



ED-AIC2000

User Manual

by EDA Technology Co., Ltd

built: 2024-11-20

1 Hardware Manual

This chapter introduces the product overview, packing list, appearance, indicator and interface.

1.1 Overview

ED-AIC2000 is an industrial smart camera based on Raspberry Pi CM4 with a sampling rate of up to 70 FPS. According to different application scenarios and user needs, different specifications of RAM, eMMC, pixels and lenses can be selected.

- RAM can choose 1GB, 2GB, 4GB and 8GB
- eMMC can choose 8GB, 16GB and 32GB
- Optional pixel specifications include 2 million pixels, 2.3 million pixels and 12 million pixels
- Optional lens specifications include M12 fixed focus lens and M12 fixed focus lens with liquid module zoom

ED-AIC2000 integrated light source module design, the light source supports 3 optional colors; using bright field and dark field modes, it can achieve the best lighting effect on normal, etched, high gloss or textured surfaces.

The ED-AIC2000 series includes three models: ED-AIC2000-020, ED-AIC2000-023 and ED-AIC2000-120. It provides power interface, IO interface, RS232 serial port and Gigabit Ethernet interface, adopts M12 aviation connector, supports IP65 waterproof grade, supports network access via Ethernet, and is mainly used in machine vision and artificial intelligence fields.



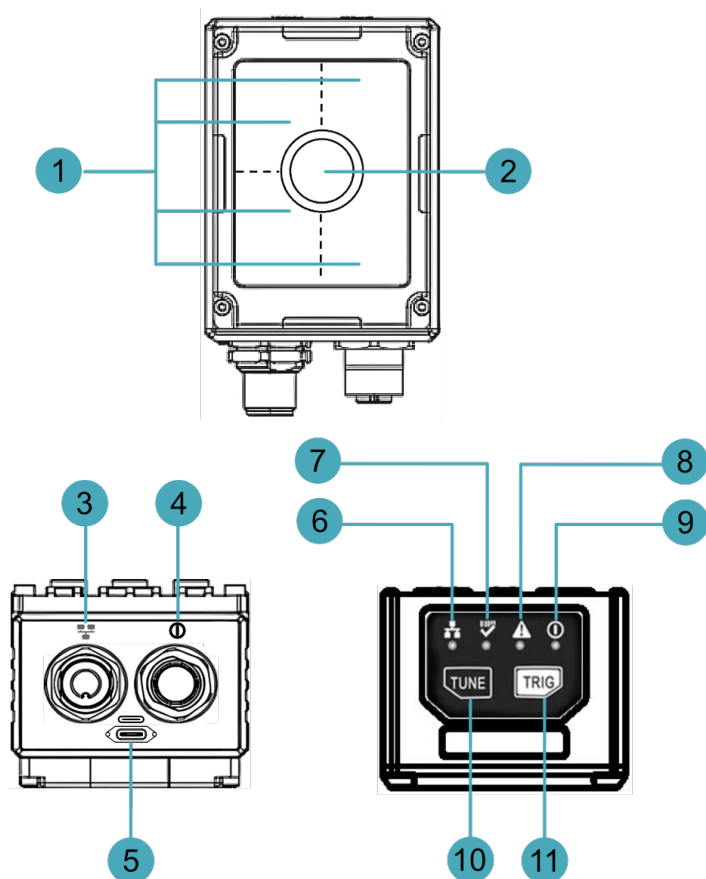
1.2 Packing List

1 x ED-AIC2000 Unit

1.3 Appearance

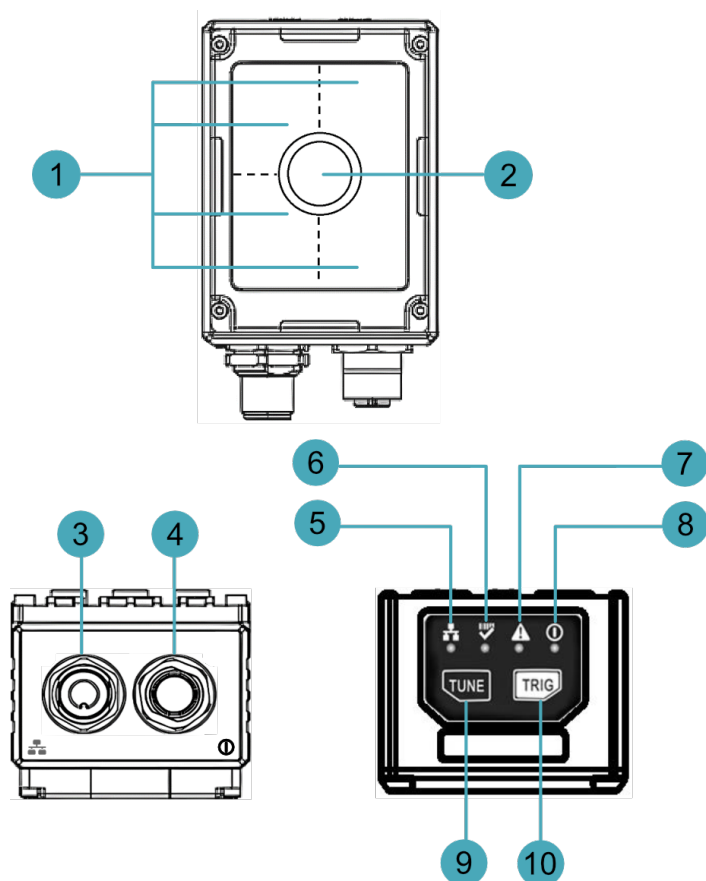
Introduce the functions and definitions of product interfaces, buttons and indicators.

ED-AIC2000-020



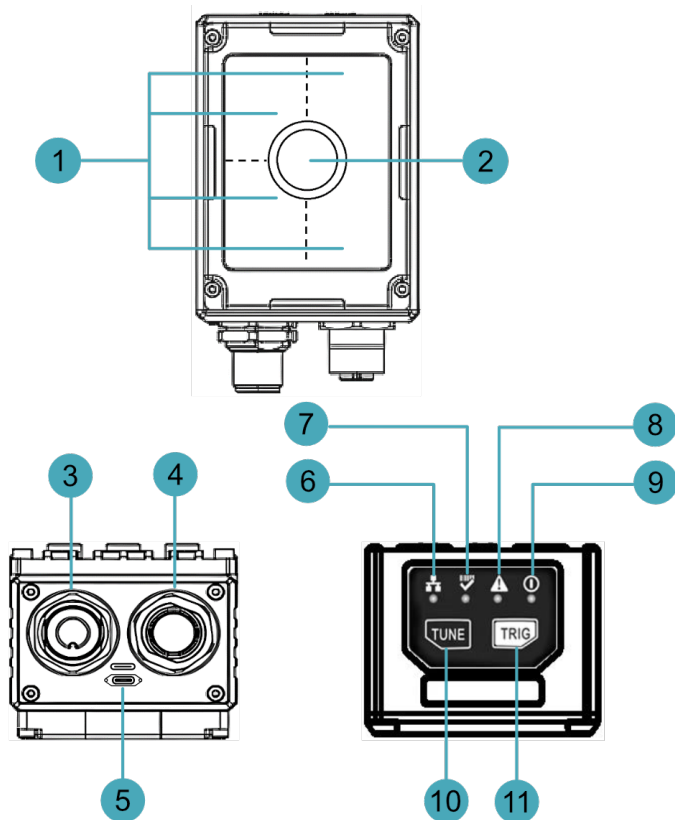
NO.	Function Definition
1	4 x light sources, using for supplementary lighting when the device is working.
2	1 x lens, using to take photos.
3	1 x communication interface, Gigabit Ethernet interface, using M12 8-Pin A-code aviation connector for access to Ethernet.
4	1 x power interface, including power interface, IO interface and RS232 serial port, using M12 12-Pin aviation connector.
5	1 x USB-c port, using to flash to eMMC.
6	1 x Network connection indicator, which is used to check the status of the network connection.
7	1 x working status indicator, which is used to check the working status of the device.
8	1 x System fault indicator, which is used to check if a system fault occurs.
9	1 x Power indicator, which is used to check the power-on status of the device.
10	1 x adjustment button, one-touch autofocus button or user-defined button.
11	1 x Trigger button, using to camera triggering or user-defined one-touch button.

