP),如果需要手动设置 IP，则 Method 选择 Manual,并点击 Add 按钮，输入要设置的 IP，掩码和网关，并点击 Save 按钮。

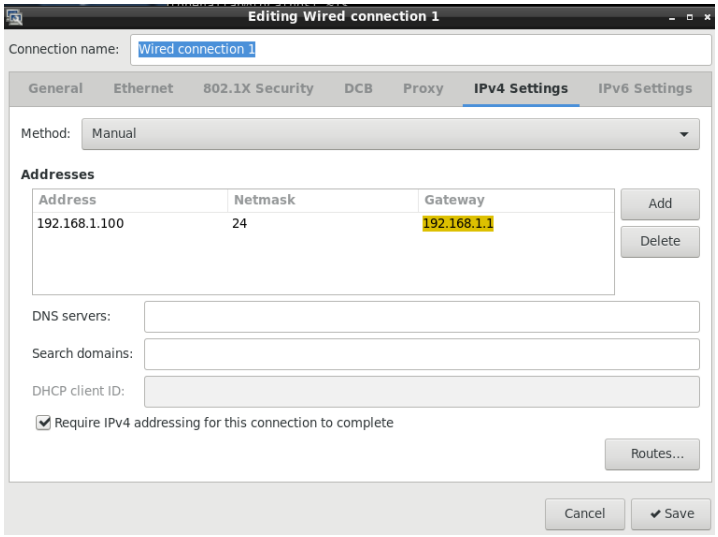

图 3-6 有线网络设置

3.2.2 连接 WIFI

1. 左键点击右下角网络连接图标

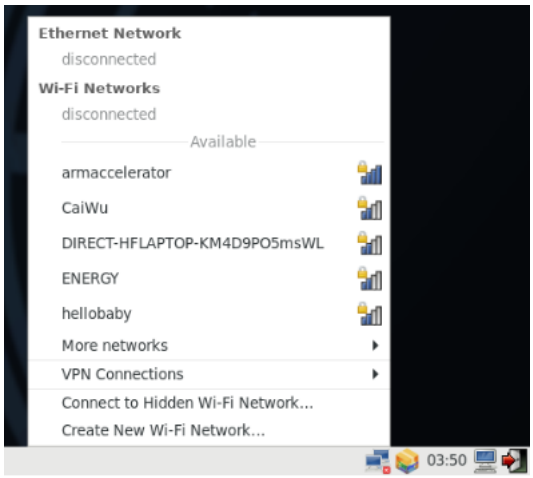

图 3-7 无线网络

2. 点击需要连接的 wifi，输入密码，点击 Connect 按钮

图 3-8 输入无线密码

4 软件及开发

4.1 系统登陆

EAIDK-610 预装 Fedora 28 及轻量级桌面系统 LXDE。缺省登录账号为 openailab，密码为 oal20200230。

EAIDK-610 的固件和源码都可以从 EAIDK 的官方 FTP 获取，其地址为 ftp://ftp.eaidk.net

4.2 环境设置

4.2.1 添加源（默认已经配置）

1. 安装 RK3399 硬件相关的系统库 RPM 包的源：

```
sudo yum -y localinstall --nogpgcheck  http://www.eaidk.net/rockchip/rockchip-repo-1.0-1.fc28.aarch64.rpm
```

2. 安装 Openailab 的系统和应用 RPM 包的源：

```
sudo yum -y localinstall --nogpgcheck http://www.eaidk.net/openailab/openailab-repo-1.0-1.fc28.aarch64.rpm
```

3. 安装第三方非开源的 RPM 包的源（可选）：

```
sudo yum localinstall --nogpgcheck http://download1.rpmfusion.org/free/fedora/rpmfusion-free-release-28.noarch.rpm

sudo yum localinstall --nogpgcheck http://download1.rpmfusion.org/nonfree/fedora/rpmfusion-nonfree-release-28.noarch.rpm
```

4.2.2 安装 RPM 包（默认已经安装）

1. 安装编译工具：
   sudo dnf -y install gcc gcc-c++ cmake automake
2. 安装 openssl(开发板上编译内核需要)：
   sudo dnf -y install openssl-devel

4.3 固件烧写

4.3.1 Windows 主机烧写

首次烧写前需要安装 Windows PC 端 USB 驱动：

双击 Tools\Windows\DriverAssitant_v4.5\ DriverInstall.exe 打开安装程序，点击“驱动安装”按提示安装驱动即可，安装界面如下所示：

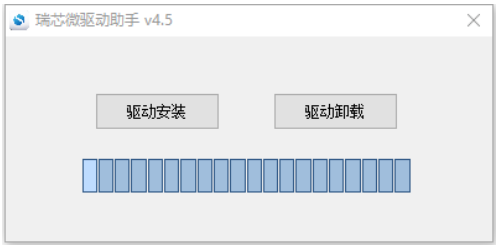

图 4-1 安装界面图

驱动安装完成后，固件烧写步骤如下：

1. Type-C 线连接主机端的 USB 接口和 EAIDK-610 开发板的 Type-C 接口。
2. 双击 Tools\Windows\EAIDK_FlashTool\ EAIDK_FlashTool.exe 打开程序。
3. 长按 EAIDK-610 开发板上 update 按键后重启机器，直到系统进入 Loader 模式，FlashTool 显示如下所示：

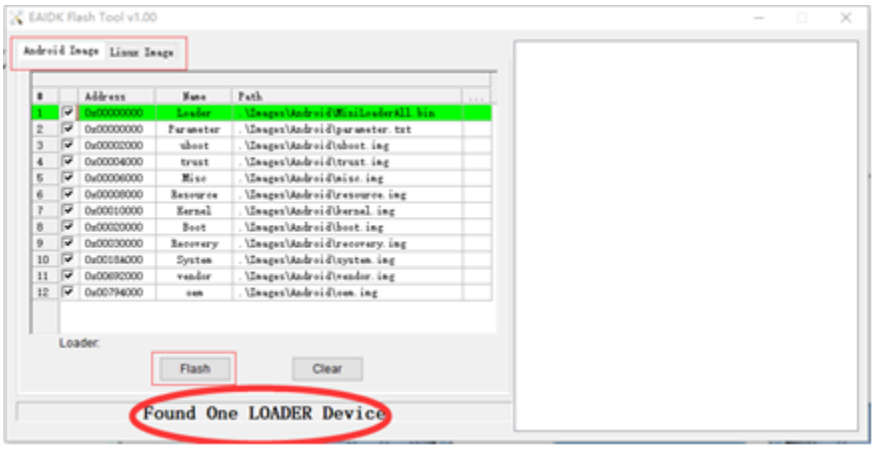

图 4-2 FlashTool 显示界面图

注意：如果 uboot 启动异常而无法进入 Loader 模式，需要长按 Maskrom 按键让系统进入 Maskrom 模式;FlashTool 提示：Found one Maskrom Device!

4. 将 Android 固件拷贝到 Tools\Windows\EAIDK_FlashTool\Images\Android 目录，或将 Linux 固件拷贝到 Tools\Windows\EAIDK_FlashTool\Images\Linux 目录。
5. 点击 TAB “Android Image” 或 “Linux Image” ，选择要烧写的系统固件。
6. 在选择好的系统固件页面，点击按钮 “Flash” ，开始烧写固件。

4.3.2 Linux 主机烧写

1. Type-C 线连接主机端的 USB 接口和 EAIDK-610 开发板的 Type-C 接口。

说明：EAIDK-610 的 fedora28 系统也可以作为开发主机给其他开发板烧写固件。

2. 长按 EAIDK-610 开发板上 update 按键后重启机器，进入 Loader 模式。
3. 进入 Tools/Linux 目录：

```
cd Tools/Linux
```

4. 将 Android 固件拷贝到 Tools/Linux/Images/Android 目录。
5. 将 Linux 固件拷贝到 Tools/Linux/Images/Linux 目录。
6. 执行命令烧写固件：

```
./flash.sh
```

1) 烧写 Android 固件：

./flash.sh -a: 烧写所有 Android 固件。

./flash.sh -a boot: 烧写 Android 的 Miniloader.img、trust.img 和 uboot.img。

./flash.sh -a kernel: 烧写 Android 的 kernel.img 和 resource.img。

./flash.sh -a system: 烧写 Android 的 system.img。

./flash.sh -a android: 烧写 Android 的 boot.img、system.img、misc.img、recovery.img、vendor.img 和 oem.img。

