# ED-AIC2000

User Manual

by EDA Technology Co., Ltd

built: 2025-08-01

# 1 Hardware Manual

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

## 1.1 Overview

ED-AIC2000 is a 12-megapixel 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 and eMMC can be selected.

- RAM can choose 1GB, 2GB, 4GB and 8GB
- eMMC can choose 8GB, 16GB and 32GB

ED-AIC2000 series devices are designed with integrated modular light source and feature an M12 fixed focal length lens with liquid module zoom, using bright field and dark field modes, it can achieve the best lighting effect on normal, etched, high gloss or textured surfaces.

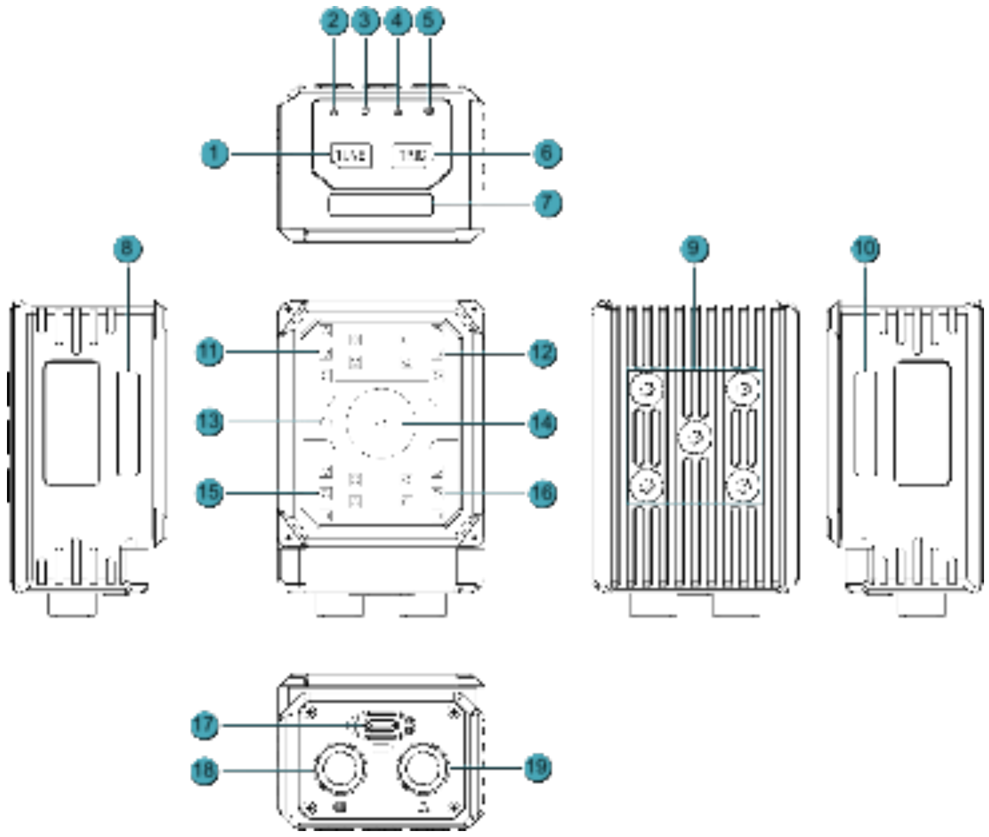
ED-AIC2000 series devices provide power interface, I/O interface, RS232 serial port and Gigabit Ethernet interface, using M12 aviation connector, support IP65 waterproof grade, support network access via Ethernet, and is mainly used in machine vision and artificial intelligence fields.

## 1.2 Packing List

1 x ED-AIC2000 Unit

## 1.3 Appearance

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



| NO. | Function Definition |
|---|---|
| 1 | 1 x TUNE button, one-touch autofocus button or user-defined button. |
| 2 | 1 x Network connection indicator, which is used to check the status of the network connection. |
| 3 | 1 x working status indicator, which is used to check the working status of the device. |
| 4 | 1 x System fault indicator, which is used to check if a system fault occurs. |
| 5 | 1 x Power indicator, which is used to check the power-on status of the device. |
| 6 | 1 x TRIG button, using to camera triggering or user-defined one-touch button. |
| 7, 8, 10 | 3 x RGB lights (1 set), lights can be set to red, green, blue, yellow and white, user customizable. |
| 9 | 5 x M4 screw holes for bracket mounting. |
| 11, 12, 15, 16 | 4 x light sources, using for supplementary lighting when the device is working. |
| 13 | 1 x laser light, red cross laser for photo positioning. |
| 14 | 1 x M 12 Lens, using to take photos. |
| 17 | 1 x USB type-c port, using to flash to eMMC. |
| 18 | 1 x power interface, including power input port, I/O ports and RS232 serial port, using M12 12-Pin aviation connector. |
| 19 | 1 x communication interface, Gigabit Ethernet interface, using M12 8-Pin A-code aviation connector for access to Ethernet. |

## 1.4 Button

The ED-AIC2000 series devices include 2 buttons, an TUNE button and a TRIG button.