ED-AIC2000-023



NO.	Function Definition
1	4 x light sources, using for supplementary lighting when the device is working.
2	1 x lens, using to take photos.
3	1 x communication interface, Gigabit Ethernet interface, using M12 8-Pin A-code aviation connector for access to Ethernet.
4	1 x power interface, including power interface, IO interface and RS232 serial port, using M12 12-Pin aviation connector.
5	1 x Network connection indicator, which is used to check the status of the network connection.
6	1 x working status indicator, which is used to check the working status of the device.
7	1 x System fault indicator, which is used to check if a system fault occurs.
8	1 x Power indicator, which is used to check the power-on status of the device.
9	1 x adjustment button, one-touch autofocus button or user-defined button.
10	1 x Trigger button, using to camera triggering or user-defined one-touch button.

ED-AIC2000-120



NO.	Function Definition
1	4 x light sources, using for supplementary lighting when the device is working.
2	1 x lens, using to take photos.
3	1 x communication interface, Gigabit Ethernet interface, using M12 8-Pin A-code aviation connector for access to Ethernet.
4	1 x power interface, including power interface, IO interface and RS232 serial port, using M12 12-Pin aviation connector.
5	1 x USB-c port, using to flash to eMMC.
6	1 x Network connection indicator, which is used to check the status of the network connection.
7	1 x working status indicator, which is used to check the working status of the device.
8	1 x System fault indicator, which is used to check if a system fault occurs.
9	1 x Power indicator, which is used to check the power-on status of the device.
10	1 x adjustment button, one-touch autofocus button or user-defined button.
11	1 x Trigger button, using to camera triggering or user-defined one-touch button.

1.4 Button

The ED-AIC2000 series devices include 2 buttons, an adjustment button and a trigger button.

- The adjustment button has "TUNE" printed on the case. Pressing the button can realize one-touch automatic focus, and it supports that users to customize its functions.
- The trigger button has "TRIG" printed on the case. Pressing the button can trigger the camera, and it support that users to customize its functions.

Pin Definition

The button pins are defined as follows:

button	CM4 pin
adjustment button	GPIO20
trigger button	GPIO12

1.5 Indicator

This section describes the various states and meanings of the indicators on the ED-AIC2000 series devices.

Indicator	Status	Description
Network connection indicator	On	The Ethernet has been connected normally
	Off	No Ethernet connection
Working status indicator	blink	The system is working normally
	Off	System working status is abnormal
System fault indicator	blink	System failure
	Off	The system has not failed
Power indicator	On	The device is powered on
	Off	The device is powered off

Pin Definition

indicator	CM4 pin
Power indicator	N/A
System fault indicator	GPIO21
Working status indicator	GPIO7 (abnormal) GPIO16(normal)

indicator	CM4 pin
Network connection indicator	N/A

1.6 Interface

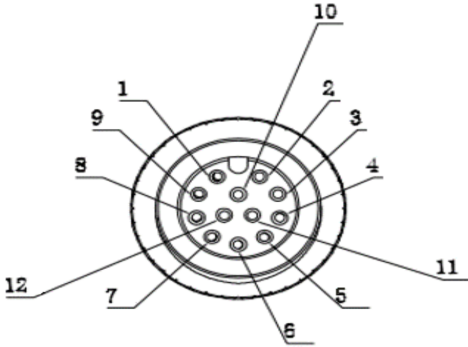
This section describes the definition and function of each interface in the product.

1.6.1 Power interface

The ED-AIC2000 series devices include 1 power interface, 1 serial port and 6 general GPIOs. The power interface uses an M12 12-Pin aviation connector, supports 10V~30V input, and the signal is defined as DC-/DC+.

Pin Definition

The pin definitions are as follows:

	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 and DO interfaces

ED-AIC2000 series devices include one universal input terminal DI1, two universal output terminals DO1 and DO2, support 10V~30V DC input and output, and have photoelectric isolation and overcurrent protection.

Pin Definition

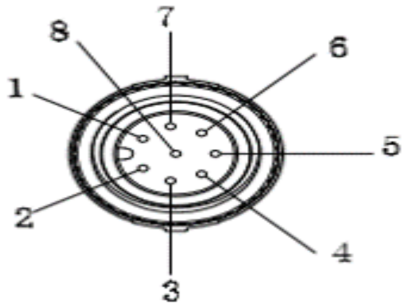
The DI and DO pins are defined as follows:

1.6.3 Communication interface

ED-AIC2000 series devices include 1 communication interface with M12 8-Pin aviation connector.

Pin Definition

The pin definitions are as follows:

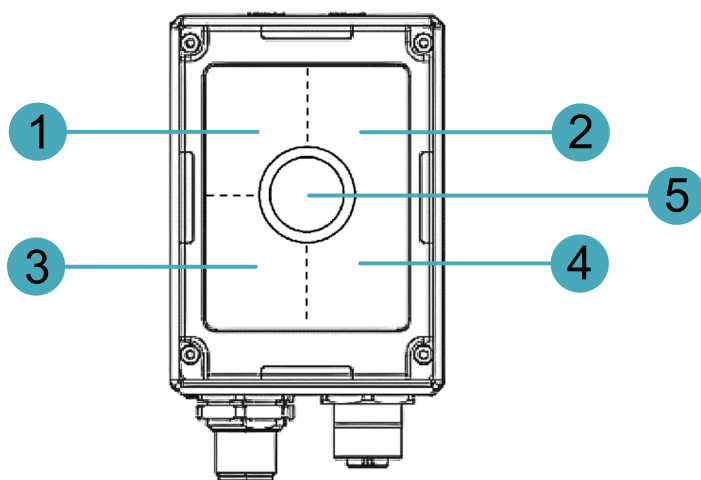
	Pin ID	Pin Name
	1	TRD0+
2	TRD0-	
3	TRD1+	
4	TRD2+	
5	TRD2-	
6	TRD1-	
7	TRD3+	
8	TRD3-	

1.6.4 USB-C interface

Both ED-AIC2000-020 and ED-AIC2000-120 include a USB type-c interface, which supports flashing to eMMC by connecting to a PC.