2) 烧写 Linux 固件：

./flash.sh -l：烧写所有 Linux 固件

./flash.sh -l boot: 烧写 Linux 的 Miniloader.img、trust.img 和 uboot.img。

./flash.sh -l kernel: 烧写 Linux 的 kernel.img 和 resource.img。

./flash.sh -l rootfs：烧写 fedora 根文件系统。

5 GPIO 编程

5.1 什么是 GPIO

GPIO，英文全称为 General-Purpose IO ports，也就是通用 IO 口。嵌入式系统中常常有数量众多，但是结构却比较简单的外部设备/电路，对这些设备/电路有的需要 CPU 为之提供控制手段，有的则需要被 CPU 用作输入信号。而且，许多这样的设备/电路只要求一位，即只要有开/关两种状态就够了，比如灯亮与灭。对这些设备/电路的控制，使用传统的串行口或并行口都不合适。所以在微控制器芯片上一般都会提供一个通用可编程 IO 接口，即 GPIO。

接口至少有两个寄存器，即“通用 IO 控制寄存器”与“通用 IO 数据寄存器”。数据寄存器的各位都直接引到芯片外部，而对这种寄存器中每一位的作用，即每一位的信号流通方向，则可以通过控制寄存器中对应位独立的加以设置。这样，有无 GPIO 接口也就成为微控制器区别于微处理器的一个特征。

GPIO 通常的用法有：

输出功能(GPO) 。通过编程实现对 I/O 输出电平的控制。

输入功能(GPI) 。可以对输入的信号进行检测。根据输入信号的电压大小，可以将相应的电压信号转化为逻辑信号。

信号复用功能。不同模块使用同一个 I/O 口作为输入输出通道，需要决定此时 I/O 口的功能。

上拉和下拉配置 。上拉就是将 I/O 口通过电阻与单片机的输入电源相接，使其在释放状态下保持在高电平状态。下拉就是将 I/O 口通过电阻与地相接，使在释放状态下保持在低电平状态。

I/O 的触发事件 。I/O 在配置成输入模式时可以用于检测外部事件的触发，这里的外部事件包括高/低电平、脉冲波的上升/下降沿。

5.2 GPIO 分布图

EAIDK-610 管脚分布实物图如下图,Pin1 和 Pin2 已经用红圈标出，其他管脚以此类推。

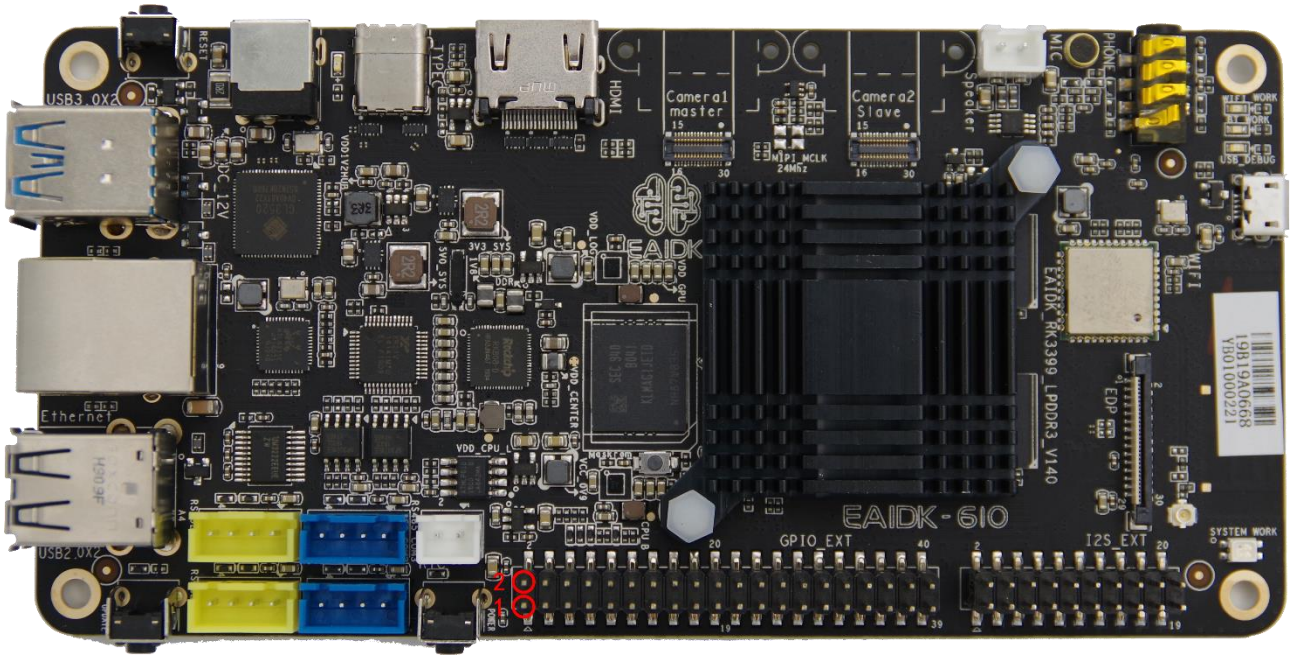

图 5-1 管脚分布图

各个管脚定义如下表

表 5-1 管脚定义

Pin	Name	Pin	Name
1	VCC3V3_SYS	2	VCC5V0_SYS
3	GPIO2_A7/I2C7_SDA	4	VCC5V0_SYS
5	GPIO2_B0/I2C7_SCL	6	GND
7	GPIO4_D0	8	GPIO2_A0/I2C2_SDA
9	GND	10	GPIO2_A1/I2C2_SCL
11	GPIO1_C6	12	
13	GPIO1_D0	14	GND
15	GPIO4_D2	16	GPIO2_B1/I2C6_SDA
17	VCC3V3_SYS	18	GPIO2_B2/I2C6_SCL
19	SPI1_TXD	20	GND
21	SPI1_RXD	22	
23	SPI1_CLK	24	SPI1_CSn0
25	GND	26	ADC_IN3
27	VCC_DC12V	28	VCC_DC12V
29	GPIO2_A2	30	GND
31	GPIO2_A4	32	ADC_IN0
33	GPIO2_A3	34	GND

35	GPIO2_B4	36	GPIO2_A6
37	GPIO4_D5	38	GPIO2_A5
39	GND	40	GPIO1_C7

从上表可以得出。PIN3, PIN5, PIN7, PIN8, PIN10, PIN11, PIN13, PIN15, PIN16, PIN18, PIN29, PIN31, PIN33, PIN35, PIN36, PIN37, PIN38, PIN40 为 GPIO。

GPIO 设置成输出模式时，高电平为 3.3V，低电平为 0V。

GPIO 设置成输入模式时，高于 1.8V 为高电平，低于 1.8V 为低电平。

5.3 控制 GPIO

从表 7-1 可以看出，GPIO 的 Name 格式都是 GPIOX_YZ 形式，其中 X，Z 是一个数字，Y 是 A-D 的大写字母，引脚号计算公式为

$$X * 32 + F(Y) * 8 + Z \qquad 式1$$

其中当 Y=A,B,C,D 时, $F(Y)$分别为 0,1,2,3。

以 GPIO2_A7 为例。通过式 1，计算出 GPIO2_A7 引脚号为$2 * 32 + 0 * 8 + 7 = 71$。

1. 切换 root 权限。

```
[openailab@localhost ~]$ sudo su
```

2. 进入 /sys/class/gpio/ 目录 。

```
[root@localhost openailab]# cd /sys/class/gpio/
```

