# ED-AIC2000 系列

# 基于 Raspberry Pi CM4 的工业智能相机

# SDK 开发指南

上海晶珩电子科技有限公司

2024年5月

# 联系我们

非常感谢您购买和使用我们公司的产品，我们将竭诚为您提供服务。

我们是 Raspberry Pi 的全球设计合作伙伴之一，致力于提供基于 Raspberry Pi 技术平台的物联网、工业控制、自动化、绿色能源和人工智能的硬件解决方案。

您可以通过以下方式联系我们：

上海晶珩电子科技有限公司

EDA Technology Co.,LTD

地址：上海市嘉定区嘉罗公路 1661 号 29 栋

邮箱：sales@edatec.cn

手机：+86-18621560183

网站：https://www.edatec.cn

**技术支持：**

邮箱： support@edatec.cn

手机：+86-18627838895

微信：zzw_1998-

# 版权声明

# 免责声明

上海晶珩电子科技有限公司不保证本手册中的信息是最新的、正确的、完整的或高质量的。上海晶珩电子科技有限公司也不对这些信息的进一步使用作出保证。如果由于使用或不使用本手册中的信息，或由于使用错误或不完整的信息而造成的物质或非物质相关损失，只要没有证明是上海晶珩电子科技有限公司的故意或过失，就可以免除对上海晶珩电子科技有限公司的责任索赔。上海晶珩电子科技有限公司明确保留对本手册的内容或部分内容进行修改或补充的权利，无需特别通知。

# 前 言

## 读者范围

本手册适用以下读者对象：

- 软件工程师
- 系统工程师

## 相关约定

### 术语约定

| 术语 | 含义 |
| --- | --- |
| CM4 | Raspberry Pi CM4 模块，全称为 Raspberry Pi Compute Module 4 |

### 符号约定

| 符号 | 说明 |
| --- | --- |
|  | 提示符号，提示重要的特点或操作。 |
|  | 注意符号，可能会对人身造成伤害，或给系统造成损害，或造成信号中断/丢失。 |
|  | 可能会对人身造成重大伤害。 |

# 安全说明

- 本产品应在符合设计规格要求的环境下使用，否则可能造成故障，因未遵守相关规定引发的功能异常或部件损坏等不在产品质量保证范围之内。
- 因违规操作产品引发的人身安全事故、财产损失等，我司将不承担任何法律责任。
- 请勿私自修改设备，修改设备可能会使设备故障。
- 安装设备时，需要固定好设备，防止跌落。
- 如果设备带有天线，正常使用时，请与设备至少保持20cm的距离。
- 请勿使用液体清洁设备，应远离液体和易燃物品。
- 本产品仅支持在室内环境使用。