1.7 Light source and lens

The ED-AIC2000 series equipment consists of four light sources and a lens.



NO.	Description
1	Light source part 1, supports individual enabling and disabling.

NO.	Description
2	Light source part 2, supports individual enabling and disabling.
3	Light source part 3, supports individual enabling and disabling.
4	Light source part 4, supports individual enabling and disabling.
5	Lenses, M12 fixed focal length lens and M12 fixed focal length lens with liquid module zoom.

Light source control command

The control commands are as follows:

Signal	CM4 pin(GPIO 8/9 , /dev/ttyAMA2)
Control light source part 1	RS232 commands <ul style="list-style-type: none"> • Enable: 1-1 • Disable: 1-0
Control light source part 2	RS232 commands <ul style="list-style-type: none"> • Enable: 2-1 • Disable: 2-0
Control light source part 3	RS232 commands <ul style="list-style-type: none"> • Enable: 3-1 • Disable: 3-0
Control light source part 4	RS232 commands <ul style="list-style-type: none"> • Enable: 4-1 • Disable: 4-0

2 Installing Device

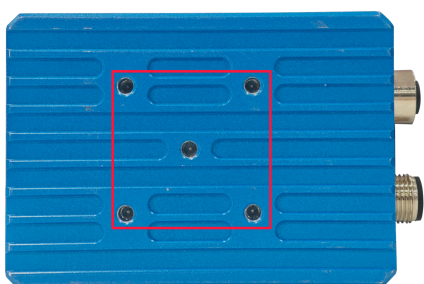
This chapter introduces how to install the device.

Preparation :

- Installation brackets and installation screws have been prepared (M4*8 with spacers, M6*10 with spacers)
- M4 and M6 allen head screwdrivers have been prepared

Steps :

1. Determine the location of the installation hole on the camera, as shown in the red frame in the figure below. Generally, it can be fixed with a central screw and one peripheral screw.



2. Place the installation bracket above the camera installation hole, so that the bracket (the side with M4 screw hole) is aligned with the central screw hole of the camera, and use a screwdriver to fix the bracket on the camera with M4 screw with spacer, as shown in the following figure.



3. Rotate the bracket, adjust the installation direction, and select top installation or side installation as needed. The effect is as shown below:



4. Use M6 screws to fix the bracket and other devices. It is recommended to use a center screw and one peripheral screw for fixing.

TIP

Adjust the appropriate installation position and angle according to different project requirements on site.

3 Booting the Device

This chapter introduces how to connect cables and boot the device.

3.1 Connecting Cables

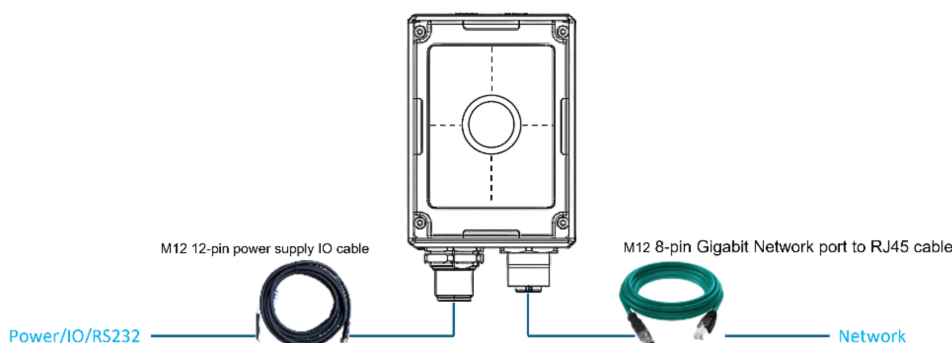
This section describes how to connect cables.

Preparation:

- The 8-core Gigabit Ethernet port to RJ45 cable and 12-core power IO cable to be connected have been obtained.
- A DC 12V 2A power adapter and auxiliary wiring connector have been prepared.
- Prepare the connectors and wiring tools required for the 12-core power IO cable as needed.

Schematic diagram of connecting cables:

For the pin definitions and wiring methods of each interface, see [1.6 Interface](#).



3.2 Booting The System For The First Time

The ED-AIC2000 series equipment has no power switch. After the power is connected, the system will start.

- The power indicator is always on, indicating that the device is powered normally.
- The working status indicator light is always on, indicating that the system starts normally.

After the system starts, the default user name and password are used for login. Because the camera cannot be connected to the monitor, it is necessary to log in to the system remotely through the PC.

The factory image of ED-AIC2000 series devices enables VNC by default, and sets the device IP to a static IP: 192.168.1.10. Users can remotely connect to the device through VNC and enter the device's desktop system. For specific operations, refer to [Connect to the device desktop via vnc](#).

TIP

Default username: pi; default password: raspberry.

4 Configuring System

This chapter introduces how to configure system.

4.1 Compile Camera Demo

How to compile the startup test Camera example.

1. Go to the folder where the test file is located.

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

2. Compile the test code

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

```
sudo ./test_camera
```

4.2 Camera Interface

- The "camera.h" file provides the control interface for operating the AI Camera Sensor. The file is stored in "/usr/include/eda/camera.h".
- The "eda-io.h" file provides the control interface for operating the AI Camera IO. The storage path of this file is "/usr/include/eda/eda-io.h".

4.3 Library Files

- The storage path of the "libeda_camera.so" library file is "/usr/lib/libeda_camera.so".
- The storage path of the "libeda_io.so" library file is "/usr/lib/libeda_io.so".

4.4 Camera API

The Camera API provides control functions for the AI Camera Sensor, which support turning the sensor on and off, setting the sensor operating mode, setting the exposure time, and setting the gain. The following describes each function according to different Camera scenarios.

Control Function

The control functions of the AI Camera Sensor provided by the Camera API are shown in the following table, including obtaining image data, opening the camera, closing the camera, setting the working mode, setting the exposure time, and setting the gain.