3. 生成一个 gpio71 的目录。

```
[root@localhost gpio]# echo 71 > export
```

4. 进入 gpio71 目录。

```
[root@localhost gpio]# cd gpio71/
```

gpio71 目录下主要有以下文件

1)direction： 设置输出还是输入模式

设置输入模式。

```
[root@localhost gpio71]# echo in > direction
```

设置输出模式。

```
[root@localhost gpio71]# echo out > direction
```

2)value： 输出时，控制高低电平；输入时，获取高低电平

设置高电平。

```
[root@localhost gpio71]# echo 1 > value
```

设置低电平。

```
[root@localhost gpio71]# echo 0 > value
```

3)edge： 控制中断触发模式，引脚被配置为中断后可以使用 poll() 函数监听引脚

非中断引脚：

```
[root@localhost gpio71]#  echo “none” > edge
```

上升沿触发：

```
[root@localhost gpio71]# echo “rising” > edge
```

下降沿触发：

```
[root@localhost gpio71]# echo “falling” > edge
```

边沿触发：

```
[root@localhost gpio71]# echo “both” > edge
```

5.如果想要卸载 gpio71，需要回到`/sys/class/gpio/` 目录并执行 `echo 71 > unexport`

5.4 点亮 LED 实例

LED 实物如图所示。

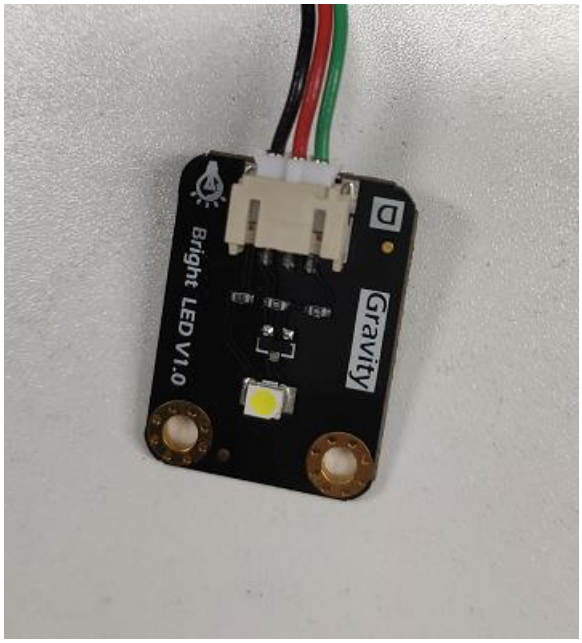

图 5-2 LED 实物

将 LED 的黑线接地，红线接 VCC，绿线接 GPIO1_C6。可以通过控制 GPIO2_A7 电位来观察 LED 是否发光。电路连线如下图。

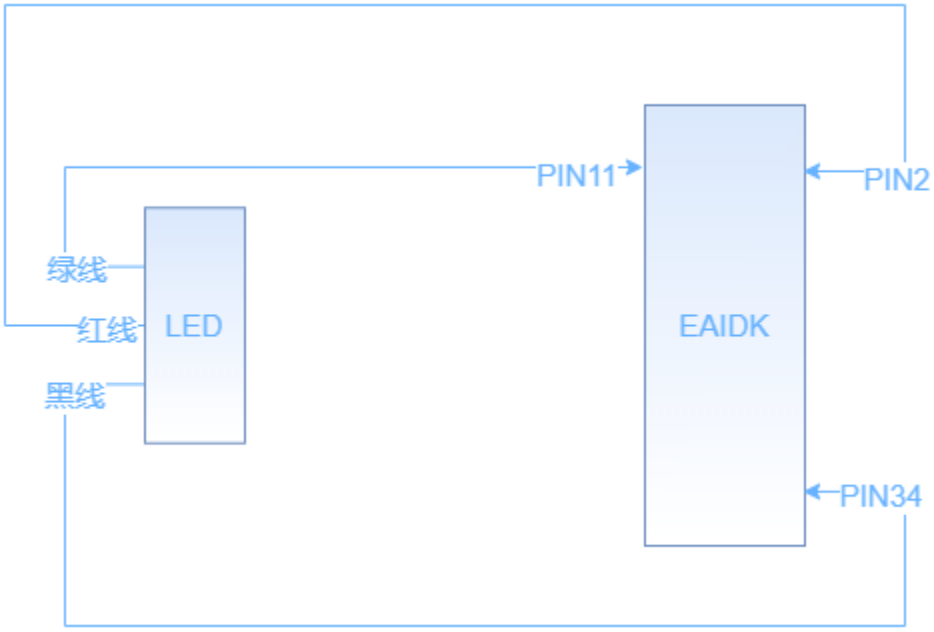

图 5-3 LED 实例电路图

根据 EAIDK 管脚分布图，PIN2 为 5V 直流电压，PIN11 为 GPIO1_C6,PIN34 为地。

把 PIN11 设置成输出模式。当 PIN11 为高电位时，LED 被点亮；当 PIN11 为低电位时，LED 被熄灭。

5.5 捕获按钮按下实例

按钮实物如图所示。

图 5-4 按钮实物

将按钮的黑线接地，红线接 VCC，绿线接 GPIO1_C6。可以通过观察 GPIO1_C6 电位来检测按钮是否被按下。电路连线如下图。

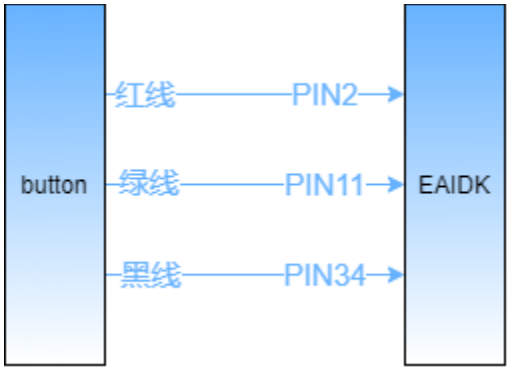

图 5-5 按钮实例电路图

根据 EAIDK 管脚分布图，PIN2 为 5V 直流电压，PIN11 为 GPIO1_C6,PIN34 为地。

把 PIN11 设置成输入模式。当按钮没有被按下，PIN11 为低电位；当按钮被按下，PIN11 为高电位。

5.6 点亮数码管实例

四位串行数码管实物图如下。

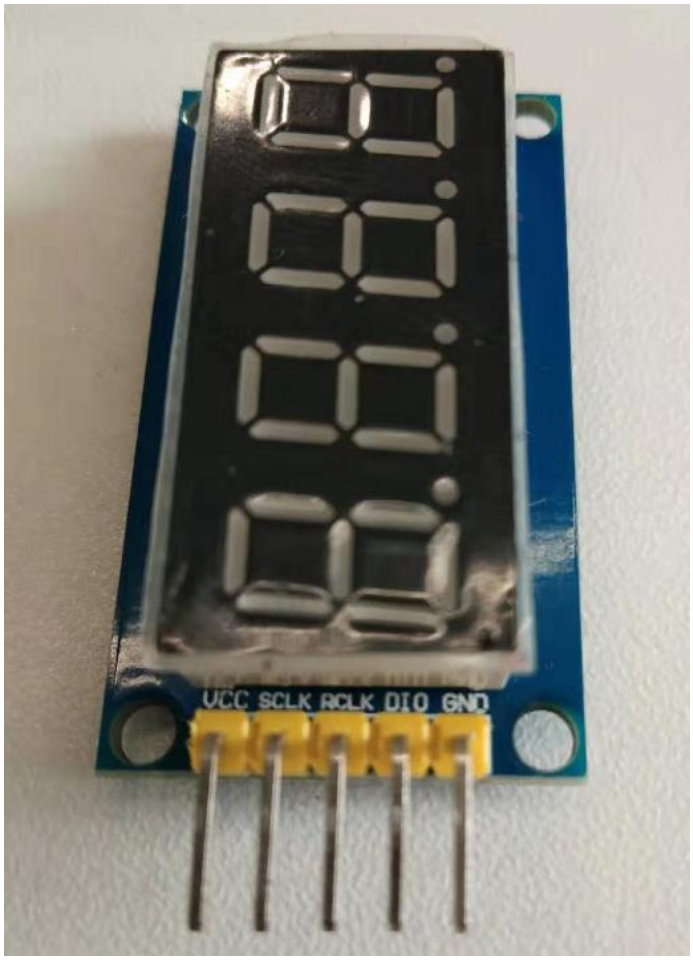

图 5-6 数码管实物

从上图可知，数码管有 5 个管脚，各管脚作用如下

表 5-6 四位串行数码管管脚定义

管脚	作用
VCC	接 5v 直流电，与 GND 一起为数码管供电
SCLK	位输入确认时钟，每次向 DIO 输入一个 bit 后要给 SCLK 一个低电平脉冲
RCLK	字节位置输入确认时钟，每次向 DIO 输入一个字节和位置后要给 RCLK 一个低电平脉冲
DIO	数据输入

