ED-AIC2000

用户手册

by EDA Technology Co., Ltd

built: 2024-11-20

1 硬件手册

本章介绍产品概述、包装清单、外观、按键、指示灯和接口等。

1.1 产品概述

ED-AIC2000是一款基于Raspberry Pi CM4的工业智能相机，采样率高达70 FPS，根据不同的应用场景和用户需求，可选择不同规格的RAM、eMMC、像素和镜头。

- RAM可选规格包含1GB、2GB、4GB和8GB
- eMMC闪存可选规格包含8GB、16GB和32GB
- 像素可选规格包含200万像素、230万像素和1200万像素
- 镜头可选规格包含M12固定焦距镜头和带液态模组变焦的M12固定焦距镜头

ED-AIC2000集成模块光源设计，光源支持3种颜色可选；采用亮场和暗场模式，可在正常、蚀刻、高光或纹理化表面实现最佳打光效果。

ED-AIC2000系列包含ED-AIC2000-020、ED-AIC2000-023和ED-AIC2000-120三个型号，提供电源接口、IO 接口、 RS232 串口和千兆以太网等接口，采用M12航空连接器，支持 IP65 防水等级，支持通过以太网接入网络，主要应用于机器视觉和人工智能领域。

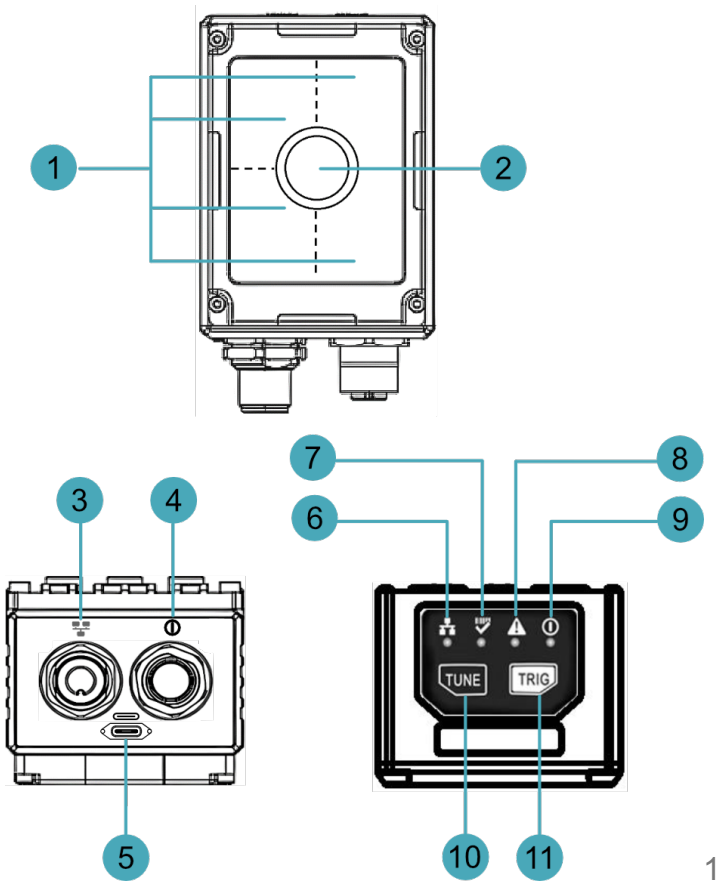
1.2 包装清单

1 x ED-AIC2000主机

1.3 产品外观

介绍产品接口、按键和指示灯的功能和定义。

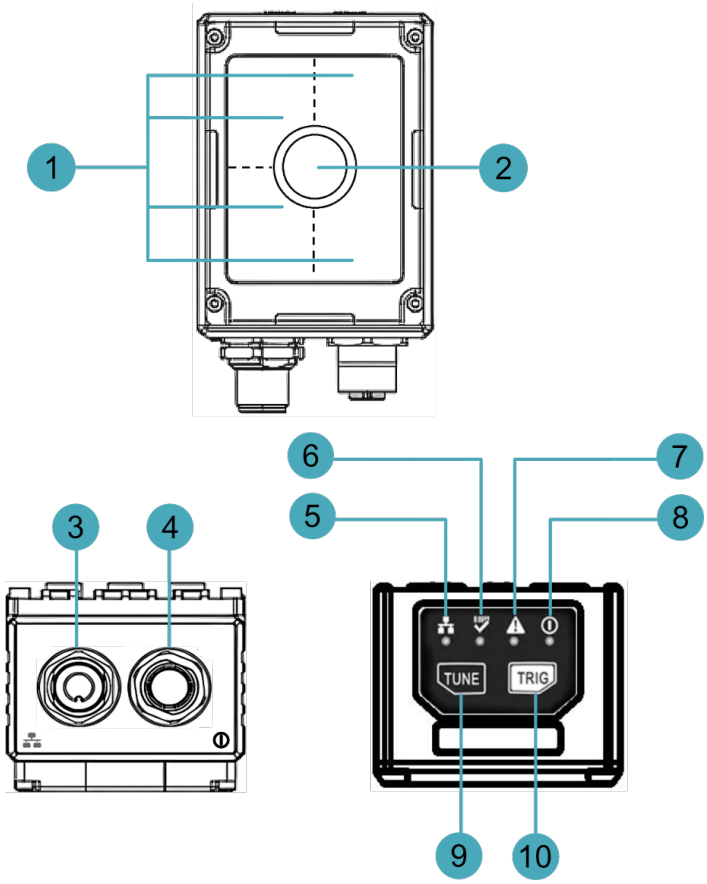
ED-AIC2000-020



1

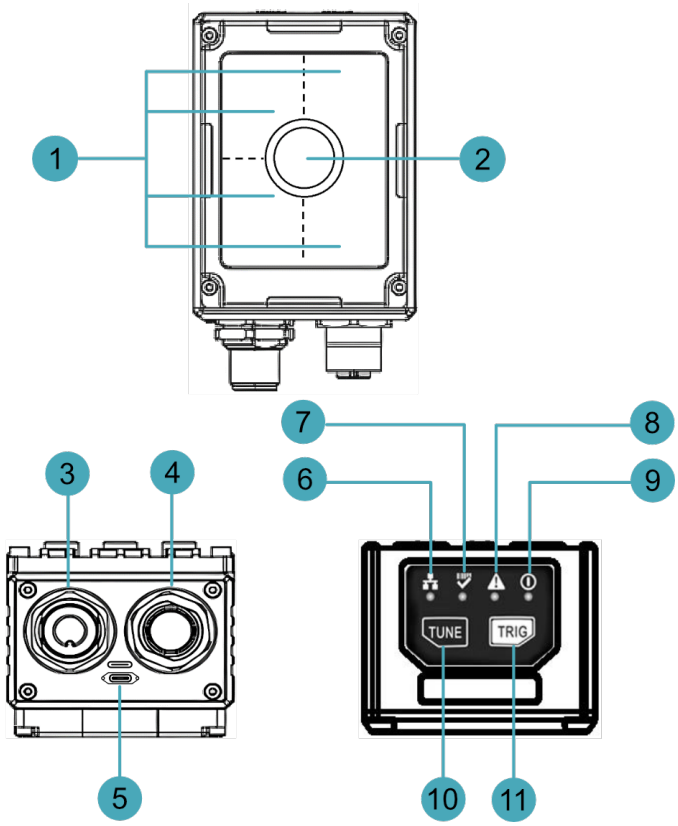
编号	功能定义
1	4 x 光源，用于设备工作时补光。
2	1 x 镜头，用于拍照。
3	1 x 通信接口，千兆以太网接口，采用M12 8-Pin A-code航空连接器，用于接入以太网。
4	1 x 电源接口，包含电源接口、IO接口和RS232串口，采用M12 12-Pin航空连接器。
5	1 x USB-c接口，用于eMMC烧录
6	1 x 网络连接指示灯，用于查看网络连接的状态。
7	1 x 工作状态指示灯，用于查看设备工作的状态。
8	1 x 系统故障指示灯，用于查看系统是否发生故障。
9	1 x 电源指示灯，用于查看设备上电状态。
10	1 x 调节按键，一键式自动对焦按钮或用户自定义按钮
11	1 x 触发按键，用于相机触发或用户定义的一键式按钮