# 目　录

# 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。

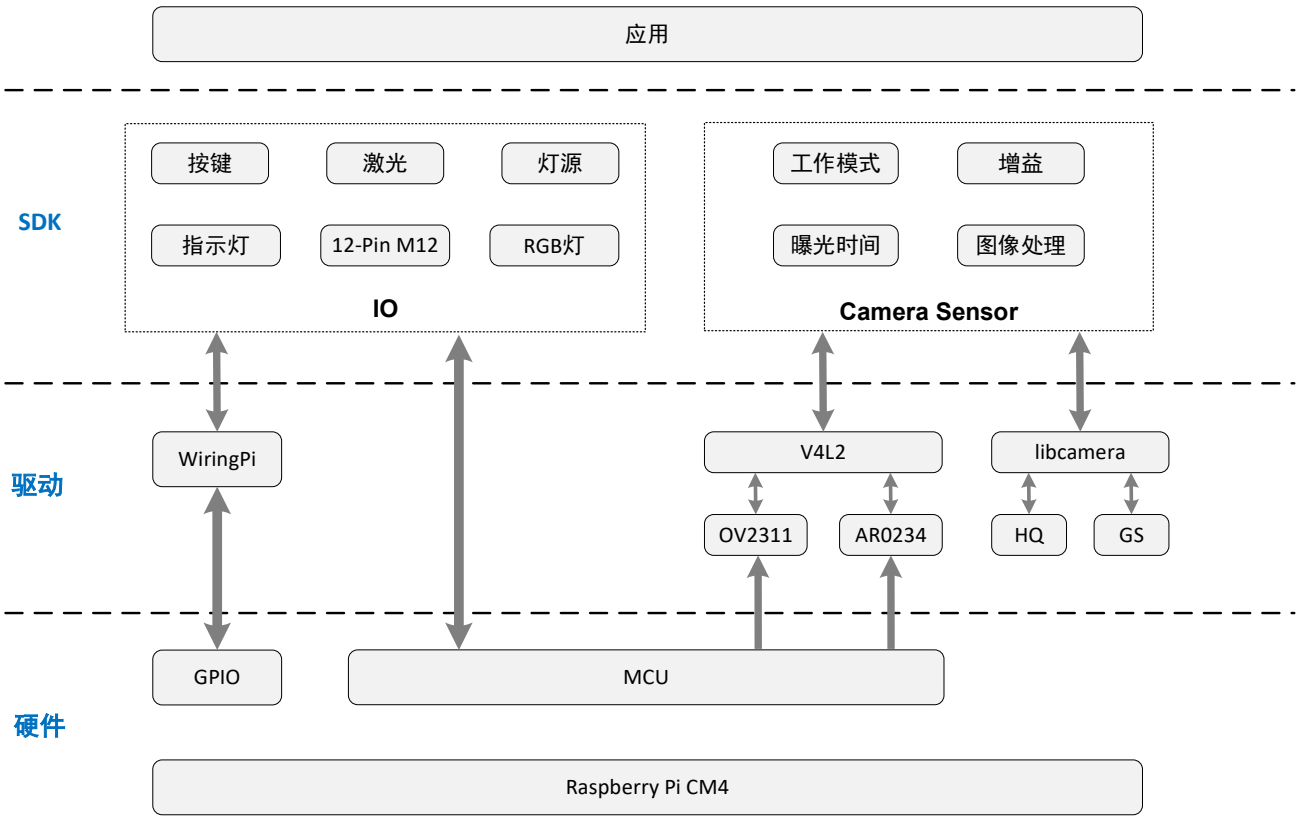
- ✓ SDK 简介
- ✓ SDK 组成

## 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 Sensor 控制 | 头文件 | camera.h | /usr/include/eda/ |
| | | 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