GND	接地，与 VCC 一起为数码管供电

DIO 输入的数据有两种作用。一种是在数码管显示数字，一种是指定哪一个数码管显示。

1.显示数字

DIO 需要输入一个 8bit 的字节来表示需要显示的数字。一个 8 段数码管引脚定义如下。

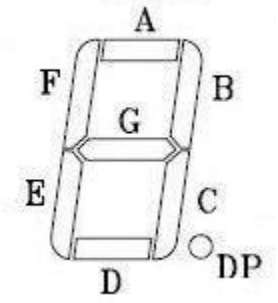

图 5-7 八段数码管引脚定义

输入的 8bit 字节第一位至第八位分别控制 DP,G,F,E,D,C,B,A。低电平点亮，高电平熄灭。比如，当 DIO 输入 0xF9 时，字节第 6 位和第 7 位是低电平，其余位为高电平。第 6 位和第 7 位分别对应数码管引脚 C 和 B，C 和 B 点亮，显示数字'1'。

2.指定数码管

DIO 需要输入一个 8bit 的字节来指定显示数字的数码管。四位数码管总共有四个数码管，输入 0x08 指定第一个数码管，输入 0x04 指定第二个数码管, 输入 0x02 指定第三个数码管，输入 0x01 指定第四个数码管。

电路连线如下图。

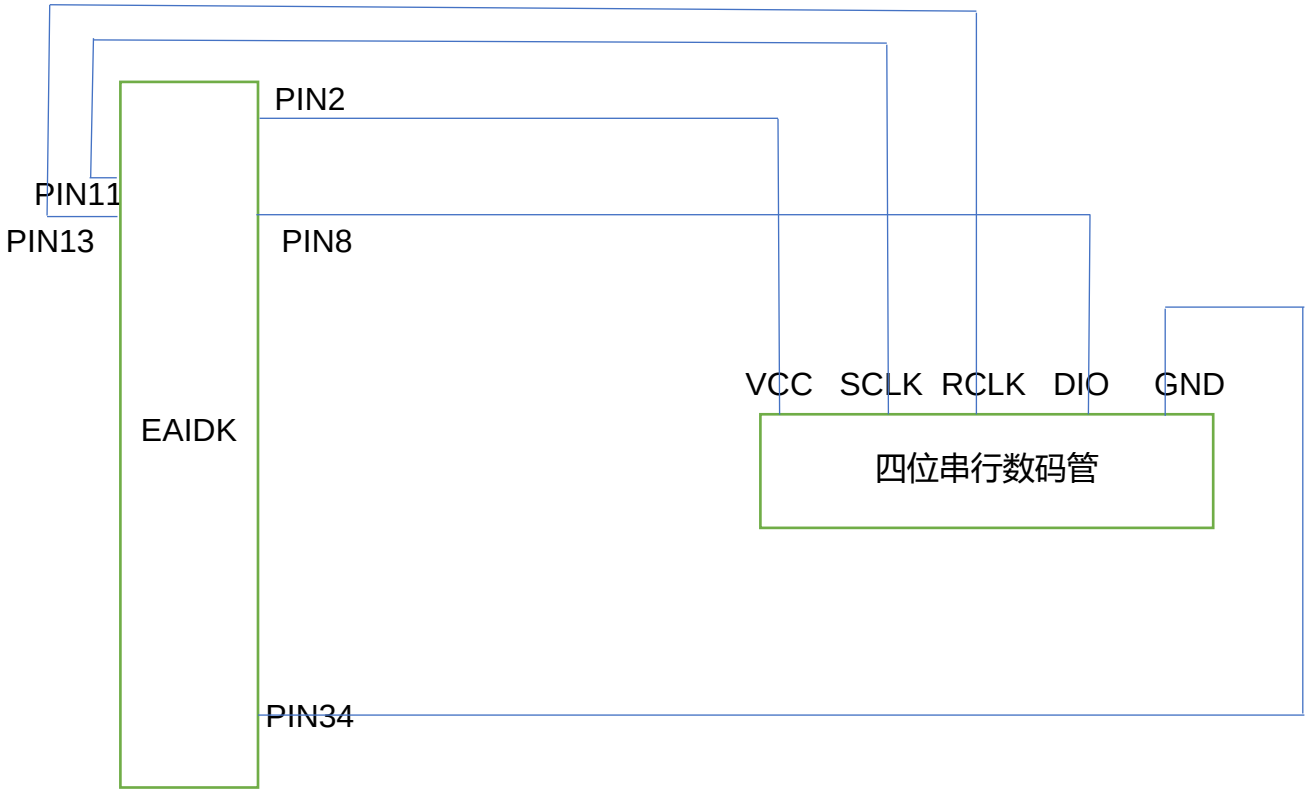

图 5-8 数码管实例电路图

根据 EAIDK 管脚分布图，PIN2 为 5V 高电平，PIN34 为地，PIN11,PIN13,PIN8 分别为 GPIO1_C6，GPIO1_D0, GPIO2_A0.

把 PIN11,PIN13,PIN8 都设置成输出模式。通过改变 PIN11,PIN13,PIN8 的电平来实现数码管的显示。

5.7 Python 语言程序说明

5.7.1 Python 文件说明

Python 将 gpio 案例的实现集成到了 gpio.py 文件中。Python 的版本为 3.6

5.7.2 枚举类型说明

与 C 语言版本一致

```
class DIRECTION(Enum):
        INVALI_DIRECTION = 0
        INPUT = 1
        OUTPUT =2

class EDGE(Enum):
        INVALI_EDGE =0
        NONE =1
        RISING =2
        FALLING =3
        BOTH =4
```

5.7.3 接口说明

接口名称	`gpio_init ( gpio, direction, edge, value)`	
接口说明	gpio管脚初始化	
参数	`gpio`	gpio管脚号
	`direction`	IO模式。INPUT表示输入，OUTPUT表示输出，INVALID_DIRECTION表示不设置，定义在枚举类型中
	`edge`	中断触发模式。NONE表示无中断模式，RISING表示上升沿触发，FALLING表示下降沿触发，BOTH表示边沿触发，INVALID_ EDGE表示不设置。定义在枚举类型中
	`value`	gpio管脚对应的控制电平的文件，即/sys/class/gpio/gpio%d/value。作输出用。**本案例中必须为True**

接口名称	`set_voltage(fd_path,value)`

接口说明	设置gpio管脚电平值	
参数	fd_path	gpio管脚对应的控制电平的文件描述符
	value	电平值。0代表低电平，1代表高电平

接口名称	get_voltage(fd)	
接口说明	获取gpio管脚电平值	
参数	fd	gpio管脚对应的控制电平的文件句柄
返回值	0：低电平；1：高电平；	-1：失败

接口名称	gpio_release(gpio)	
接口说明	gpio管脚资源释放	
参数	gpio	gpio管脚号
返回值	无	

接口名称	led_demo()	
接口说明	点亮LED案例演示	
参数	无	
返回值	0：成功	-1：失败

接口名称	button_demo()	
接口说明	捕获按钮按下案例演示	
参数	无	
返回值	0：成功	-1：失败

5.7.4 程序运行

进入/pyCases/case/embedded/gpio 目录，获取 gpio.py 脚本;

按照图 5-3 连接好电路。运行 sudo python3 gpio.py -l，可演示点亮 LED 实例。可以观察到发光二极管亮灭交替;

按照图 5-5 连接好电路。运行 sudo python3 gpio.py -b，可演示捕获按钮按下实例。当按下按钮是，可以屏幕输出信息；

按照图 5-8 连接好电路。运行 sudo python3 gpio.py -d，可演示点亮数码管实例。可以观察到数码管显示 1234。

> 注意，在选择 GPIO PIN 时，不要选择具有多重功能的 PIN，例如 PIN3 和 PIN5。如果此引脚复用容易导致 IIC 案例运行失败。如真出现此类情况，我们 unexport 该引脚后重启设备，即可恢复正常。同时，在运行 GPIO 案例时可能遇到 device or resource busy 的报错。出现这个的原因是因为我们多次运行了 GPIO 案例，导致同一引脚重复注册，该打印并不影响设备正常功能。若想解决此类打印，我们可以先进入到/sys/class/gpio 目录下 unexport 对应引脚，再运行程序，即可消除报错。