- AR0234 means 2.3 million pixel camera
- OV2311 means 2 million pixel Camera

Function	Definition	Parameter Value
virtual int callback_image_ready(img_Callback callback)=0	Get image data	-
virtual int open(int mode, int width, int height) = 0	Open and configure the camera	<p>The working mode value includes 0, 1, 5</p> <ul style="list-style-type: none"> • 0 means continuous mode (the camera is always on), which is supported by both AR0234 and OV2311. • 1 indicates hardware trigger mode, which is supported by AR0234 and OV2311, and is triggered by 5V via the Trigger pin. • 5 indicates software trigger mode, which is only supported by OV2311 and is triggered by manual adjustment.
virtual int close() = 0	Close camera	-
virtual int set_exposure(int exp_value) = 0	Setting the exposure time	<ul style="list-style-type: none"> • OV2311: t_exposure value range is 0~65523, unit is us • AR0234: t_exposure ranges from 1 to 1500. The value of exposure multiplied by 6.8 is expressed in us.
virtual int get_exposure(int *exp_value) = 0	Get exposure time	-
virtual int set_gain(int gain_value) = 0	Setting the Gain	<ul style="list-style-type: none"> • OV2311: t_gain value range is 0~30 • AR0234: t_gain value range is 0~64
virtual int get_gain(int *gain_value) = 0	Get Gain	-

Get AR0234 instance

```

#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);
    }
    ...
}

```

C++

4.5 IO API

The IO API provides control functions for the AI Camera IO, supporting control of indicator lights, lasers, side lights, and outputs.

4.5.1 C language environment

The control function of AI Camera IO in C language environment is as follows:

Function	Definition
<code>eda::Edalo *em = eda::Edalo::getInstance()</code>	Get IO control instance
<code>void setup()</code>	Initialize IO settings
<code>void openLaser()</code>	Turn on the laser
<code>void closeLaser()</code>	Turn off the laser
<code>void setScanStat(bool good)</code>	Set status indicator
<code>void openAlarm()</code>	Turn on the warning light
<code>void closeAlarm()</code>	Turn off the warning light
<code>void setDo1High(bool high)</code>	Set output1 output
<code>void setDo2High(bool high)</code>	Set output2 output
<code>void registerInput(IoTrigger callback)</code>	Register input trigger callback function
<code>void registerTrigger(IoTrigger callback)</code>	Register trigger button callback function
<code>void registerTune(IoTrigger callback)</code>	Register Tune button callback function
<code>void setRgbLight(uint8_t light)</code>	Set up RGB lighting

4.5.2 Python3 language environment

The control function of AI Camera IO in Python3 language environment is as follows:

Function	Definition
<code>eda = Edalo.singleton()</code>	Get IO control instance
<code>eda.setup()</code>	initialization
<code>eda.openLaser()</code>	Turn on the laser
<code>eda.closeLaser()</code>	Turn off the laser
<code>eda.setScanStat(True)</code>	Set status indicator
<code>eda.openAlarm()</code>	Turn on the warning light
<code>eda.closeAlarm()</code>	Turn off the warning light
<code>eda.setDo1High(True)</code>	Set output1 output
<code>eda.setDo2High(False)</code>	Set output2 output
<code>registerInput(func_trigger)</code>	Register input trigger callback function

Function	Definition
registerTrigger(func_trigger)	>Register trigger button callback function
registerTune(func_trigger)	Register Tune button callback function
eda.setRgbLight(1)	Set up RGB lighting

4.5.3 Operating Instructions

1. initialization

- Before operating IO, you need to obtain an IO instance `eda::EdaIo *em = eda::EdaIo::getInstance();`
- Initialize the instance `em->setup();`

2. IO control supports event registration callback functions

- input event `em->registerInput(trigger_input);`
- Trigger button `em->registerTrigger(trigger_trigger);`
- Tune button `em->registerTune(trigger_tune);`

3. Control IO (must be initialized first)

- Controlling lasers `em->openLaser()` and `em->closeLaser();`
- Control status indicator `em->setScanStat(true)` and `em->setScanStat(false);`
- Control alarm indicator light `em->openAlarm()` and `em->openAlarm();`
- Control two outputs `em->setDo1High(false)` and `em->setDo2High(false);`

4. Control lights (must be initialized first)

- Control side light color `em->setRgbLight(1);`
 - 0: Close
 - 1: Red
 - 2: Green
 - 3: Blue
- Control light source
 - Enable light source (enabled by default) `em->enableLightSection(1)` The value range is 1~4, corresponding to different partitions
 - Disable light source `em->disableLightSection(1)` , the value range is 1~4, corresponding to different partitions

5 Installing Raspberry Pi OS

The device is shipped with an operating system by default. If the OS is corrupted during use or the user needs to replace the OS, it is necessary to re-download the appropriate system image and install it.

The following section describes the specific operations of image download and eMMC flashing.

5.1 Download System image

You can download the ED-AIC2000 system image of the corresponding model according to actual needs. The download path is shown in the following table.

Product Model	Download Path
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 Flash eMMC

It is recommended to use the official Raspberry Pi flashing tool, and the download path is as follows:

- 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)

Preparation:

- The download and installation of the flashing tool to the computer has been completed.
- A USB-C to USB-A flashing cable has been prepared.
- An 8-Pin M12 male to RJ45 network cable has been prepared.
- A 12-pin M12 female to bare wire power IO cable has been prepared.
- The image file to be flashed has been obtained.

Steps:

The steps are described using Windows system as an example.

1. Connect the power cord and USB flashing cable (USB-C to USB-A).

- Connect the flashing cable: one end is connected to the USB type-C port on the device side, and the other end is connected to the USB port on the PC.

- Connect the power cable: one end is connected to the M12 12-Pin power interface on the device side (the interface in the red box in the figure below), and the other end is connected to the external power supply.



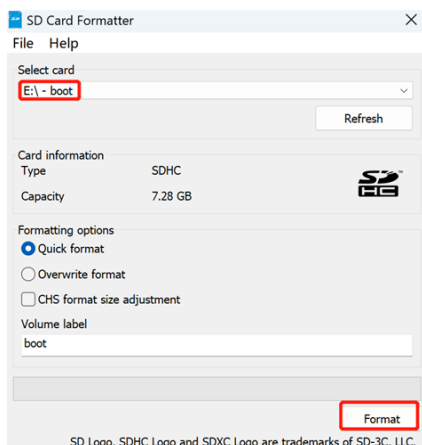
2. Disconnect the device power supply, then press and hold the TRIG button, and power on the device again. The device will automatically enter the flashing mode.
3. Open the installed rpiboot tool to automatically convert the drive to a letter.

```

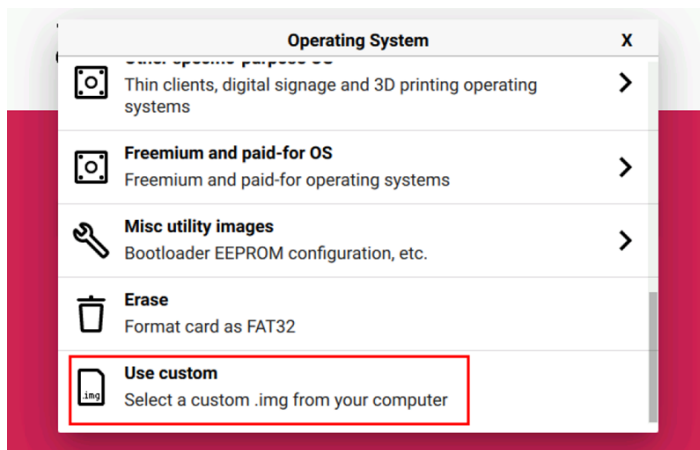
rpiboot
RPiBOOT: build-date Dec 16 2022 version 20221215-105525 lafa26c5
Waiting for BCM2835/6/7/2711...
Loading embedded: bootcode4.bin
Sending bootcode.bin
Successful read 4 bytes
Waiting for BCM2835/6/7/2711...