ED-AIC2000-023



编号	功能定义
1	4 x 光源，用于设备工作时补光。
2	1 x 镜头，用于拍照。
3	1 x 通信接口，千兆以太网接口，采用M12 8-Pin A-code航空连接器，用于接入以太网。
4	1 x 电源接口，包含电源接口、IO接口和RS232串口，采用M12 12-Pin航空连接器。
5	1 x 网络连接指示灯，用于查看网络连接的状态。
6	1 x 工作状态指示灯，用于查看设备工作的状态。
7	1 x 系统故障指示灯，用于查看系统是否发生故障。
8	1 x 电源指示灯，用于查看设备上电状态。
9	1 x 调节按键，一键式自动对焦按钮或用户自定义按钮
10	1 x 触发按键，用于相机触发或用户定义的一键式按钮

ED-AIC2000-120



编号	功能定义
1	4 x 光源，用于设备工作时补光。
2	1 x 镜头，用于拍照
3	1 x 通信接口，千兆以太网接口，采用M12 8-Pin A-code航空连接器，用于接入以太网。
4	1 x 电源接口，包含电源接口、IO接口和RS232串口，采用M12 12-Pin航空连接器。
5	1 x type-c接口，用于eMMC烧录
6	1 x 网络连接指示灯，用于查看网络连接的状态。
7	1 x 工作状态指示灯，用于查看设备工作的状态。
8	1 x 系统故障指示灯，用于查看系统是否发生故障。
9	1 x 电源指示灯，用于查看设备上电状态。
10	1 x 调节按键，一键式自动对焦按钮或用户自定义按钮
11	1 x 触发按键，用于相机触发或用户自定义的一键式按钮

1.4 按键

ED-AIC2000系列设备包含2个按键，调节按键和触发按键。

- 调节按键，在外壳上的丝印为“TUNE”，按下按键可以一键式自动对焦，支持用户自定义其功能。

• 触发按键，在外壳上的丝印为“TRIG”，按下按键可以触发相机，支持用户自定义其功能。

引脚定义

按键引脚定义如下：

按键	CM4 引脚
调节按键	GPIO20
触发按键	GPIO12

1.5 指示灯

介绍ED-AIC2000系列设备包含的指示灯的各种状态及含义。

指示灯	状态	描述
网络连接指示灯	常亮	已正常接入以太网
	熄灭	未接入以太网
工作状态指示灯	闪烁	系统工作状态正常
	熄灭	系统工作状态异常
系统故障指示灯	闪烁	系统出现故障
	熄灭	系统未出现故障
电源指示灯	常亮	设备已上电
	熄灭	设备未上电

引脚定义

指示灯	CM4 引脚
电源指示灯	N/A
故障指示灯	GPIO21
工作状态指示灯	GPIO7(异常) GPIO16(正常)
网络指示灯	N/A

1.6 接口

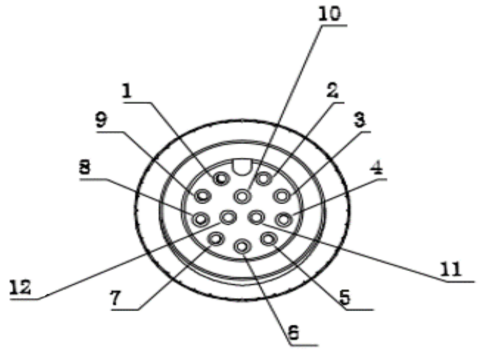
介绍产品中各接口的定义和功能。

1.6.1 电源接口

ED-AIC2000系列设备包含1路电源接口、1路串口和6路通用GPIO，电源接口采用M12 12-Pin航空连接器，支持10V~30V输入，信号定义为DC-/DC+。

引脚定义

引脚定义如下：



Pin ID	Pin Name
1	DC-
2	DC+
3	COMMON_IN
4	DI1
5	Trigger
6	COMMON_OUT
7	External Strobe
8	DO1
9	DO2
10	RS232_GND
11	RS232_TX
12	RS232_RX

1.6.2 DI和DO接口

ED-AIC2000系列设备包含一路通用输入端DI1，两路通用输出端DO1和DO2，支持10V~30V DC输入输出，具备光电隔离和过流保护。

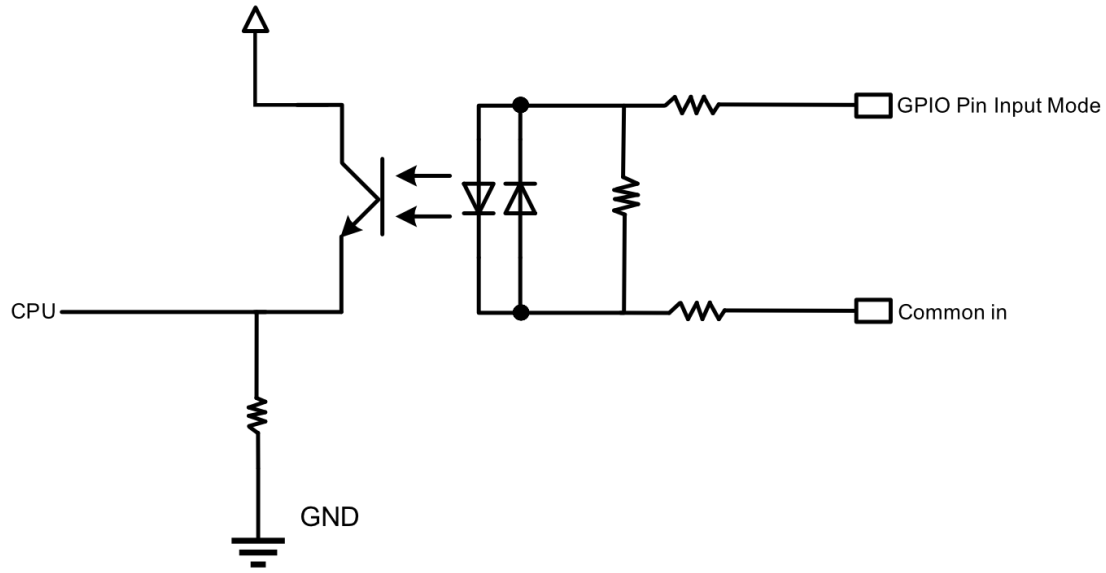
引脚定义

DI和DO引脚定义如下：

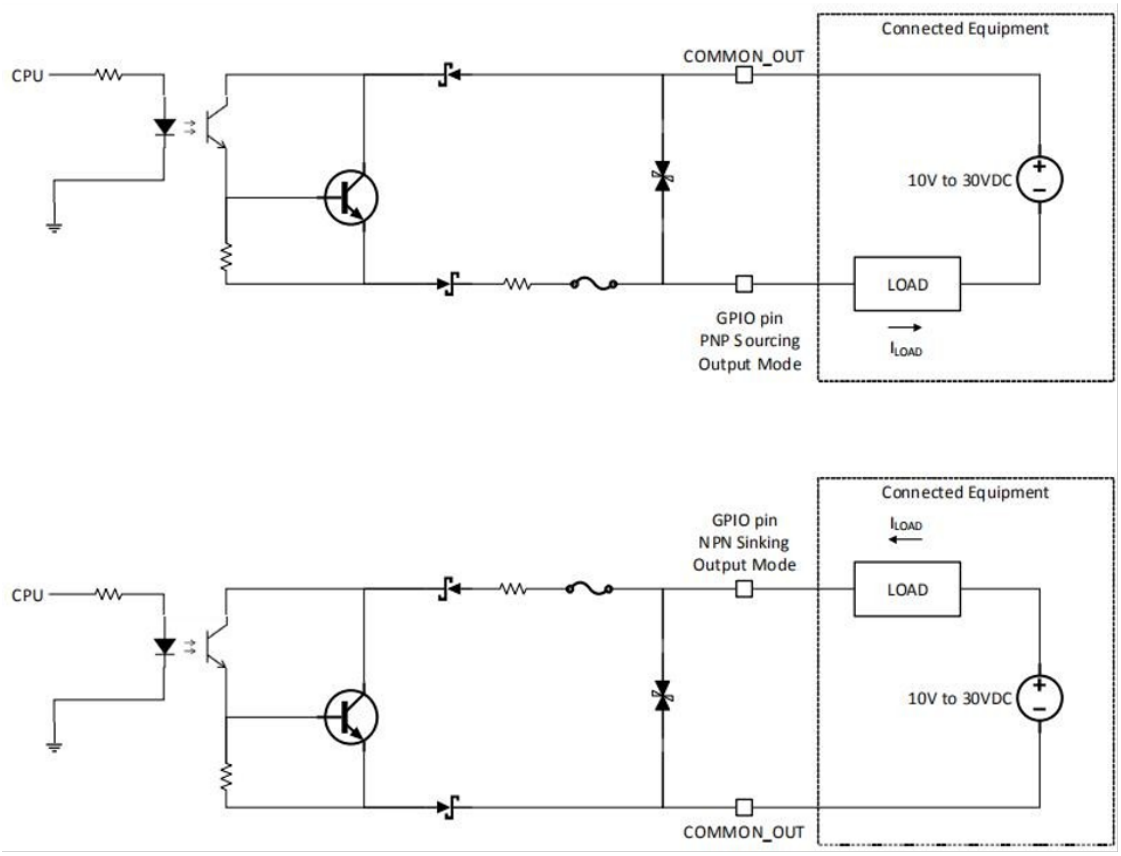
Signal	CM4 GPIO NAME
DI1	GPIO17
DO1	GPIO22
DO2	GPIO27