6 同步串行口编程

6.1 同步串行口分布图

EAIDK-610 管脚分布实物图如下图,Pin1 和 Pin2 已经用红圈标出，其他管脚以此类推。

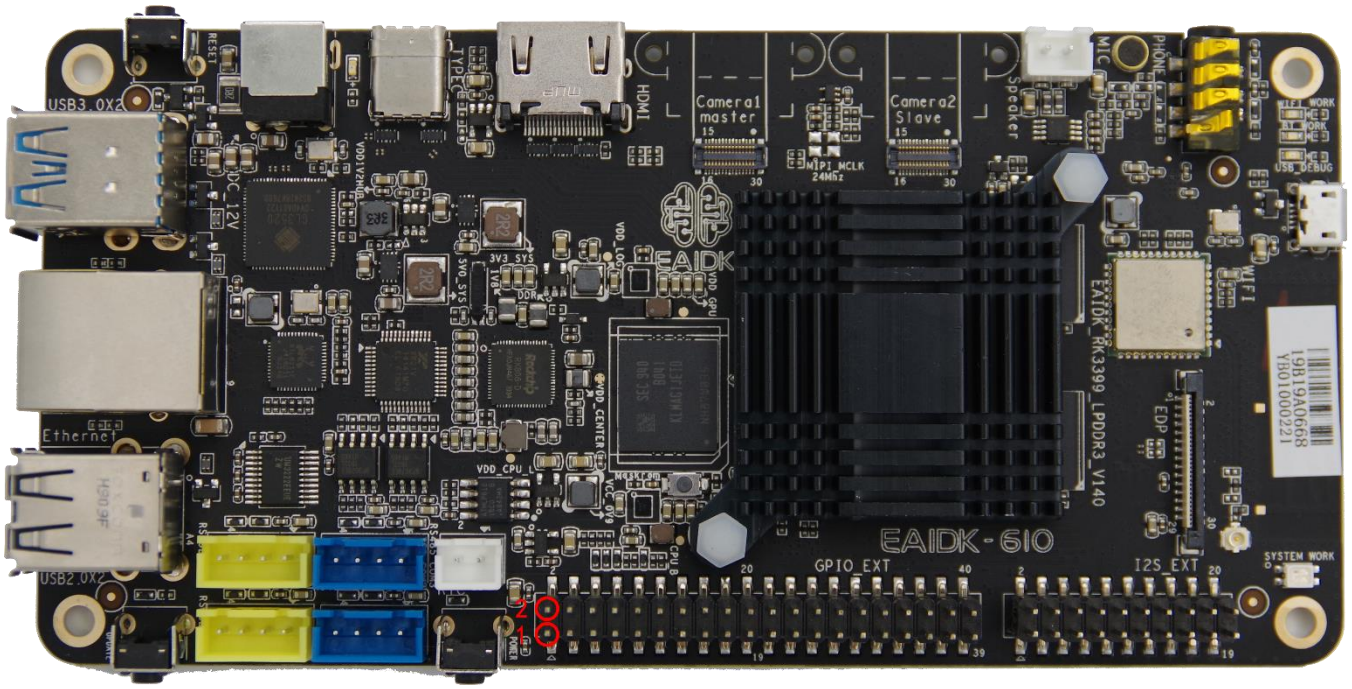

图 6-1 管脚分布图

各个管脚定义如下表

表 6-1 管脚定义

Pin	Name	Pin	Name
1	VCC3V3_SYS	2	VCC5V0_SYS
3	GPIO2_A7/I2C7_SDA	4	VCC5V0_SYS
5	GPIO2_B0/I2C7_SCL	6	GND
7	GPIO4_D0	8	GPIO2_A0/I2C2_SDA
9	GND	10	GPIO2_A1/I2C2_SCL
11	GPIO1_C6	12	
13	GPIO1_D0	14	GND
15	GPIO4_D2	16	GPIO2_B1/I2C6_SDA
17	VCC3V3_SYS	18	GPIO2_B2/I2C6_SCL
19	SPI1_TXD	20	GND

21	SPI1_RXD	22	
23	SPI1_CLK	24	SPI1_CSn0
25	GND	26	ADC_IN3
27	VCC_DC12V	28	VCC_DC12V
29	GPIO2_A2	30	GND
31	GPIO2_A4	32	ADC_IN0
33	GPIO2_A3	34	GND
35	GPIO2_B4	36	GPIO2_A6
37	GPIO4_D5	38	GPIO2_A5
39	GND	40	GPIO1_C7

从上表可以得出。EAIDK-610 支持的 IIC 有三组，分别是｛PIN3（I2C7_SDA），PIN5（I2C7_SCL）｝，｛PIN8（I2C2_SDA），PIN10（I2C2_SCL）｝，｛PIN16（I2C6_SDA），PIN18（I2C6_SCL）｝。支持的 SPI 管脚分别是 PIN19（SPI_TXD），PIN21（SPI_RXD），PIN23（SPI_CLK），PIN24（SPI_CSn0）。

6.2 通过 bme280 获取实时温度实例

bme280 是一种温度湿度气压三合一的传感器，可以提供 IIC 和 spi 两种接口模式。bme280 实物如图所示。

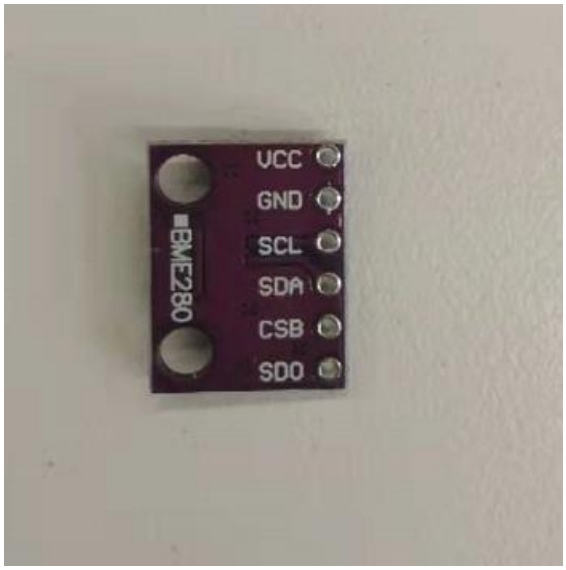

图 6-2 bme280 实物

6.2.1 通过 IIC 获取温度值

将 bme280 的 VCC 接高电平，GND 接低电平，SCL 接 EAIDK-610 的 PIN5（I2C7_SCL），SDA 接 EAIDK-610 的 PIN3（I2C7_SDA）。可以实时监测当前温度。电路连线如下图。

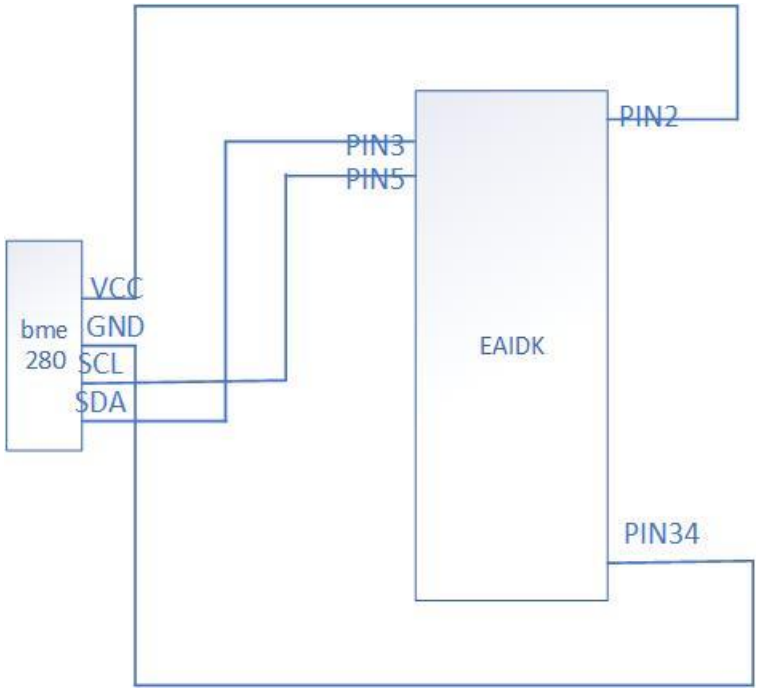

图 6-3 bme28 iic 实例电路图

根据 EAIDK-610 管脚分布图，PIN5 为 I2C7_SCL，PIN3 为 I2C7_SDA。,PIN2 为 5V 高电平，PIN34 为地。

通过 I2C 接口，可以实时读取 bme280 监测到的温度值。

6.2.2 通过 spi 获取温度值

bme280 的 SPI 接口复用了 IIC 接口，复用关系如下：

表 6-2 bme280 接口复用表

IIC 接口	SPI 接口
SCL	SCK
SDA	SDI

将 bme280 的 VCC 接高电平，GND 接低电平，SCK（SCL）接 EAIDK-610 的 PIN23（SPI1_CLK），SDI（SDA）接 EAIDK-610 的 PIN19（SPI1_TXD），CSB 接 EAIDK-610 的 PIN24（SPI1_CSn0），SDO 接 EAIDK-610 的 PIN21（SPI1_RXD）。可以实时监测当前温度。电路连线如下图。