```

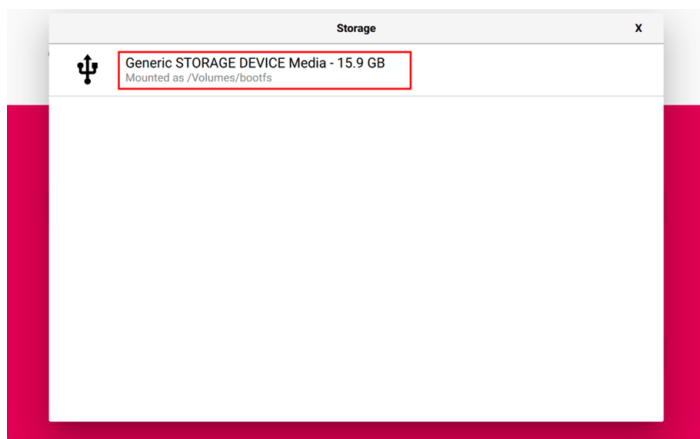
4. After the completion of the drive letter, the drive letter will pop up in the lower right corner of the computer.
5. Open SD Card Formatter, select the formatted drive letter, and click "Format" at the lower right to format.



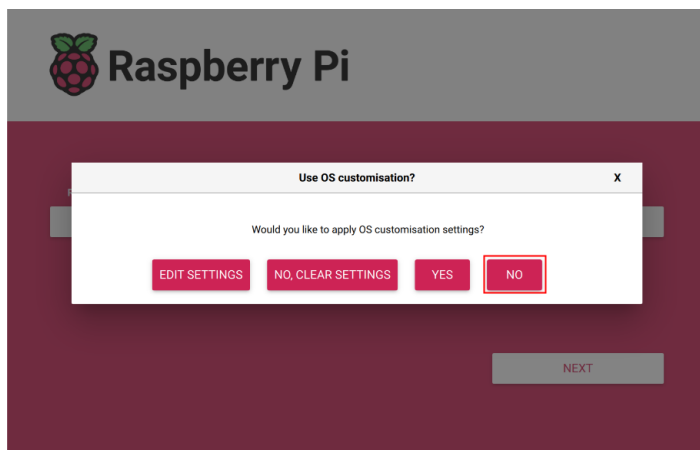
6. In the pop-up prompt box, select "Yes".
7. When the formatting is complete, click "OK" in the prompt box.
8. Close SD Card Formatter.
9. Open Raspberry Pi Imager, select "CHOOSE OS" and select "Use Custom " in the pop-up pane.



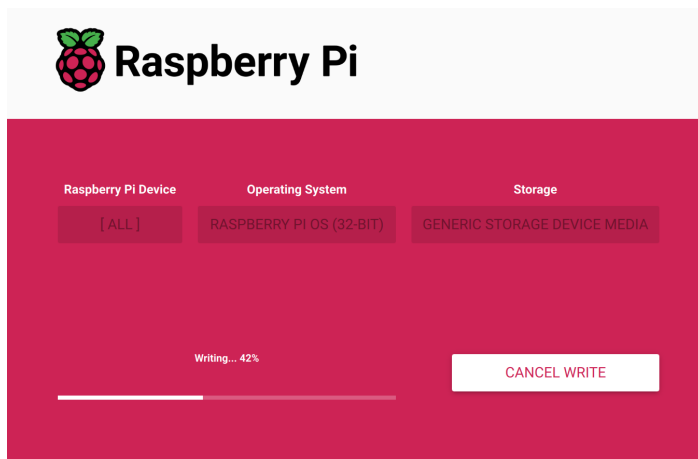
10. According to the prompt, select the OS file under the user-defined path and return to the main page.
11. Click "CHOOSE STORAGE", select the default device in the "Storage" interface, and return to the main page.



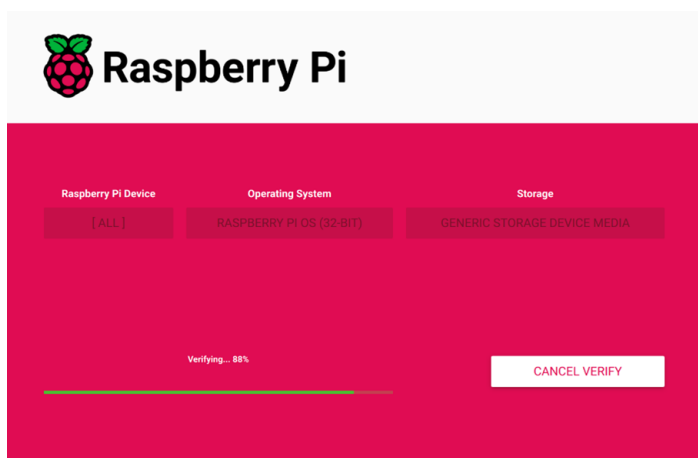
12. Click "NEXT", select "NO" in the pop-up "Use OS customization?" pane.



13. Select "YES" in the pop-up "Warning" pane to start writing the image.



14. After the OS writing is completed, the file will be verified.



15. After the verification is completed, click "CONTINUE" in the pop-up "Write Successful" box.
16. Close Raspberry Pi Imager, remove USB cable and power on the device again.

6 SDK Development Guide

1 SDK Overview

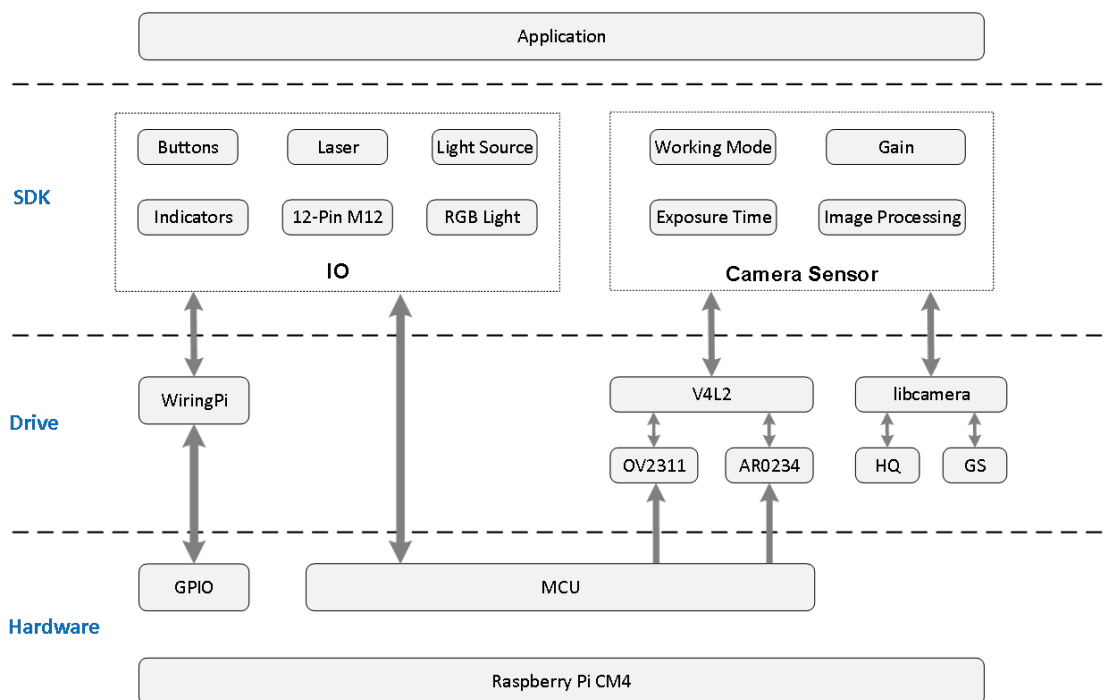
This chapter introduces the definition and composition of SDK to help users understand the SDK better.