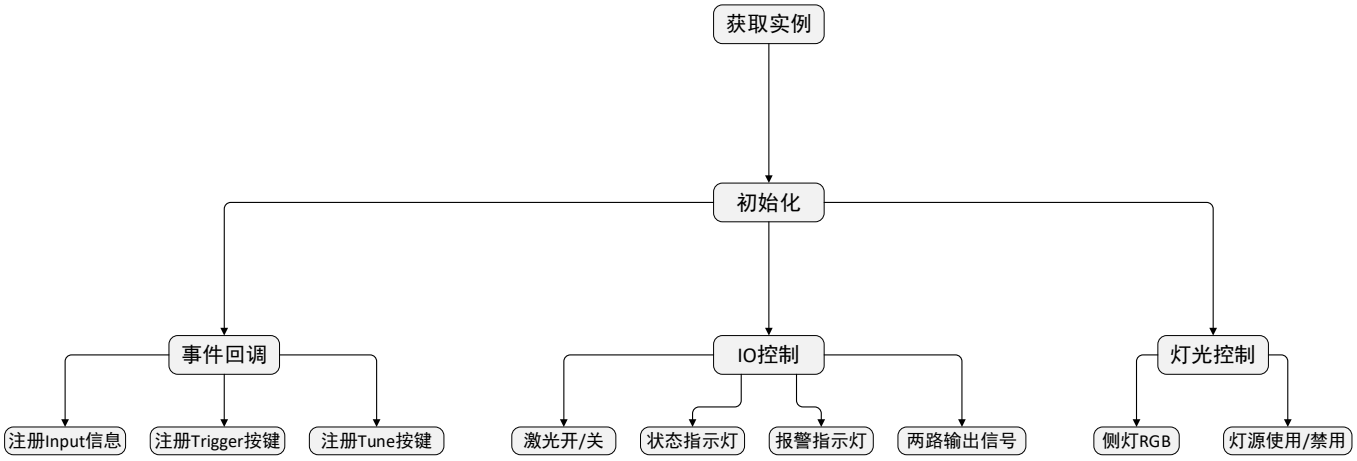
本章介绍各项功能对应代码的编写方法，帮助用户编写上层应用所需要的代码。

- ✓ IO 控制(C++)
- ✓ IO 控制(python)
- ✓ Camera Sensor 控制(C++)
- ✓ Camera Sensor 控制(python)

# 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**

```
#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）

```
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();
```

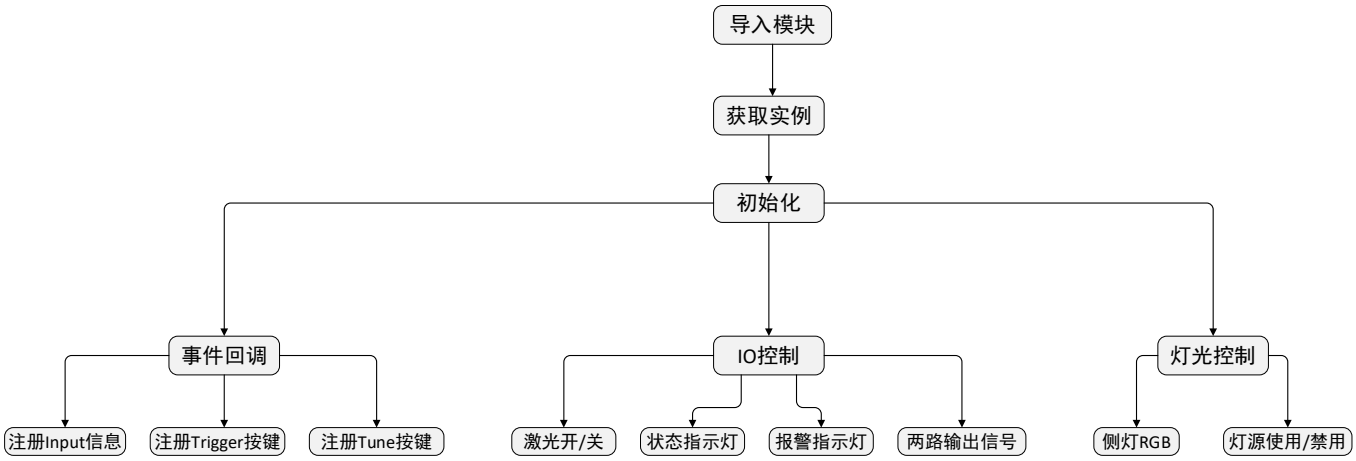
```
/**
 * @brief 关闭 alarm 指示灯
 *
 */
void closeAlarm();
/**
 * @brief
 *
 * @param section 1~4
 * @return int
 */
int enableLightSection(int section);
/**
 * @brief
 *
 * @param section 1~4
 * @return int
 */
int disableLightSection(int section);
/**
 * @brief 设置 output1 输出 [高/低]
 *
 * @param high
 */
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 Edalo,registerInput,registerTrigger,registerTune
```

### 2.2.3 获取实例并初始化

导入模块后再获取 IO 实例并对实例进行初始化，操作步骤如下。

1. 获取 IO 实例。

   ```
   edalo = Edalo.singleton();
   ```

2. 对实例进行初始化。

   ```
   edalo.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**

```
#!/usr/bin/python3
from libedaio import Edalo,registerInput
def func_input(v):
    print("[Debug] Trigger: input!", v)
def main() -> int:
    eda = Edalo.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

  edaIo.openLaser();

  edaIo.closeLaser();

- 控制状态指示灯

  edaIo.setScanStat(true)

  edaIo.setScanStat(false)

- 控制警报指示灯

  edaIo.openAlarm()

  edaIo.openAlarm()

- 控制 2 路输出信号

  edaIo.setDo1High(false);

  edaIo.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

  edaIo.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）

  edaIo.enableLightSection(1);

  取值范围为 1~4，分别对应不同的分区。

- 禁用

  edaIo.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）

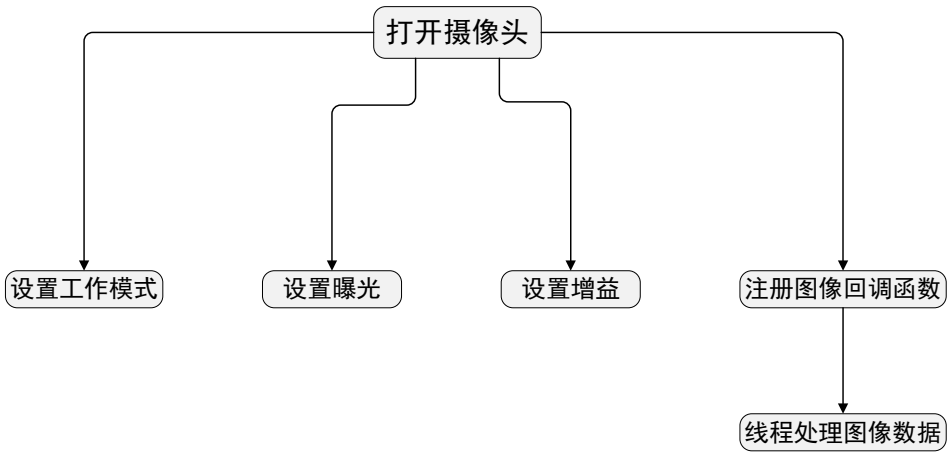
```
from libedaio import EdaIo,registerInput,registerTrigger,registerTune