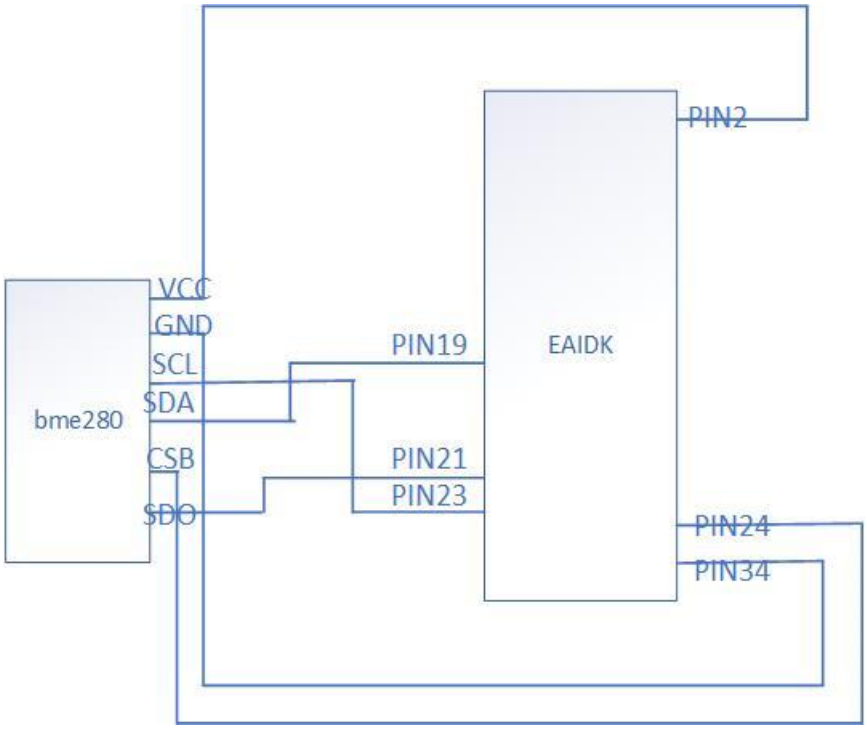

图 6-4 bme280 spi 实例电路图

6.3 Python IIC 程序说明

6.3.1 文件说明

Python 将 IIC 案例的实现集成到了 iic_bme280.py 文件中。Python 的版本为 3.6。使用的 python smbus2 库。使用之前需要使用命令 sudo pip3 install smbus2 安装。

6.3.2 接口说明

Smbus2 使用官方文档 **https://smbus2.readthedocs.io/en/latest/**

接口名称	`read_i2c_block_data(i2c_addr, register, length, force=None)`
接口说明	从给定的寄存器读取字节数据块。

参数	i2c_addr	i2c地址
	register	开始寄存器
	length	所需的块长度
返回值	字节列表	

接口名称	write_byte_data(i2c_addr, register, value, force=None)	
接口说明	将字节写入给定寄存器	
参数	i2c_addr	i2c地址
	register	要写入的寄存器
	value	要传输的字节值
返回值	无	

6.3.3 运行程序

1、 进入案例目录/pyCases/case/embedded/iic
2、 按照图 6-3 连接好电路。
3、 运行 sudo python3 iic_bme280.py -t，可以实时观察到温度值。
4、 运行 sudo python3 iic_bme280.py -p，可以实时观察到气压值。
5、 运行 sudo python3 iic_bme280.py -h，可以实时观察到湿度值。
6、 运行 sudo python3 iic_bme280.py -a，可以实时观察到全部值。

6.4 Python 语言 spi 程序说明

6.4.1 文件说明

Python 将 SPI 案例的实现集成到了 spi_bme280.py 文件中。Python 的版本为 3.6。使用编译的 spidev 模块。安装 spidev 模块，执行 sudo pip3 install spidev 完成安装。

6.4.2 接口说明

接口名称	write_addr(fd, addr,ch)	
接口说明	写数据	
参数	fd	数据写入的文件描述符
	addr	写入的寄存器地址
	ch	写入的数据内容
返回值	无	

接口名称	read_addr(fd, addr, size)	
接口说明	读数据	
参数	fd	数据读取的文件描述符
	addr	读取寄存器的地址
	size	最大读取大小
返回值	成功：0	失败：-1

6.4.3 程序运行

进入 pyCases/case/embedded/spi 目录

按照图 6-4 连接好电路。运行一下程序

sudo python3 spi_bme280.py

7 异步串行口编程

7.1 异步串行口分布图

EAIDK-610 管脚分布实物图如下图,Pin1 和 Pin2 已经用红圈标出，其他管脚以此类推。

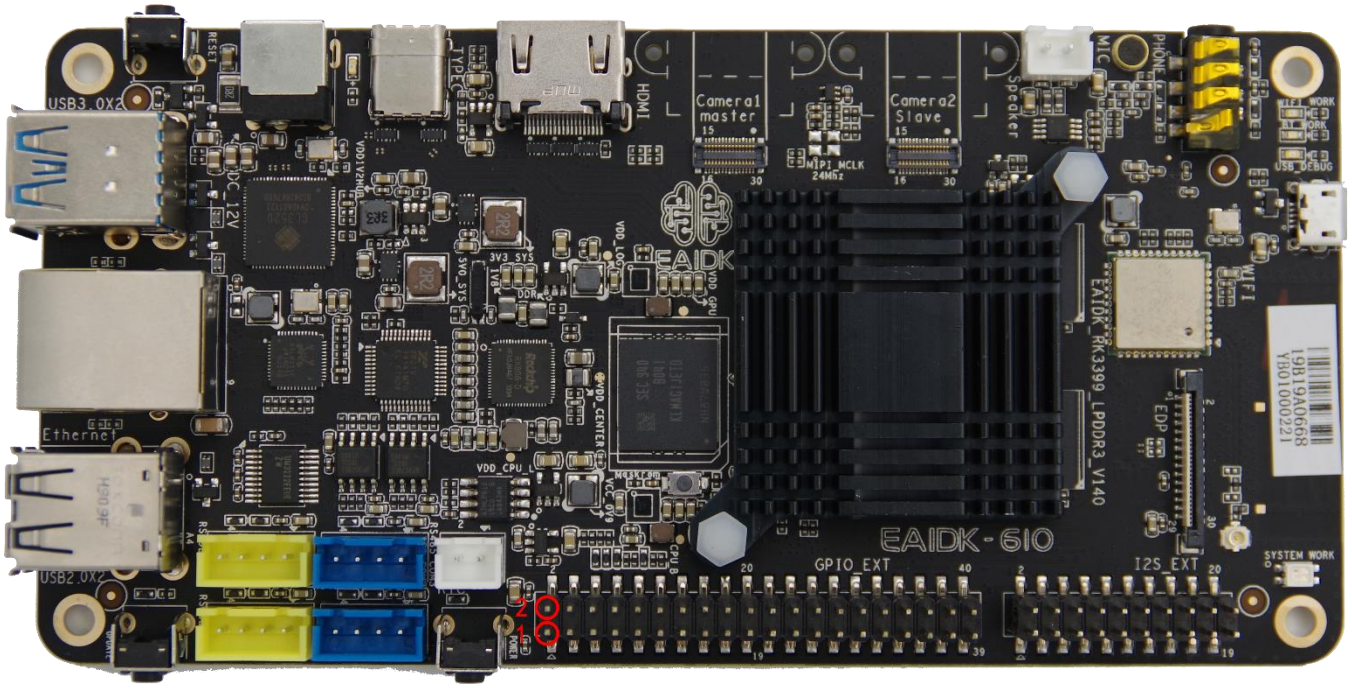

图 7-1 管脚分布图

各个管脚定义如下表

表 7-1 管脚定义

Pin	Name	Pin	Name
1	VCC3V3_SYS	2	VCC5V0_SYS
3	GPIO2_A7/I2C7_SDA	4	VCC5V0_SYS
5	GPIO2_B0/I2C7_SCL	6	GND
7	GPIO4_D0	8	GPIO2_A0/I2C2_SDA
9	GND	10	GPIO2_A1/I2C2_SCL
11	GPIO1_C6	12	
13	GPIO1_D0	14	GND
15	GPIO4_D2	16	GPIO2_B1/I2C6_SDA
17	VCC3V3_SYS	18	GPIO2_B2/I2C6_SCL
19	SPI1_TXD	20	GND