1.1 SDK Introduction

The SDK of the ED-AIC2000 series Camera is a set of software development kit, which provides users with the interfaces required for upper-layer applications to facilitate secondary development of the camera.

The SDK functions of the ED-AIC2000 series Camera include registering Trigger/Tune button, DI definition, laser control, status indicator control, alarm indicator control, 2-channel DO control, Light and light source control, camera working mode setting, camera exposure time setting, camera gain setting and image data processing.

The location of SDK in the camera system is shown in the figure below.



1.2 SDK Composition

The SDK of camera is composed of multiple header files and library files. The details file names and installation paths are as follows.

Function	File Type	File Names	Installation Paths
IO Control	Head File	eda-io.h	/usr/include/eda/

Function	File Type	File Names	Installation Paths
	Library	libeda_io.so	/usr/lib/
Camera Sensor Control	Head File	camera.h	/usr/include/eda/
		CameraManger.h	
		camera_0234.h	
		camera_2311.h	
	Library	libeda_camera.so	/usr/lib/

During the development process, users can complete the development of upper-layer applications based on actual needs and refer to the corresponding function description below.

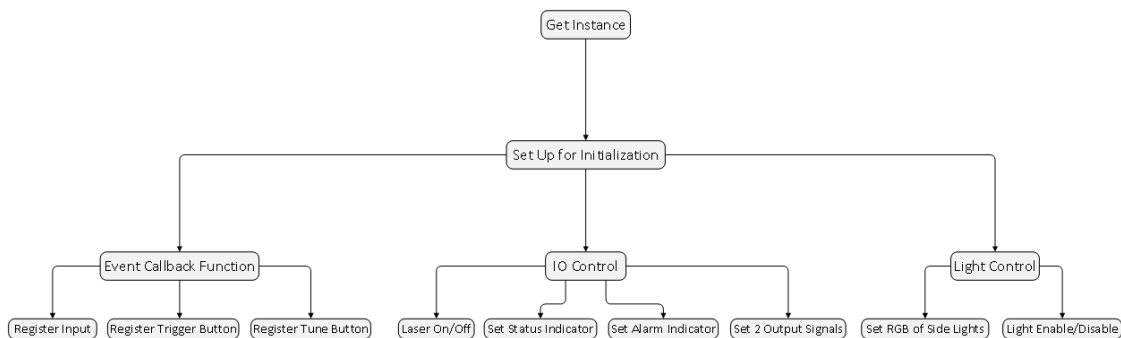
2 Function Description

This chapter introduces how to write the code corresponding to each function to help users write the code required for upper-layer applications.

2.1 IO Control(C++)

This section introduces the operations of indicator control, laser control, event callback, and output control.

2.1.1 Flow Diagram



2.1.2 Getting Instance and Initializing

Before operating IO, you need to obtain an IO instance and initialize the instance. The steps are as follows.

1. Getting an IO instance.

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

2. Initializing the instance.

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

2.1.3 Event Callback Function

IO control supports registering callback functions for events, including registering Input, registering Trigger button, and registering Tune button.

- DI 1 trigger event

```
em->registerInput(trigger_input);
```

The COMMON_IN pin in the 12-Pin M 12 interface is connected to ground signal, and the DI1 pin is connected to 5V signal for triggering.

- Registering Trigger button

```
em->registerTrigger(trigger_trigger);
```

- Registering Tune button

```
em->registerTune(trigger_tune);
```

Sample:

```
#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();
    ....
}
```

C++

2.1.4 Controlling IO

Using IO to control the on/off of the laser, the on/off of the status indicator, the on/off of the alarm indicator and the enable/disable of the 2 outputs.

Preparation

Initialization of the instance has been completed.

Operating Instructions

- Laser On/Off

```
em->openLaser();
```

```
em->closeLaser();
```

- Status indicator On/Off

```
em->setScanStat(true);
```

```
em->setScanStat(false);
```

- Alarm indicator On/Off

```
em->openAlarm();
```

```
em->openAlarm();
```

- 2 outputs enable/disable

```
em->setDo1High(false);
```

```
em->setDo2High(false);
```

2.1.5 Controlling light

Both the camera side lights and area lights can be controlled independently.

Preparation

Initialization of the instance has been completed.

Operating Instructions

- Side light color

```
em->setRgbLight(1);
```

- 0: Closing side light
- 1: Setting the color to Red
- 2: Setting the color to Green
- 3: Setting the color to Blue
- RGB color of side light

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

- Area lights

- Enable (The default state)

```
em->enableLightSection(1);
```

The value range is 1~4, corresponding to different partitions.

- Disable

```
em->disableLightSection(1);
```

The value range is 1~4, corresponding to different partitions.

Enabling/disabling the area light does not turn on/off the light. The area light and the camera are linked. The area light will only turn on when the area light is enabled and the camera is turned on.

2.1.6 Source File

IO Control Class (C++)

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

class EdaIo{
public:
    static EdaIo* getInstance();
    static void close_io();
    ~EdaIo();
    /**
     * @brief Laser On
     *
     */
    void openLaser();
    /**
     * @brief Laser Off
     *
     */
    void closeLaser();
    /**
     * @brief set status indicator
     *
     * @param good
     */
    void setScanStat(bool good);
    /**
     * @brief alarm indicator On
     *
     */
    void openAlarm();
    /**
     * @brief alarm indicator Off
     *
     */
    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 set output1 to [high/low]
     *
     */