连接线缆

单路DI接口的接线示意图如下：



两路DO接口的接线示意图如下：



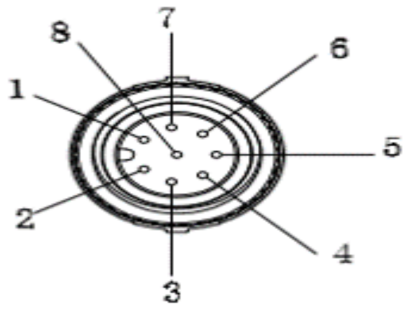
1.6.3 通信接口

ED-AIC2000系列设备包含1路通信接口，采用M12 8-Pin航空连接器。

引脚定义

引脚定义如下：

	Pin ID	Pin Name



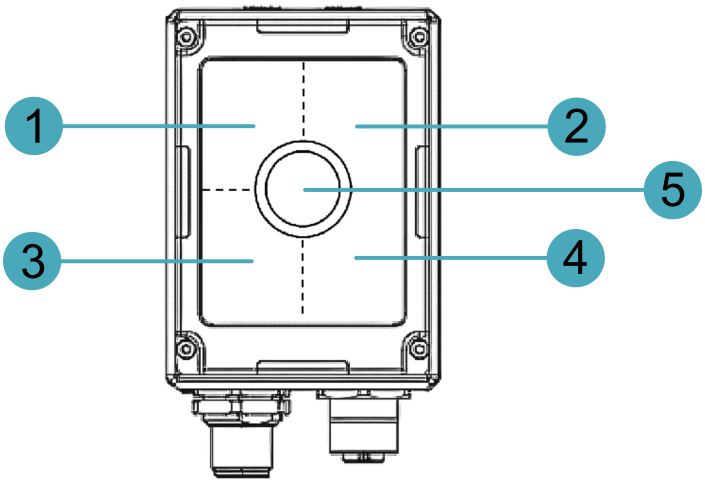
1	TRD0+
2	TRD0-
3	TRD1+
4	TRD2+
5	TRD2-
6	TRD1-
7	TRD3+
8	TRD3-

1.6.4 USB-c接口

ED-AIC2000-020和ED-AIC2000-120均包含1路USB type-c接口，支持通过连接PC对设备的eMMC进行烧录。

1.7 光源和镜头

ED-AIC2000系列设备包含四部分光源和一个镜头。



编号	描述
1	光源部分 1，支持单独启用和禁用。
2	光源部分 2，支持单独启用和禁用。
3	光源部分 3，支持单独启用和禁用。
4	光源部分 4，支持单独启用和禁用。
5	镜头，M12固定焦距镜头和带液态模组变焦的M12固定焦距镜头。

光源控制命令

控制命令如下：

信号	CM4 引脚(GPIO 8/9，/dev/ttyAMA2)
控制光源部分 1	RS232命令 • 启用：1-1 • 禁用：1-0
控制光源部分 2	RS232命令 • 启用：2-1 • 禁用：2-0
控制光源部分 3	RS232命令 • 启用：3-1 • 禁用：3-0
控制光源部分 4	RS232命令 • 启用：4-1 • 禁用：4-0

2 安装设备

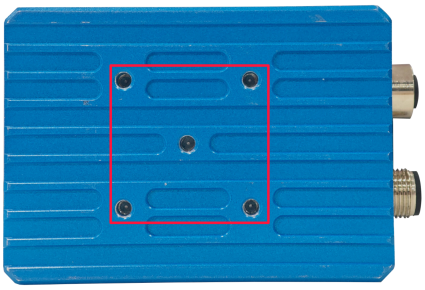
本章介绍安装设备的具体操作。

前提条件：

- 已准备安装支架及安装螺钉(M4*8带垫片、M6*10带垫片)
- 已准备M4和M6的内六角螺丝刀

操作步骤：

1. 确定相机上安装孔的位置，如下图红框位置，一般用中央螺钉和一颗外围螺钉固定即可。



2. 将安装支架放置在相机安装孔的上方，使支架(带M4螺孔的一侧)和相机的中央螺丝孔位对齐，通过螺丝刀使用带垫圈的M4螺丝将支架固定在相机上，如下图所示。

3. 旋转支架，调整安装方向，按需选择顶部安装或者侧面安装，效果如下图：

4. 使用M6螺钉固定支架和其他设备，建议使用中心螺钉和一颗外围螺钉固定。

> 提示
>
> 根据现场不同的工程要求，调整合适的安装位置和角度。

3 启动设备

本章介绍连接线缆和启动设备的具体操作。

3.1 连接线缆

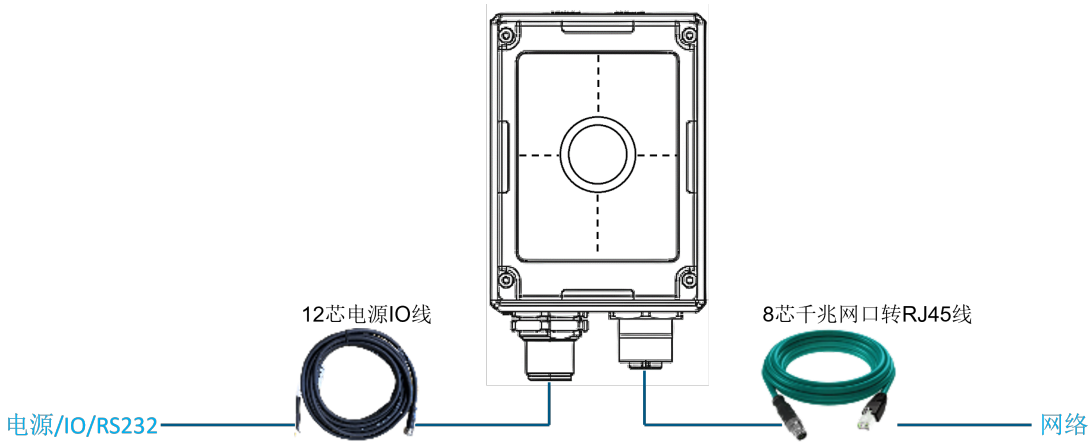
介绍线缆的连接方法。

准备工作：

- 已获取待连接的8芯千兆网口转RJ45线和12芯电源IO线。
- 已准备DC 12V 2A的电源适配器和辅助接线的连接器。
- 按需准备12芯电源IO线所需的连接器和接线工具。

连接线缆示意图：

各接口的引脚定义及连线的具体方法，请参见1.6接口。



3.2 首次启动系统

ED-AIC2000系列设备无电源开关，接入电源后，系统将会开始启动。

- 电源指示灯常亮，表示设备已正常供电。
- 工作状态指示灯常亮，表示系统正常启动。

系统启动后，默认使用用户名和密码进行登录，因为相机无法连接显示器，故需要通过PC远程登录系统。
ED-AIC2000系列设备出厂镜像默认使能VNC，并将设备IP设置为静态IP：192.168.1.10，用户可以通过VNC来远程连接设备，并进入设备的桌面系统。具体的操作参考通过VNC远程连接到桌面。