21	SPI1_RXD	22	
23	SPI1_CLK	24	SPI1_CSn0
25	GND	26	ADC_IN3
27	VCC_DC12V	28	VCC_DC12V
29	GPIO2_A2	30	GND
31	GPIO2_A4	32	ADC_IN0
33	GPIO2_A3	34	GND
35	GPIO2_B4	36	GPIO2_A6
37	GPIO4_D5	38	GPIO2_A5
39	GND	40	GPIO1_C7

从上表可以得出。EAIDK-610 支持的 TTL 管脚分别是 PIN8（UART1_TX）,PIN10（UART1_RX）。另外，由于板卡没有 RS485 的接口，故需要用 usb 转 RS485 接口来实现 RS485 通信功能。

7.2 3 通过 TTL 获取 TGS2600 实时烟雾浓度实例

TGS2600 是一种烟雾浓度传感器，可以提供 TTL 接口。由于 EAIDK-610 板卡没有 TTL 接口，故用 USB 转 TTL 设备来实现获取 TGS2600 的烟雾浓度。如下图所示。

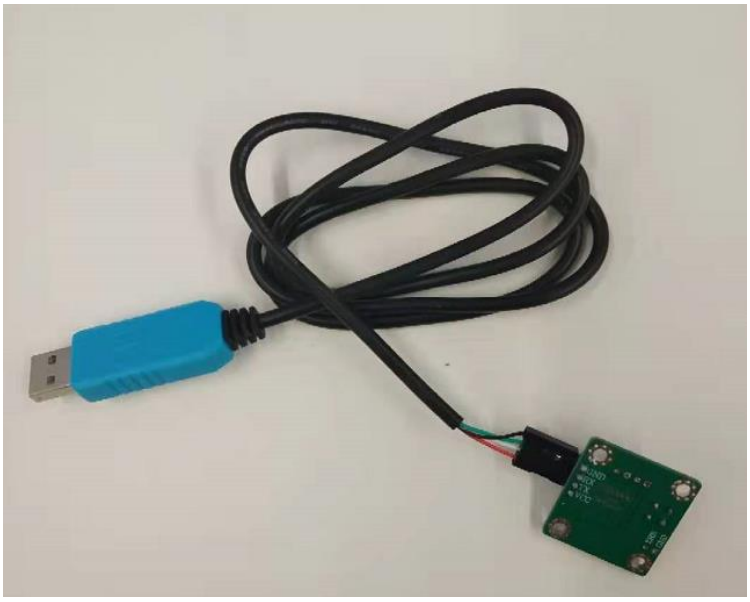

图 7-2 USB 转 TTL TGS2600 实物图

注意接线顺序，黑线接 GND，绿线接 RX，白线接 TX，红线接 VCC。

图 7-3 接线图

将 USB 口接入 EAIDK-610，可以在 EAIDK-610 上看到生成了/dev/ttyUSB0 这个设备（如果/dev/ttyUSB0 被其他 USB 设备占用了，就生成/dev/ttyUSB1，以此类推。具体情况以实际为准）。接下来通过运行程序，可以实现实时检测 TGS2600 的烟雾浓度。

7.3 通过 RS485 获取 SHT20 温度实例

SHT20 是一种温度传感器，可以提供 RS485 接口。我们用 USB 转 RS485 设备来实现获取 SHT20 的温度值。如下图所示。

图 7-4 USB 转 RS485 SHT20 实物图

注意接线顺序，红线接+5v，黑线接 GND，白线接 B，黄线接 A。

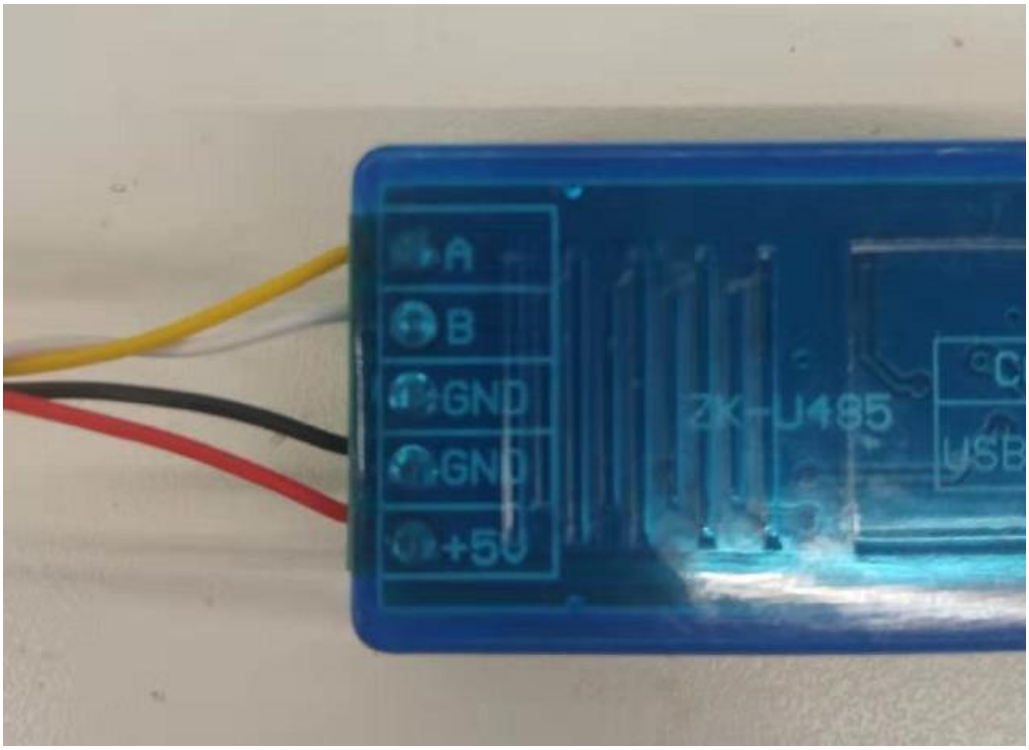

图 7-5 接线图

将 USB 口接入 EAIDK-610，可以在 EAIDK-610 上看到生成了/dev/ttyUSB0 这个设备（如果/dev/ttyUSB0 被其他 USB 设备占用了，就生成/dev/ttyUSB1，以此类推。具体情况以实际为准）。接下来通过运行程序，可以实现实时检测 SHT20 的温度值。

7.4 Python 语言 TTL 程序

7.4.1 文件说明

Python 将 tgs2600 案例的实现集成到了 uart_tgs2600.py 文件中。Python 的版本为 3.6。使用的 python pyserial 库。使用之前需要使用命令 `sudo pip3 install pyserial` 安装。

7.4.2 接口说明

Pyserial 官方使用简介 **https://pythonhosted.org/pyserial/shortintro.html**

Pyserial 模块封装了对串行端口的访问。它提供了在 Windows，OSX，Linux，BSD（可能是任何 POSIX 兼容系统）和 IronPython 上运行的 Python 的后端。名为“串行”的模块会自动选择适当的后端。

- 在所有支持的平台上基于相同类的接口。
- 通过 Python 属性访问端口设置。
- 通过 RTS / CTS 和/或 Xon / Xoff 支持不同的字节大小，停止位，奇偶校验和流控制。
- 有或没有接收超时的工作。
- 像 API 这样的文件，带有“读”和“写”（也支持“ readline”等）。
- 该软件包中的文件是 100％纯 Python。
- 该端口已设置为二进制传输。没有 NULL 字节剥离，CR-LF 转换等（对于 POSIX 启用了很多次）。这使该模块具有通用性。
- 与 IO 库兼容

7.4.3 运行程序

进入/pyCases/case/embedded/ttl 目录

sudo python3 uart_tgs2600.py

7.5 Python 语言 USB-RS485 程序说明

7.5.1 文件说明

Python 将 USB-RS485 案例的实现集成到了 uart_rs458.py 文件中。Python 的版本为 3.6。使用的 python pyserial 库。使用之前需要使用命令 sudo pip3 install pyserial 安装