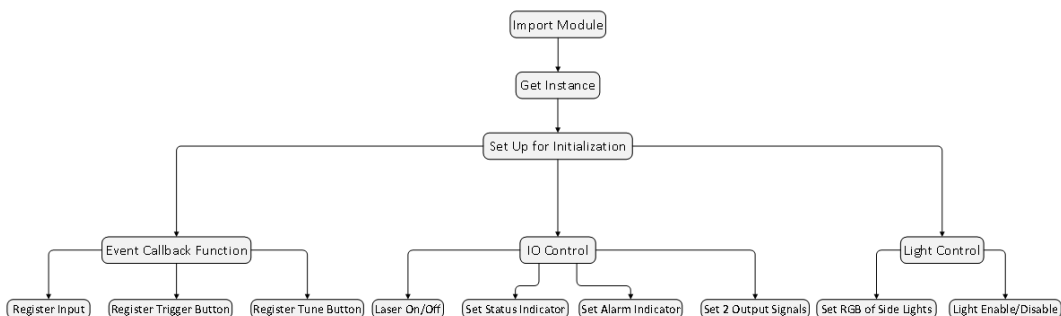
```

```
* @param high
*/
void setDo1High(bool high);
/**
 * @brief set output2 to [high/low]
 *
 * @param high
 */
void setDo2High(bool high);
// void setAimerColor(RGBColor color);
/**
 * @brief register input trigger callback function
 *
 * @param callback
 */
void registerInput(IoTrigger callback);
/**
 * @brief register button callback function
 *
 * @param callback
 */
void registerTrigger(IoTrigger callback);
/**
 * @brief register Tune button callback function
 *
 * @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 initializing IO settings
 *
 */
void setup();
};
```


2.2 IO Control (Python)

This section introduces the operations of indicator control, laser control, event callback, and output control.

2.2.1 Flow Diagram



2.2.2 Import Module

Before operating IO, you need to import modules.

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

2.2.3 Getting Instance and Initializing

After importing the library environment, you need to obtain an IO instance and initialize the instance. The steps are as follows.

1. Getting an IO instance.

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

2. Initializing the instance.

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

2.2.4 Event Callback Function

IO control supports registering callback functions for events, including registering Input, registering Trigger button, and registering Tune button.

- DI 1 trigger event

```
registerInput(func_input);
```

The COMMON_IN pin in the 12-Pin M 12 interface is connected to ground signal, and the DI1 pin is connected to 5V signal for triggering.

- Registering Trigger button

```
registerTrigger(func_trigger);
```

- Registering Tune button

```
registerTune(func_tune);
```

Sample

```
#!/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()
```

C++

2.2.5 Controlling IO

Using IO to control the on/off of the laser, the on/off of the status indicator, the on/off of the alarm indicator and the enable/disable of the 2 outputs.

Preparation

Initialization of the instance has been completed.

Operating Instructions

- Laser On/Off

```
eda.openLaser();
```

```
eda.closeLaser();
```

- Status indicator On/Off

```
eda.setScanStat(true);
```

```
eda.setScanStat(false);
```

- Alarm indicator On/Off

```
eda.openAlarm();
```

```
eda.openAlarm();
```

- 2 outputs enable/disable

```
eda.setDo1High(false);
```

```
eda.setDo2High(false);
```

2.2.6 Controlling light

Both the camera side lights and area lights can be controlled independently.

Preparation

Initialization of the instance has been completed.

Operating Instructions

- Side light color

```
eda.setRgbLight(1);
```

- 0: Closing side light
- 1: Setting the color to Red
- 2: Setting the color to Green
- 3: Setting the color to Blue

- Area lights

- Enable (The default state)

```
eda.enableLightSection(1);
```

The value range is 1~4, corresponding to different partitions.

- Disable

```
eda.disableLightSection(1);
```

The value range is 1~4, corresponding to different partitions.

Enabling/disabling the area light does not turn on/off the light. The area light and the camera are linked. The area light will only turn on when the area light is enabled and the camera is turned on.

2.2.7 Source File

IO Control (Python 3)

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

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

...
eda = EdaIo.singleton(); # Get IO control instance
registerTrigger(func_trigger); # Register Trigger button callback
# registerInput(func_trigger); # Register Input callback
# registerTune(func_trigger); # Register Tune button callback
eda.setup(); # Initialization
...
eda.openLaser(); # Laser On
# eda.closeLaser(); # Laser Off
eda.setScanStat(True); # Set status indicator
eda.openAlarm(); # Alarm indicator on
# eda.closeAlarm(); # Alarm indicator off
```

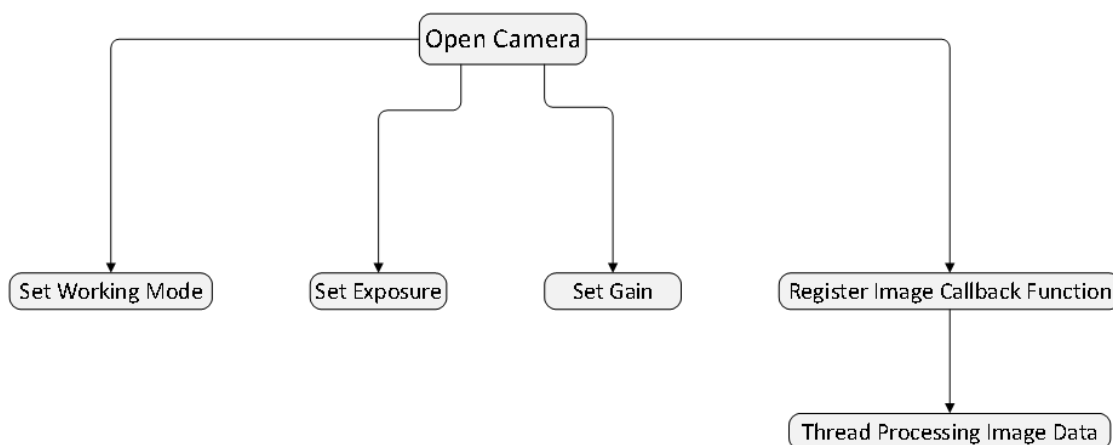
py

```
eda.setDo1High(True); # Set output1
eda.setDo2High(False); # Set output2
eda.setRgbLight(1); # Set side light, 0: Off; 1: Red; 2: Green; 3: Blue
```

2.3 Sensor Control (C++)

This section introduces the operations of opening/closing camera, setting the camera working mode, setting the camera exposure time and setting the camera gain.

2.3.1 Flow Diagram



2.3.2 Operating Steps

Before operating the Camera, you need to obtain the IO instance and initialize it (for specific operations, see [2.1.2 Getting Instance and Initializing](#2.1.2-Getting Instance and Initializing)), and then start the following operations.

1. Getting an instance

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

2. Checking sensor type

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

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

AR0234 is the 2.3-megapixel camera.

OV2311 is the 2-megapixel camera.

3. Open the camera and set working mode, camera area width and camera area height.

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

- mod is the working mode, the value includes 0, 1 and 5.
 - 0 means continuous mode (the camera keeps opening), both AR0234 and OV2311 support this mode.
 - 1 means hardware trigger mode, connecting 5V signal to trigger through trigger pin. Both AR0234 and OV2311 support this mode.
 - 5 means software trigger mode, triggering through manual adjustment. Only OV2311 supports this mode.

```
int call_trigger();
```

- width is area width of camera.
- height is area height of camera.