> 提示
>
> 默认用户名：pi；默认密码：raspberry。

4 系统配置

本章介绍系统配置的具体操作。

4.1 编译Camera演示

如何编译启动测试Camera示例。

1. 进入到测试文件所在的文件夹。

```
cd /usr/share/ed-aic-lib/examples
```

2. 编译测试代码

```
sudo make test_camera.cpp
```
或
```
python test_camera.py
```

```
sudo ./test_camera
```

4.2 Camera接口

- “camera.h“文件提供了如何操作AI Camera Sensor的控制接口，该文件的存放路径为”/usr/include/eda/camera.h“。
- "eda-io.h"文件提供了如何操作AI Camera IO的控制接口，该文件的存放路径为”/usr/include/eda/eda-io.h“。

4.3 库文件

- “libeda_camera.so”库文件的存放路径为”/usr/lib/libeda_camera.so“。
- ”libeda_io.so“库文件的存放路径为”/usr/lib/libeda_io.so“。

4.4 Camera API

Camera API提供了AI Camera Sensor的控制函数，支持打开和关闭传感器、设置传感器工作模式、设置曝光时间和设置增益，下文分别按照不同的Camera场景进行说明。

控制函数

Camera API提供的AI Camera Sensor的控制函数如下表，包含获取图像数据、打开摄像头、关闭摄像头、设置工作模式、设置曝光时间和设置增益等。

- AR0234表示230万像素的Camera
- OV2311表示200万像素的Camera

函数	功能	参数取值
	获取图像数据	-

函数	功能	参数取值
virtual int callback_image_ready(img_Callback callback)=0		
virtual int open(int mode, int width, int height) = 0	打开并配置摄像头	工作模式取值包含0、1、5 • 0表示连续模式（一直打开摄像头），AR0234和OV2311均支持。 • 1表示硬件触发模式，AR0234和OV2311均支持，通过Trigger引脚接5V触发。 • 5表示软件触发模式，仅OV2311支持，通过手动调节来触发。
virtual int close() = 0	关闭摄像头	-
virtual int set_exposure(int exp_value) = 0	设置曝光时间	• OV2311：t_exposure取值范围为0~65523，单位为us • AR0234：t_exposure取值范围为1~1500，exposure的值乘以6.8后的值的单位是us
virtual int get_exposure(int *exp_value) = 0	获取曝光时间	-
virtual int set_gain(int gain_value) = 0	设置增益	• OV2311：t_gain取值范围为0~30 • AR0234：t_gain取值范围为0~64
virtual int get_gain(int *gain_value) = 0	获取增益	-

获取AR0234实例

```C++
#include "CameraManger.h"
#include "camera_0234.h"

void test()
  eda::Camera *t_camera = eda::create_ar0234();
  if(t_camera){
  eda::Camera_0234 *t_camera_1 = static_cast<eda::Camera_0234*>(t_camera);
  }
  ...
}
```

4.5 IO API

IO API提供了AI Camera IO的控制函数，支持控制指示灯、控制激光、控制侧灯和控制输出等。

4.5.1 C语言环境

AI Camera IO在C语言环境下的控制函数如下所示：

函数	功能
eda::EdaIo *em = eda::EdaIo::getInstance()	获取IO控制实例
void setup()	初始化IO设置
void openLaser()	打开激光
void closeLaser()	关闭激光
void setScanStat(bool good)	设置状态指示灯
void openAlarm()	打开警告指示灯
void closeAlarm()	关闭警告指示灯
void setDo1High(bool high)	设置output1输出
void setDo2High(bool high)	设置output2输出
void registerInput(IoTrigger callback)	注册input触发回调函数
void registerTrigger(IoTrigger callback)	注册trigger按键回调函数
void registerTune(IoTrigger callback)	注册Tune按键回调函数
void setRgbLight(uint8_t light)	设置RGB灯光

4.5.2 Python3语言环境

AI Camera IO在Python3语言环境下的控制函数如下所示：

函数	功能
eda = EdaIo.singleton()	获取IO控制实例
eda.setup()	初始化
eda.openLaser()	打开激光
eda.closeLaser()	关闭激光
eda.setScanStat(True)	设置状态指示灯
eda.openAlarm()	打开警告指示灯
eda.closeAlarm()	关闭警告指示灯
eda.setDo1High(True)	设置output1输出
eda.setDo2High(False)	设置output2输出
registerInput(func_trigger)	注册input触发回调函数
registerTrigger(func_trigger)	注册trigger按键回调函数
registerTune(func_trigger)	注册Tune按键回调函数

函数	功能
eda.setRgbLight(1)	设置RGB灯光

4.5.3 操作说明

1. 初始化

- 操作IO前需要获取IO实例 `eda::EdaIo *em = eda::EdaIo::getInstance();`
- 对实例初始化 `em->setup();`

2. IO控制支持对事件注册回调函数

- input 输入事件 `em->registerInput(trigger_input);`
- Trigger 按键 `em->registerTrigger(trigger_trigger);`
- Tune 按键 `em->registerTune(trigger_tune);`

3. 控制IO（必须先完成初始化）

- 控制激光 `em->openLaser()` 和 `em->closeLaser();`
- 控制状态指示灯 `em->setScanStat(true)` 和 `em->setScanStat(false);`
- 控制警报指示灯 `em->openAlarm()` 和 `em->openAlarm();`
- 控制两路输出 `em->setDo1High(false)` 和 `em->setDo2High(false);`

4. 控制灯光（必须先完成初始化）

- 控制侧灯颜色 `em->setRgbLight(1);`
  - 0: 关闭
  - 1: 红色
  - 2: 绿色
  - 3: 蓝色
- 控制灯源
  - 使能灯源（默认已使能） `em->enableLightSection(1)` 取值范围1~4，对应不同分区
  - 禁用灯源 `em->disableLightSection(1)` 取值范围1~4，对应不同分区

5 安装Raspberry Pi OS

设备出厂时，默认已安装操作系统。如果在使用过程中操作系统被损坏或者用户需要更换操作系统，则需要重新下载对应的系统镜像并进行烧录。

下文介绍镜像下载、eMMC烧录的具体操作。

5.1 镜像下载

可根据实际的需要下载对应型号的ED-AIC2000系统镜像，下载路径如下表。

产品型号	下载路径
ED-AIC2000-020	2024-08-14_ed-aic2000-200w_arm64/ (https://vip.123pan.cn/1826505135/7439075)
ED-AIC2000-023	2024-08-14_ed-aic2000-230w_arm64/ (https://vip.123pan.cn/1826505135/7439077)
ED-AIC2000-120	2024-08-15_ed-aic2000-1200w_arm64/ (https://vip.123pan.cn/1826505135/7444871)

5.2 eMMC烧录

建议使用Raspberry Pi官方烧录工具，下载路径如下：

- Raspberry Pi Imager : https://downloads.raspberrypi.org/imager/imager_latest.exe (https://downloads.raspberrypi.org/imager/imager_latest.exe)
- SD Card Formatter : https://www.sdcardformatter.com/download/ (https://www.sdcardformatter.com/download/)
- Rpiboot : https://github.com/raspberrypi/usbboot/raw/master/win32/rpiboot_setup.exe (https://github.com/raspberrypi/usbboot/raw/master/win32/rpiboot_setup.exe)

前提条件：

- 已完成烧录工具的下载，并安装至电脑。
- 已准备一根USB-C转USB-A的烧录线。
- 已准备一根8-Pin M12公头转RJ45的网线。
- 已准备一根12-Pin M12母头转裸线的电源IO线。
- 已获取待烧录的镜像文件。