7.5.2 接口说明

同 7.5.2 章节，不再赘述

7.5.3 运行程序

进入/pyCases/case/embedded/rs485 目录

sudo python3 uart_rs458.py

8 视频采集

OpenCV 由加里·布拉德斯基（Gary Bradsky）于 1999 年在英特尔创立，第一版于 2000 年发布。瓦迪姆·皮萨列夫斯基（Vadim Pisarevsky）与加里·布拉德斯基（Gary Bradsky）一起管理英特尔的俄罗斯软件 OpenCV 团队。2005 年，OpenCV 用于 Stanley，该车赢得了 2005 年 DARPA 大挑战赛的冠军。后来，在 Willow Garage 的支持下，其积极发展得以继续，由 Gary Bradsky 和 Vadim Pisarevsky 领导了该项目。OpenCV 现在支持与计算机视觉和机器学习有关的多种算法，并且正在日益扩展。

EAIDK 采用 opencv 库去实现视觉相关的任务。Opencv 广泛应用于各种视觉任务中。

8.1.1 文件说明

camera.py: 程序主文件

esp.json: 控制打开摄像头类别的文件

需要安装依赖：

```
sudo dnf install libjpeg-turbo-devel -y
pip3 install --user Pillow
pip3 install --user v4l2CaptureForEaidk
```

如果安装速度过慢，可以指定下载源如：

```
pip3 install --user Pillow -i https://pypi.tuna.tsinghua.edu.cn/simple
sudo dnf install gstreamer1-libav-1.14.1-1.fc28.aarch64
sudo dnf install gstreamer-rtsp-0.10.8-18.fc28.aarch64
sudo dnf install gstreamer1-plugins-bad-free-1.14.0-1.fc28.aarch64
```

8.1.2 接口说明

读取摄像头的接口主要是 cv2.VideoCapture()。

通常，我们必须使用摄像机捕获实时流。OpenCV 为此提供了一个非常简单的界面。让我们从相机捕获视频，将其转换为灰度视频并显示。只是一个简单的任务即可开始。

要捕获视频，您需要创建一个 VideoCapture 对象。它的参数可以是设备索引或视频文件的名称。设备索引只是指定哪个摄像机的编号。通常，将连接一台摄像机（EAIDK-610 有自带的 mipi 相

机）。所以我只是传递 0（或-1）。您可以通过传递 1 来选择第二台摄像机，依此类推。之后，您可以逐帧捕获。但最后，不要忘记释放捕获。

接口名称	cv2.VideoCapture(parameter)	
接口说明	Opencv处理视频对象的一个类	
参数	Parameter	设备索引或视频文件的名称
返回值	返回一个对象	

8.1.3 运行程序

进入/pyCases/case/vision/camera 目录

打开 esp.json 文件，修改“CameraType”的参数：USB 为 USB 摄像头，IPC 为网络摄像头，Mipi 为 mipi 摄像头。若选择网络摄像头，除 CameraType 外，我们仍需将网络摄像头的 IP 改为当前所使用的 IP。

```
{
    "CameraType":"USB",
    "ipc":"10.15.4.240"
}
```

python3 camera.py 运行程序，即可显示摄像头的画面。

注意，本公开教程仅支持同时支持单独摄像机。若想同时支持多个摄像机，我们需要先确认摄像机具体设备名称，再修改代码相应位置。具体操作如下：

首先，我们通过命令行查看当前所有的摄像头设备：cd /sys/class/video4linux/ && ll

```
[openailab@localhost video4linux]$ cd /sys/class/video4linux/ && ll
total 0
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video0 -> ../../devices/platform/ff910000.cif_isp/video4linux/video0
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video1 -> ../../devices/platform/ff910000.cif_isp/video4linux/video1
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video2 -> ../../devices/platform/ff910000.cif_isp/video4linux/video2
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video3 -> ../../devices/platform/ff910000.cif_isp/video4linux/video3
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video4 -> ../../devices/platform/fe380000.usb/usb5/5-1/5-1.1/5-1.1:1.0/video4linux/video4
lrwxrwxrwx 1 root root 0 Aug 12 11:03 video5 -> ../../devices/platform/fe380000.usb/usb5/5-1/5-1.2/5-1.2:1.0/video4linux/video5
```

这里，video0 至 video5 分别代表当前有的摄像头设备（文本中为 mipi 与 USB 同时连接）

以 video0 为例，我们可以通过 cat video0/name 来查看此设备名称

```
[openailab@localhost video4linux]$ cat video0/name
rkisp10_selfpath
```

若名称为我们想要打开的摄像头，那么 video0 即为设备文件名。我们到 pyCases/platform/source/vision/esp_camera.py 中，根据摄像头类型修改对应位置。若为 USB 摄像头，修改此处：

```
def init(self):
    self.cap = cv2.VideoCapture("/dev/video5")
```

若为 mipi 摄像头，修改此处数字：

```
camno = 2
```

注意，mipi 摄像头一般会有四个摄像头设备，如果打开不正确的设备可能造成图像不正常。

9 音频采集

ALSA 是 Advanced Linux Sound Architecture 的缩写，目前已经成为了 linux 的主流音频体系结构。PyAlsaAudio 软件包包含用于从 Python 访问 ALSA API 的包装器 。EAIDK-610 语音接口基于 PyAlsaAudio。

9.1.1 文件说明

代码位于 record.py 文档。Python 版本 3.5。PyAlsaAudio 需要安装。安装命令如下：

```
sudo dnf install alsa-lib-devel-1.1.6-2.fc28.aarch64
sudo pip3 install pyalsaaudio
```

9.1.2 接口说明

接口名称	alsaaudio.PCM(alsaaudio.PCM_CAPTURE, alsaaudio.PCM_NONBLOCK, device=device)	
接口说明	捕获音频流	
参数	Device	音频录入的设备
	其他参数用默认参数	
返回值	返回一个对象	

9.1.3 运行程序

进入/pyCases/case/speech/record 目录下

python3 record.py 运行程序

Control C 停止。

录制结果将被存放在 rec.pcm 文件内

文件参数：8K 采样率 16bit 单通道

10 音频播放

10.1 实验介绍

最早出现在 Linux 上的音频编程接口是 OSS（Open Sound System），它由一套完整的内核驱动程序模块组成，可以为绝大多数声卡提供统一的编程接口。OSS 出现的历史相对较长，这些内核模块中的一部分（OSS/Free）是与 Linux 内核源码共同免费发布的，另外一些则以二进制的形式由 4Front Technologies 公司提供。由于得到了商业公司的鼎力支持，OSS 已经成为在 Linux 下进行音频编程的事实标准，支持 OSS 的应用程序能够在绝大多数声卡上工作良好。

虽然 OSS 已经非常成熟，但它毕竟是一个没有完全开放源代码的商业产品，ALSA（Advanced Linux Sound Architecture）恰好弥补了这一空白，它是在 Linux 下进行音频编程时另一个可供选择的声卡驱动程序。ALSA 除了像 OSS 那样提供了一组内核驱动程序模块之外，还专门为简化应用程序的编写提供了相应的函数库，与 OSS 提供的基于 ioctl 的原始编程接口相比，ALSA 函数库使用起来要更加方便一些。ALSA 的主要特点有：

- 支持多种声卡设备
- 模块化的内核驱动程序
- 支持 SMP 和多线程
- 提供应用开发函数库
- 兼容 OSS 应用程序

随着 Linux 平台下多媒体应用的逐渐深入，需要用到数字音频的场合必将越来越广泛。虽然数字音频牵涉到的概念非常多，但在 Linux 下进行最基本的音频编程却并不十分复

杂，关键是掌握如何与 OSS 或者 ALSA 这类声卡驱动程序进行交互，以及如何充分利用它们提供的各种功能，熟悉一些最基本的音频编程框架和模式对初学者来讲大有裨益。

10.2 文件说明

播放案例代码放于/pyCases/speech/playsound 目录下

- playalsa.py: 通过 alsa 实现播放功能的程序文件
- playsox.py: 通过 sox 实现播放功能的程序文件
- esp_demo.wav:案例音频
- input.wav:案例音频
- voicebaidu.mp3:案例音频

程序所需要的依赖

```
sudo dnf install ffmpeg
sudo pip3 install pydub
sudo pip3 install pyaudio
pip3 install pyalsaaudio --user
pip3 install sox –user
sudo dnf install sox-devel-14.4.2.0-20.fc28.aarch64
```

10.3 运行程序

将音箱连接到 610 上

进入案例目录/pyCases/case/speech/playsound，用 python3 playalsa.py 运行 alsa 播放案例，用 python3 playsox.py 运行 sox 播放案例