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

```
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;

};
```

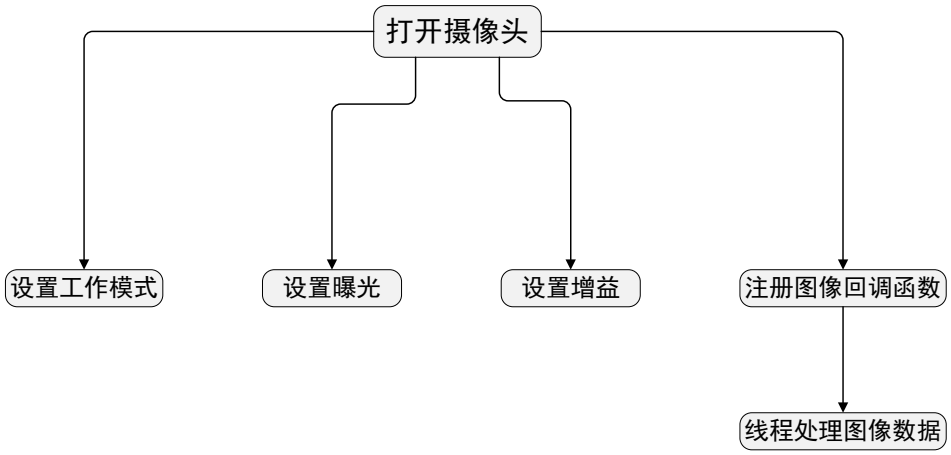
### 获取 AR0234 实例

```
#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 Camera 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， 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。

- ✓ 编写代码
- ✓ 运行和编译代码

## 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：

```
#include "eda/eda-io.h"

#include <unistd.h>
#include "stdlib.h"

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

1. 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。
2. 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。
3. 执行如下命令，对编写的代码进行编译。

   **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）
4. 执行如下命令，查看编译后生成的新文件，如下图 test-io。

   **ls**

   ```
   pi@raspberrypi:~/tmp $ ls
   test123.cpp  test-io
   ```
5. 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。