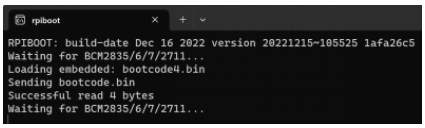
操作步骤：

操作步骤以Windows系统为例进行说明。

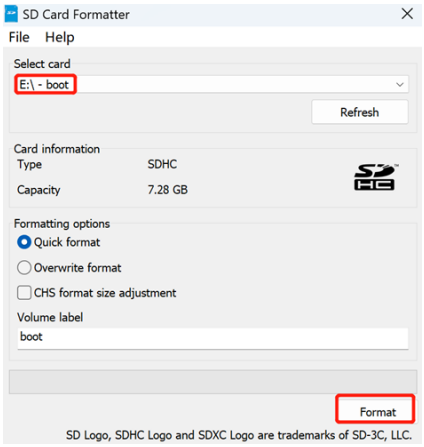
1. 连接好电源线和USB烧录线（USB-C转USB-A）。

- 连接烧录线：一端连接设备侧的USB type-C接口，另一端连接PC上的USB接口。
- 连接电源线：一端连接设备侧的M12 12-Pin电源接口 (下图红框中的接口)，另一端连接外部电源。

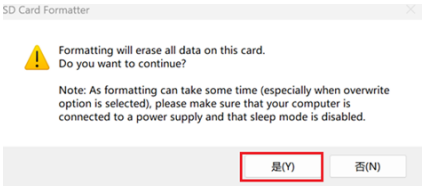
2. 断开设备电源，再按住TRIG按键，对设备重新上电，设备会自动进入烧录模式。
3. 打开已安装的rpiboot工具，自动进行盘符化。



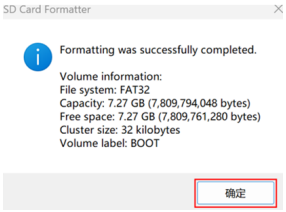
4. 待盘符化完成后，电脑右下角会弹出盘符。
5. 打开SD Card Formatter，选择被格式化的盘符，单击右下方“Format”进行格式化。



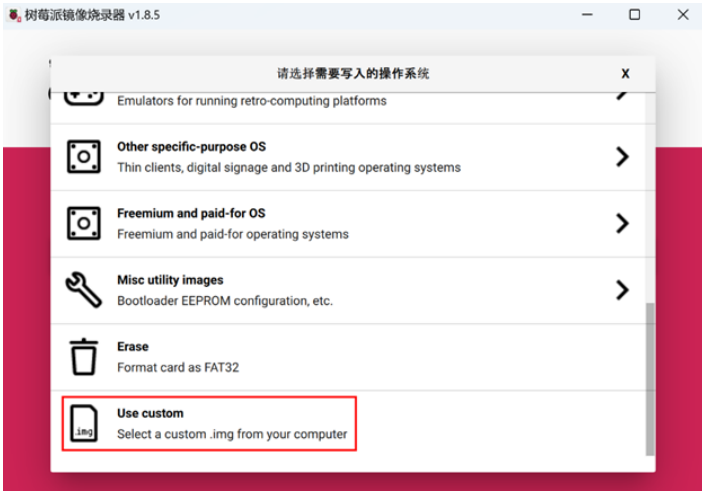
6. 在弹出的提示框中，单击“是”。



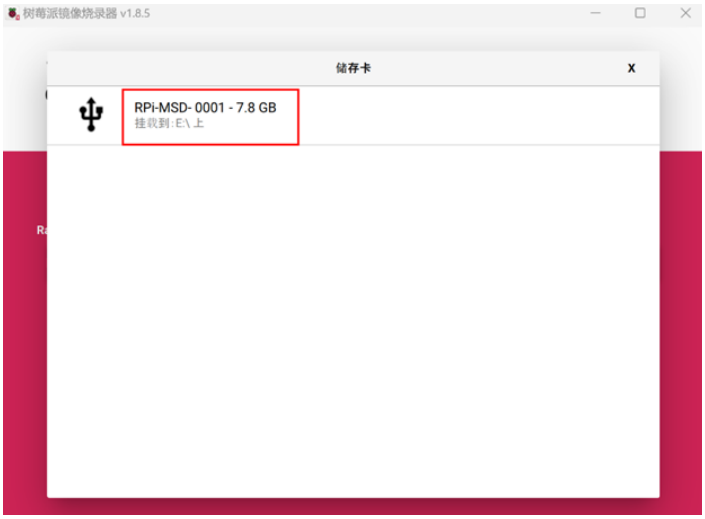
7. 格式化完成后，在提示框中单击 “确定”。



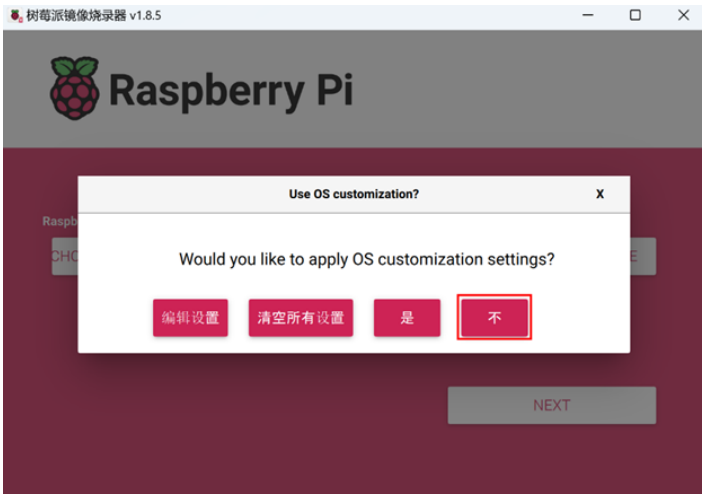
8. 关闭SD Card Formatter。
9. 打开Raspberry Pi Imager，单击“选择操作系统”，在弹出的窗格中选择“Use custom”。



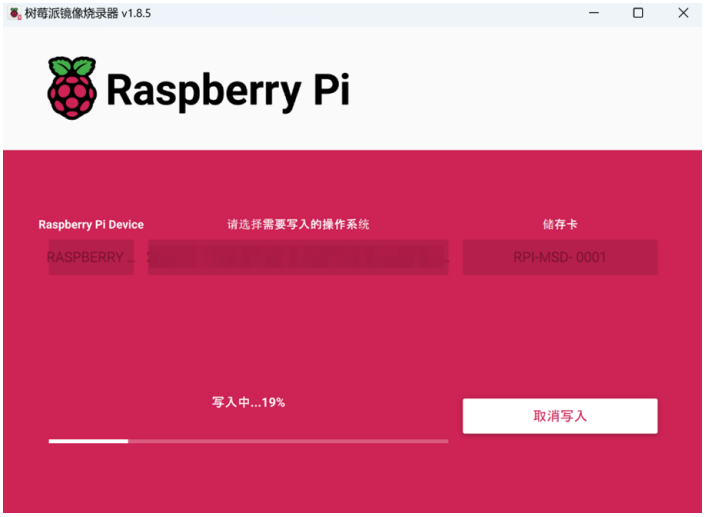
10. 根据提示，在自定义路径下选择已获取的镜像文件，并返回至烧录主界面。
11. 单击“选择SD卡”，在“存储卡”界面选择默认的SD卡，并返回至烧录主界面。



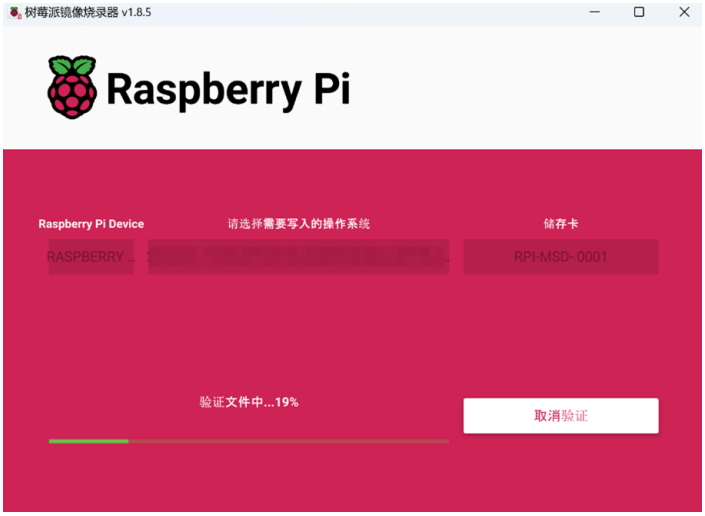
12. 单击“NEXT”，在弹出的“Use OS customization？”提示框中选择“不”，开始写入镜像。