4. Setting gain

```
t_camera->set_gain(gain);
```

- OV2311: The value range of gain is 0~30.
- AR0234: The value range of gain is 0~64.

5. Setting exposure time

```
t_camera->set_exposure(exposure);
```

- OV2311: The value range of exposure is 0~65523, which unit is microsecond.
- AR0234 : The value range of exposure is 1~1500, which unit of 6.8 times the value is microsecond.

6. Obtaining camera data through callback.

```
t_camera->callback_image_ready(image_callback);
```

In the callback function, it is recommended to only obtain data and not process logic.

7. Close camera

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

2.3.3 Source File

Sensor Control

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

enum CameraName{
    AR0234, OV2311
};

class Camera
{
public:
    /**
     * @brief initializing camera
```

C++

```

*
* @param mode 0 - continuous mode; 1 - hardware trigger mode; 5 - software trigger mode
* @param width
* @param height
* @return int
*/
int open(int mode, int width, int height) = 0;
/**
* @brief close camera
*
* @return int
*/
int close() = 0;
/**
* @brief set exposure time
*
* @param exp_value
* @return int
*/
int set_exposure(int exp_value) = 0;
/**
* @brief obtain exposure time
*
* @param exp_value
* @return int
*/
int get_exposure(int *exp_value) = 0;
/**
* @brief set gain
*
* @param gain_value
* @return int
*/
int set_gain(int gain_value) = 0;
/**
* @brief obtain gain
*
* @param gain_value
* @return int
*/
int get_gain(int *gain_value) = 0;
/**
* @brief register callback function, obtain image data
*
* @param callback
* @return int
*/
int callback_image_ready(img_Callback callback)=0;

CameraName name() = 0;
};

```

Getting AR0234 instance

```

#include "CameraManger.h"
#include "camera_0234.h"

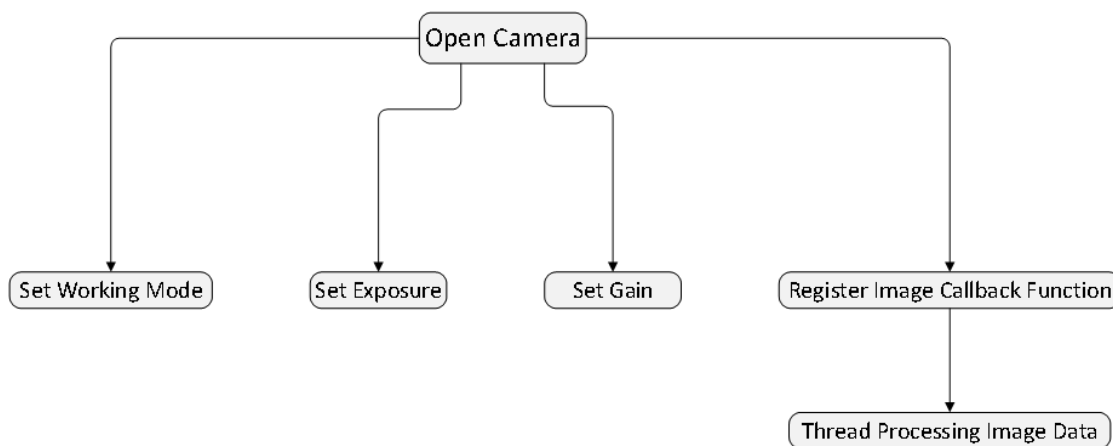
void test()
{
    eda::Camera *t_camera = eda::create_ar0234();
    if(t_camera){
        eda::Camera_0234 *t_camera_1 = static_cast<eda::Camera_0234*>(t_camera);
    }
    ...
}

```

2.4 Sensor Control (Python)

This section introduces the operations of opening/closing camera, setting the camera working mode, setting the camera exposure time and setting the camera gain.

2.4.1 Flow Diagram



2.4.2 Operating Steps

Before operating the Camera, you need to import the IO module first, obtain the IO instance and initialize it (for specific operations, see [2.2.2 Import Module](#2.2.2-Import Module) and [2.2.3 Getting Instance and Initializing](#2.2.3-Getting Instance and Initializing)), and then start the following operations.

1. Import module

```
from libedacamera import EdaCamera
```

2. Getting an instance

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

3. Checking sensor type

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

- return "AR0234"
- return "OV2311"

AR0234 is the 2.3-megapixel camera.

OV2311 is the 2-megapixel camera.

4. Open the camera and set working mode, camera area width and camera area height.

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

- t_mode is the working mode, the value includes 0, 1 and 5.
 - 0 means continuous mode (the camera keeps opening), both AR0234 and OV2311 support this mode.
 - 1 means hardware trigger mode, connecting 5V signal to trigger through trigger pin. Both AR0234 and OV2311 support this mode.
 - 5 means software trigger mode, triggering through manual adjustment. Only OV2311 supports this mode.

```
eda.call_trigger();
```

- t_width is area width of camera.
- t_height is area height of camera.

5. Setting gain

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

- OV2311: The value range of gain is 0~30.
- AR0234: The value range of gain is 0~64.

6. Setting exposure time

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

- OV2311: The value range of exposure is 0~65523, which unit is microsecond.
- AR0234 : The value range of exposure is 1~1500, which unit of 6.8 times the value is microsecond.

7. Obtaining camera data through callback.

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

In the callback function, it is recommended to only obtain data and not process logic.

8. Close camera

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

3 Example

This chapter introduces detailed code examples, including writing code, compiling code, and running code.

3.1 Writing Code

The following takes the function of "turn on the laser, wait for 2 seconds and then turn off the laser" as an example, using C++ language to write the code.

The detailed code is as follows

```
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();  
    //open Laser  
    em->openLaser();  
    sleep(2);  
    // close Laser  
    em->closeLaser();  
    eda::EdaIo::close_io();  
    return 0;  
}
```

After writing is completed, save it as test123.cpp file.

TIP

The file name can be customized.

3.2 Compiling and Running Code

After the C++ code is written, you need to log in to the camera device, compiling and running it on the Raspberry Pi OS.

Preparation:

- The connection of camera cables has been completed. For detailed operations, please refer to the "ED-AIC2000 User Manual".
- The camera has been powered and connected to the network through the router.
- Obtained the camera IP address and successfully logged in to the camera system.

Steps:

Create a folder on the camera OS and upload the code files written in Chapter [3.1 Writing Code] (#3.1-Writing Code) to the folder.

Execute the following command to view the files in the folder and ensure that the code file has been uploaded successfully.

```
ls
```

1. Execute the following command to compile the written code.
2. `g++ -l eda_io -o test-io test123.cpp`

`test123.cpp` : means the code file written in Chapter [3.1 Writing Code](#3.1-Writing Code).

`test-io` : means the file name generated after compilation, the file name can be customized.

Execute the following command to view the new file generated after compilation, as shown below “test-io”.

```
ls
```

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

Execute the following command to run the compiled code.

```
sudo ./test-io
```

`test-io` : means the file name generated after compilation.

TIP

After successful operation, you can see that the laser lights up and goes out after waiting for 2 seconds.