13. 在弹出的“警告”提示框中选择‘”是“，开始写入镜像。



14. 待镜像写入完成后，会进行文件的验证。



15. 验证完成后，弹出“烧录成功”提示框，单击“继续”完成烧录。

16. 关闭Raspberry Pi Imager，取下USB烧录线，重新给设备上电。

6 SDK开发指南

1 SDK概述

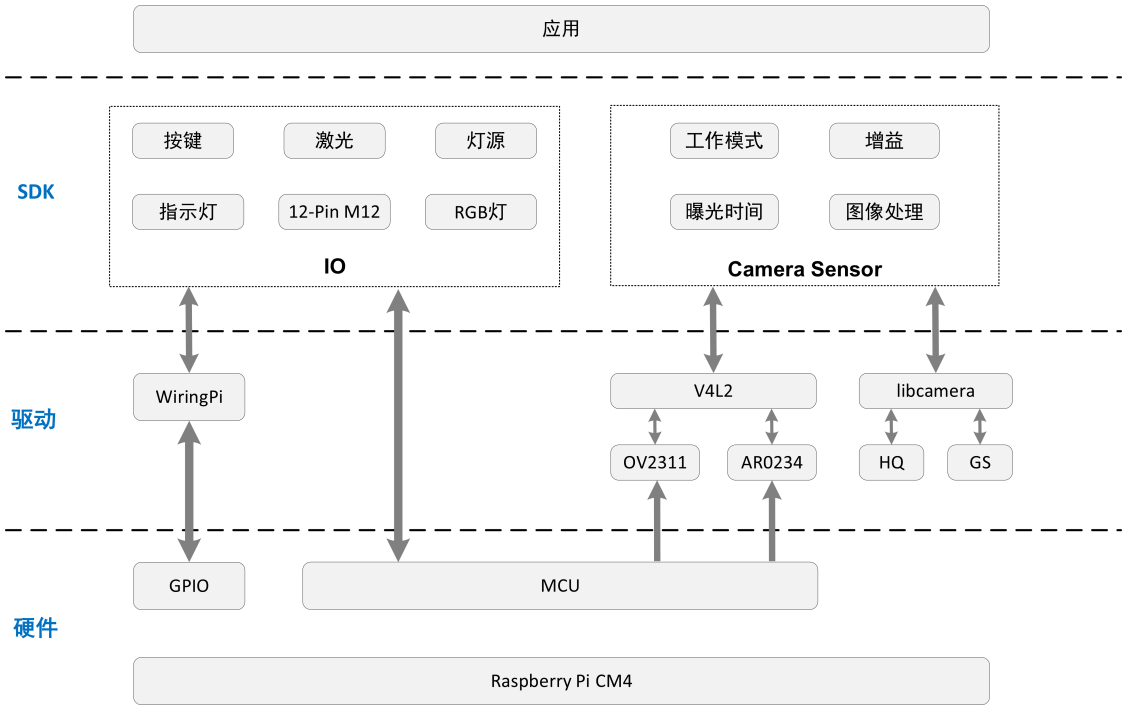
本章介绍SDK的定义和组成，帮助用户更好的了解SDK。

1.1 SDK 简介

ED-AIC2000系列Camera的SDK是一组软件工具开发包（Software Development Kit），给用户提供上层应用所需的接口，便于对Camera进行二次开发。

ED-AIC2000系列Camera的SDK功能包含Trigger/Tune按键的定义、12-Pin M12接口中的DI的定义、激光开关的控制、状态指示灯的控制、报警指示灯的控制、2路DO的控制、灯光和灯源的控制、相机工作模式的设置、相机曝光时间的设置、相机增益的设置和图像数据的处理。

SDK在整个Camera系统中的位置如下图所示。



1.2 SDK组成

Camera的SDK是由多个头文件和库文件组成的，具体文件名和安装路径如下表。

功能类型	文件类型	文件名	安装路径
IO控制	头文件	eda-io.h	/usr/include/eda/
	库文件	libeda_io.so	/usr/lib/
Camera Senso控制	头文件	camera.h	/usr/include/eda/

功能类型	文件类型	文件名	安装路径
		CameraManger.h	
		camera_0234.h	
		camera_2311.h	
	库文件	libeda_camera.so	/usr/lib/

在开发过程中用户可以根据实际需要实现的功能，参考下文对应的功能代码来完成上层应用的开发。

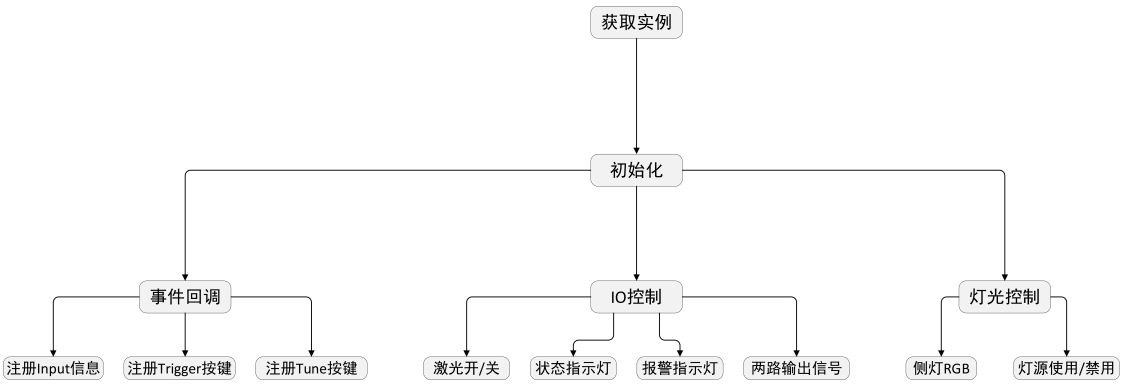
2 功能说明

本章介绍各项功能对应代码的编写方法，帮助用户编写上层应用所需要的代码。

2.1 IO控制(C++)

本节介绍指示灯控制、激光控制、事件监听和输出控制等的具体操作。

2.1.1 流程图



2.1.2 获取实例并初始化

在操作IO前需要先获取IO实例并对实例进行初始化，操作步骤如下。

1. 获取IO实例。

```
eda::EdaIo *em = eda::EdaIo::getInstance();
```

2. 对实例进行初始化。

```
em->setup();
```

2.1.3 事件回调

IO控制支持对事件注册回调函数，包含注册Input信息、注册Trigger按键和注册Tune按键。

- DI1触发事件

`em->registerInput(trigger_input);`

12-Pin M12接口中的COMMON_IN引脚接地，DI1引脚接5V触发

- 注册Trigger按键

`em->registerTrigger(trigger_trigger);`

- 注册Tune按键

`em->registerTune(trigger_tune);`

举例：

```c++
#include "eda/eda-io.h"
void trigger_input(int b){
    printf("[Test] Tirgger input: %d\n", b);
}
int main(int argc, char *argv[]){
    eda::EdaIo *em = eda::EdaIo::getInstance();
    em->registerInput(trigger_input);
    em->setup();
    ....
}
```

2.1.4 控制IO

通过IO来控制激光的开/关、状态指示灯的点亮/熄灭、报警指示灯的点亮/熄灭和2路输出信号的使能/禁用。

前提条件：

已完成实例的初始化。

操作说明：

- 控制激光

`em->openLaser();`

`em->closeLaser();`

- 控制状态指示灯

`em->setScanStat(true);`

`em->setScanStat(false);`

- 控制警报指示灯

`em->openAlarm();`

`em->openAlarm();`

- 控制2路输出信号

`em->setDo1High(false);`

`em->setDo2High(false);`

2.1.5 控制灯光

Camera侧面灯和区域灯均可独立控制。

前提条件：

已完成实例的初始化。

操作说明：

- 控制侧灯颜色
  `em->setRgbLight(1);`
  - 0: 关闭
  - 1: 红色
  - 2: 绿色
  - 3: 蓝色
- 控制侧灯RGB颜色

`void setRgbLight_rgb(uint8_t r, uint8_t g, uint8_t b);`

- 控制灯源
  - 使能（默认状态为使能）
  `em->enableLightSection(1);`
  取值范围为1~4，分别对应不同的分区
  - 禁用
  `em->disableLightSection(1);`
  取值范围为1~4，分别对应不同的分区

灯源的使能/禁用不是打开/关闭灯源，灯源与摄像头是联动的，只有当灯源已使能且摄像头打开的条件下灯源才会亮。

2.1.6 源文件

IO控制Class（C++语言）

```c++
typedef void (*IoTrigger)(int level);
```

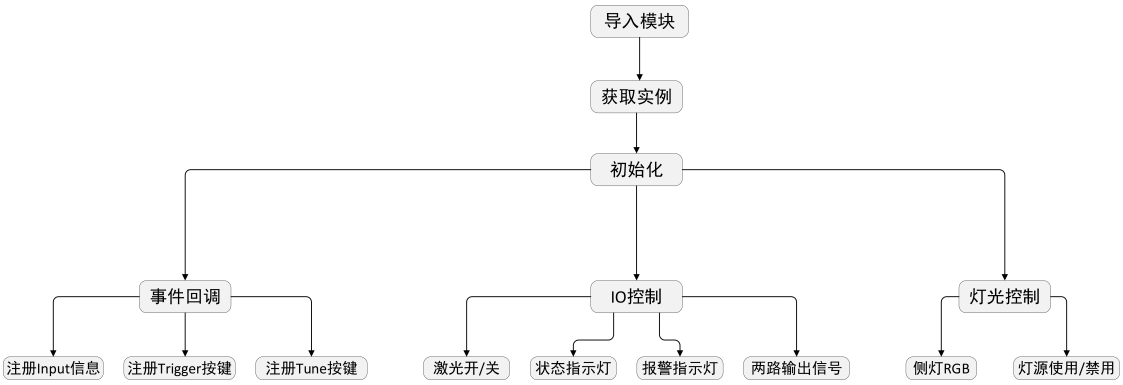
```
class EdaIo{
public:
    static EdaIo* getInstance();
    static void close_io();
    ~EdaIo();
    /**
     * @brief 打开激光
     *
     */
    void openLaser();
    /**
     * @brief 关闭激光
     *
     */
    void closeLaser();
    /**
     * @brief 设置状态指示灯
     *
     * @param good
     */
    void setScanStat(bool good);
    /**
     * @brief 打开alarm 指示灯
     *
     */
    void openAlarm();
    /**
     * @brief 关闭alarm 指示灯
     *
     */
    void closeAlarm();
    /**
     * @brief
     *
     * @param section 1~4
     * @return int
     */
    int enableLightSection(int section);
    /**
     * @brief
     *
     * @param section 1~4
     * @return int
     */
    int disableLightSection(int section);
    /**
     * @brief 设置output1 输出 [高/低]
     *
     * @param high
     */
    void setDo1High(bool high);
```

```
    /**
     * @brief 设置output2 输出 [高/低]
     *
     * @param high
     */
    void setDo2High(bool high);
    // void setAimerColor(RGBColor color);
    /**
     * @brief 注册input触发回调函数
     *
     * @param callback
     */
    void registerInput(IoTrigger callback);
    /**
     * @brief 注册register按键 回调函数
     *
     * @param callback
     */
    void registerTrigger(IoTrigger callback);
    /**
     * @brief 注册Tune按键 回调函数
     *
     * @param callback
     */
    void registerTune(IoTrigger callback);
    /**
     * @brief set RGB light
     *
     * @param light 0: Close; 1: Red; 2: Green; 3: Blue,
     * @return int
     */
    void setRgbLight(uint8_t light);
    /**
     * @brief Set the RGB Light
     *
     * @param r red
     * @param g green
     * @param b blue
     */
    void setRgbLight_rgb(uint8_t r, uint8_t g, uint8_t b);
    /**
     * @brief 初始化IO 设置
     *
     */
    void setup();
};
```

2.2 IO控制(Python)

本节介绍指示灯控制、激光控制、事件监听和输出控制等的具体操作。

2.2.1 流程图



2.2.2 导入模块

在操作IO前需要先导入模块。

```
from libedaio import EdaIo,registerInput,registerTrigger,registerTune
```

2.2.3 获取实例并初始化

在操作IO前需要先获取IO实例并对实例进行初始化，操作步骤如下。

1. 获取IO实例。

```
eda = EdaIo.singleton();
```

2. 对实例进行初始化。

```
eda.setup();
```

2.2.4 事件回调

IO控制支持对事件注册回调函数，包含注册Input信息、注册Trigger按键和注册Tune按键。

- DI1触发事件

```
registerInput(func_input);
```

12-Pin M12接口中的COMMON_IN引脚接地，DI1引脚接5V触发

- 注册Trigger按键

```
registerTrigger(func_trigger);
```

- 注册Tune按键

```
registerTune(func_tune);
```

举例：

```c++
#!/usr/bin/python3
from libedaio import EdaIo,registerInput
def func_input(v):
    print("[Debug] Trigger: input!", v)
def main() -> int:
    eda = EdaIo.singleton();
    registerInput(func_input)
eda.setup()
…
if __name__ == "__main__":
main()
```

2.2.5 控制IO

通过IO来控制激光的开/关、状态指示灯的点亮/熄灭、报警指示灯的点亮/熄灭和2路输出信号的使能/禁用。

前提条件：

已完成实例的初始化。

操作说明：

- 控制激光

`eda.openLaser();`

`eda.closeLaser();`

- 控制状态指示灯

`eda.setScanStat(true);`

`eda.setScanStat(false);`

- 控制警报指示灯

`eda.openAlarm();`

`eda.openAlarm();`

- 控制2路输出信号

`eda.setDo1High(false);`

`eda.setDo2High(false);`

2.2.6 控制灯光

Camera侧面灯和区域灯均可独立控制。

前提条件：

已完成实例的初始化。

操作说明：

- 控制侧灯颜色
  `eda.setRgbLight(1);`
  - 0: 关闭
  - 1: 红色
  - 2: 绿色
  - 3: 蓝色
- 控制灯源
  - 使能（默认状态为使能）
  `eda.enableLightSection(1);`
  取值范围为1~4，分别对应不同的分区
  - 禁用
  `eda.disableLightSection(1);`
  取值范围为1~4，分别对应不同的分区

灯源的使能/禁用不是打开/关闭灯源，灯源与摄像头是联动的，只有当灯源已使能且摄像头打开的条件下灯源才会亮。

2.2.7 源文件

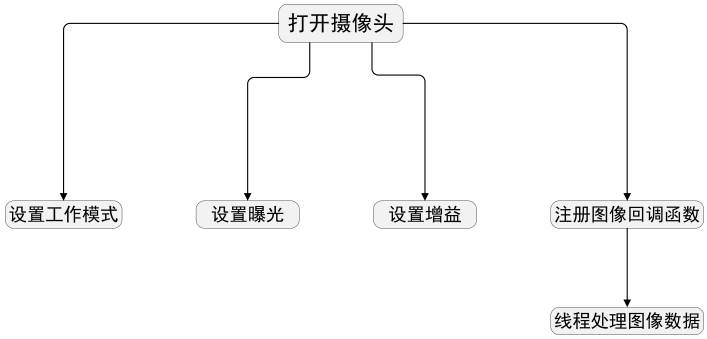
IO控制（Python3）

```py
from libedaio import EdaIo,registerInput,registerTrigger,registerTune

def func_trigger(v):
    print("[Debug] Trigger: trigger button!", v)

...
eda = EdaIo.singleton(); # 获取IO控制实例
registerTrigger(func_trigger); # 注册Trigger 按键回调
# registerInput(func_trigger); # 注册Input输入回调
# registerTune(func_trigger); # 注册Tune 按键回调
eda.setup(); # 初始化
...
eda.openLaser(); # 打开激光
# eda.closeLaser(); # 关闭激光
eda.setScanStat(True); #  设置状态指示灯
eda.openAlarm(); # 打开警告指示灯
# eda.closeAlarm(); # 关闭警告指示灯
eda.setDo1High(True); # 设置第一路输出
eda.setDo2High(False); # 设置第二路输出
eda.setRgbLight(1); # 设置侧灯，0: 关闭; 1: 红色; 2: 绿色; 3: 蓝色
```

2.3 Camera Sensor控制(C++)

本节介绍打开/关闭camera、设置相机工作模式、设置相机曝光时间和设置相机增益等的具体操作。

2.3.1 流程图



2.3.2 操作步骤

在操作Camera之前，需要先获取IO实例并初始化（具体操作参见2.1.2 获取实例并初始化），再进行如下操作。

1. 获取实例

```
eda::Camera *t_camera = eda::load_default();
```

2. 查询Sensor类型

```
t_camera->name()
```

- `eda::CameraName::AR0234`
- `eda::CameraName::OV2311`

AR0234表示230万像素的Camera

OV2311表示200万像素的Camera

3. 打开摄像头并设置工作模式、摄像头区域宽和摄像头区域高。

```
t_camera->open(mod, width, height);
```

- mod表示工作模式，取值包含0、1和5
  - 0表示连续模式（一直打开摄像头），AR0234和OV2311均支持
  - 1表示硬件触发模式，AR0234和OV2311均支持，通过Trigger引脚接5V触发
  - 5表示软件触发模式，仅OV2311支持，通过手动调节来触发
    `int call_trigger();`
- width表示摄像头区域宽度
- height表示摄像头区域高度

4. 设置增益

`t_camera->set_gain(gain);`

- OV2311：gain取值范围为0~30
- AR0234：gain取值范围为0~64

5. 设置曝光

`t_camera->set_exposure(exposure);`

- OV2311：exposure取值范围为0~65523，单位为us
- AR0234：exposure取值范围为1~1500， exposure的值乘以6.8后的值的单位是us)

6. 通过回调方式获取摄像头数据

`t_camera->callback_image_ready(image_callback);`

回调函数中，推荐只获取数据不处理逻辑。

7. 关闭摄像头

`eda::EdaIo::close_io();`

2.3.3 源文件

Sensor控制

```c++
typedef int(*img_Callback)(char *img_buff, int img_len);

enum CameraName{
    AR0234, OV2311
};
class Camera
{
public:
    /**
     * @brief 初始化摄像头
     *
     * @param mode 0 - 连续工作模式; 1 - 硬件触发模式； 5 - 软件触发模式
     * @param width
     * @param height
     * @return int
     */
    int open(int mode, int width, int height) = 0;
    /**
     * @brief 关闭摄像头
     *
     * @return int
     */
    int close() = 0;

    /**
```

```
     * @brief 设置曝光时间
     *
     * @param exp_value
     * @return int
     */
    int set_exposure(int exp_value) = 0;
    /**
     * @brief 获取曝光时间
     *
     * @param exp_value
     * @return int
     */
    int get_exposure(int *exp_value) = 0;
    /**
     * @brief 设置增益
     *
     * @param gain_value
     * @return int
     */
    int set_gain(int gain_value) = 0;
    /**
     * @brief 获取增益
     *
     * @param gain_value
     * @return int
     */
    int get_gain(int *gain_value) = 0;
     /**
     * @brief 注册回调函数，获取图像数据
     *
     * @param callback
     * @return int
     */
    int callback_image_ready(img_Callback callback)=0;

    CameraName name() = 0;
};
```

获取AR0234实例

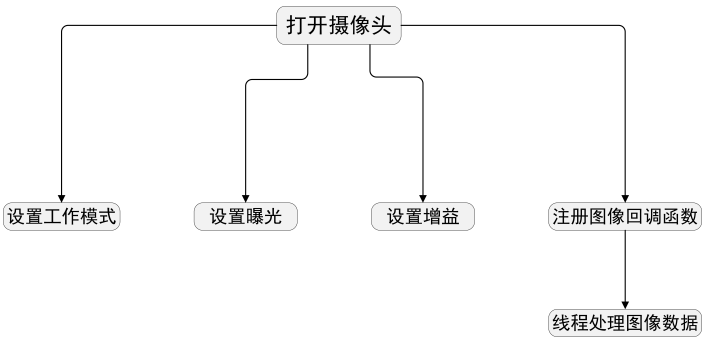
```c++
#include "CameraManger.h"
#include "camera_0234.h"

void test()
    eda::Camera *t_camera = eda::create_ar0234();
    if(t_camera){
        eda::Camera_0234 *t_camera_1 = static_cast<eda::Camera_0234*>(t_camera);
    }
```

```
    ...
}
```

2.4 Camera Sensor控制(Python)

本节介绍打开/关闭camera、设置相机工作模式、设置相机曝光时间和设置相机增益等的具体操作。

2.4.1 流程图



2.4.2 操作步骤

在操作Camera之前，需要先导入IO模块再获取IO实例并初始化（具体操作参见2.2.2 导入模块和2.2.3 获取实例并初始化），再进行如下操作。

1. 导入模块

```
from libedacamera import EdaCamera
```

2. 获取实例

```
eda = EdaCamera.load_default();
```

3. 查询Sensor类型

```
eda.get_name();
```

- `return “AR0234”`
- `return “OV2311”`

AR0234表示230万像素的Camera

OV2311表示200万像素的Camera

4. 打开摄像头并设置工作模式、摄像头区域宽和摄像头区域高。

```
ret = eda.open(t_mode,t_width, t_height);
```

- t_mode表示工作模式，取值包含0、1和5
  - 0表示连续模式（一直打开摄像头），AR0234和OV2311均支持
  - 1表示硬件触发模式，AR0234和OV2311均支持，通过Trigger引脚接5V触发

    - 5表示软件触发模式，仅OV2311支持，通过手动调节来触发
      `eda.call_trigger();`
- t_width表示摄像头区域宽度
- t_height表示摄像头区域高度

5. 设置增益

```
eda.set_gain(int(t_gain));
```

- OV2311：t_gain取值范围为0~30
- AR0234：t_gain取值范围为0~64

6. 设置曝光

```
eda.set_exposure(int(t_exposure));
```

- OV2311：t_exposure取值范围为0~65523，单位为us
- AR0234：t_exposure取值范围为1~1500， t_exposure的值乘以6.8后的值的单位是us）

7. 通过回调方式获取摄像头数据

```
eda.callback_image_ready(func_image_data);
```

回调函数中，推荐只获取数据不处理逻辑。

8. 关闭摄像头

```
eda.close();
```

3 示例

本章介绍具体的操作示例，包含编写代码、编译代码和运行代码。

3.1 编写代码

下文以实现“打开激光等待2s后关闭激光”的功能为例，使用C++语言进行编写代码。

编写的内容如下：

```c++
#include "eda/eda-io.h"
#include <unistd.h>
#include "stdlib.h"

int main(int argc, char *argv[]){
    eda::EdaIo *em = eda::EdaIo::getInstance();
    em->setup();
    //打开激光
    em->openLaser();
    sleep(2);
    // 关闭激光
```

```
    em->closeLaser();
    eda::EdaIo::close_io();
    return 0;
}
```

编写完成后，保存为test123.cpp文件。

> 提示
>
> 文件名自定义即可。

3.2 编译和运行代码

C++代码编写完成后需要登录Camera设备，在Raspberry Pi系统上进行编译和运行。

前提条件：

- 已完成Camera的硬件部分的连线，具体的操作请参见《ED-AIC2020用户手册》。
- 已将Camera上电并正常接入网络。
- 已获取Camera IP地址，并成功登录Camera系统。

操作步骤：

在Camera系统上创建一个文件夹，将章节3.1 编写代码中编写的代码文件上传至文件夹中。

执行`ls`命令，查看文件夹中的文件，确保代码文件已上传成功。

1. 执行如下命令，对编写的代码进行编译。
2. `g++ -l eda_io -o test-io test123.cpp`

`test123.cpp`:表示章节3.1 编写代码中编写的代码文件。

`test-io`：表示编译后生成的文件名（可自定义）

执行`ls`命令，查看编译后生成的新文件，如下图`test-io`。

```
pi@raspberrypi:~/tmp $ ls
test123.cpp  test-io
```

执行如下命令，运行编译后的代码。

`sudo ./test-io`

`test-io`:表示编译后生成的文件名。

> 提示
>
> 运行成功后，可以看到激光点亮且等待2s后熄灭。