- The TUNE 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 TRIG 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 | CM5 Pin |
|---|---|
| TUNE button | GPIO20 |
| TRIG 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 | CM5 Pin |
|---|---|
| Power indicator | N/A |
| System fault indicator | GPIO21 |
| Working status indicator | GPIO7 (abnormal)<br>GPIO16 (normal) |

| Indicator | CM5 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 & I/O Interface

The ED-AIC2000 series devices feature a Power & I/O Interface with a 12-pin M12 aviation connector. This interface integrates:

- 1 power input
- 1 serial port (RS232)
- 1 digital input (DI)
- 2 digital outputs (DOs)

The Power & I/O cable connects to this interface:

- One end is terminated with an M12 connector for camera integration.
- The other end consists of bare wires for connecting to power, DI, DO, and RS232 devices.

The pinout definitions for the M12 connector and the corresponding bare wire assignments are detailed in the following table:

| M12 Pin | Bare Wire Color | Definition |
|---|---|---|
| 1 | Yellow 🟨 | DC- |
| 2 | White/Yellow 🟨⬜ | DC+ |
| 3 | Brown 🟫 | COMMON_IN |
| 4 | White/Brown 🟫⬜ | DI_1 |
| 5 | Purple 🟪 | Trigger |
| 6 | White/Purple 🟪⬜ | COMMON_OUT |
| 7 | Red 🟥 | External Strobe |
| 8 | Black ⬛ | DO_1 |
| 9 | Green 🟩 | DO_2 |
| 10 | Orange 🟧 | RS232_GND |
| 11 | Blue 🟦 | RS232_TX |

| M12 Pin | Bare Wire Color | Definition |
|---|---|---|
| 12 | Gray ▨ | RS232_RX |
| Shell | Black (Thick) ⬛ | PE |

The 1 DI (Digital Input) and 2 DO (Digital Output) channels correspond to the CM4's GPIO pins, as detailed in the following table:

| Signal | CM4 Pin |
|---|---|
| DI1 | GPIO17 |
| DO1 | GPIO22 |
| DO2 | GPIO27 |

TIP

- The thickest black bare wire is PE.
- For Raspberry Pi HQ Camera, Pins 5 and 7 of the M12 connector are undefined.
- During installation, ensure that wire colors align with pin definitions and properly connect the PE to the ground terminal to guarantee safety and signal integrity.
- If there are unused bare wires during wiring, cut off the exposed metal ends or insulate them with electrical tape.

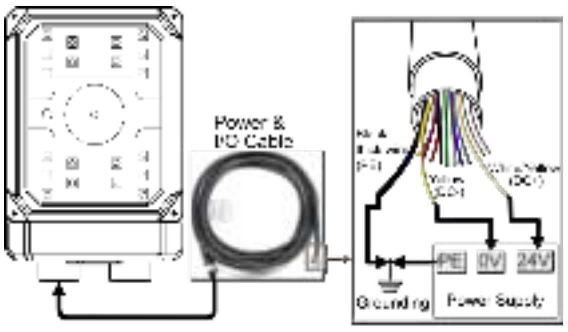
WARNING

Strictly follow the wiring instructions below to connect the power & I/O cable; incorrect wiring may result in device damage.

### 1.6.1.1 Power Connection

For the power & I/O cable:

- White/Yellow and Yellow wires connect to the positive (+) and negative (-) terminals of the external power supply, respectively.
- The thick black wire (PE) connects to the ground terminal.



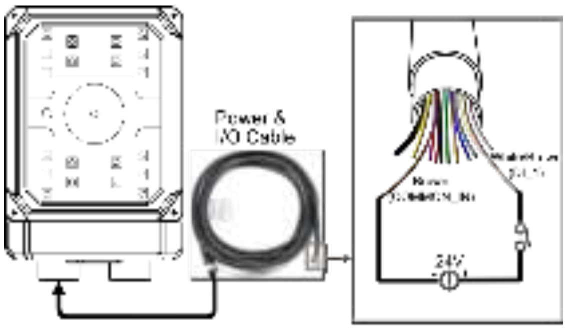
WARNING

- Device Power Supply: DC24V (±10%). It is recommended to use a DC 24V 2A power adapter.
- When connecting the power cable, connect the PE in the bare wires and the external power supply's PE to ground terminal to ensure proper grounding.

### 1.6.1.2 DI Connection

For the power & I/O cable:

- White/Brown is DI Channel 1.
- Brown is the DI common terminal (COMMON_IN).
- Supports NPN-type or PNP-type sensor.



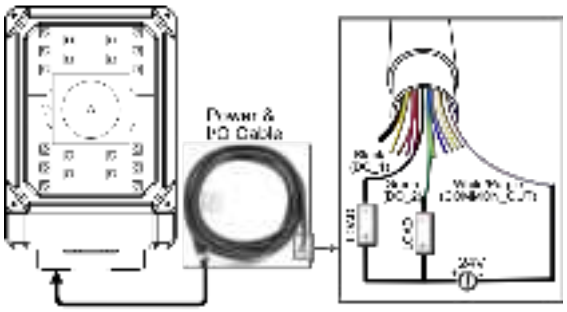
### 1.6.1.3 DO Connection

For the power & I/O cable:

- Black and green wires correspond to the 2 DO channels.
- White/purple wire serves as the common terminal for the DO channels.

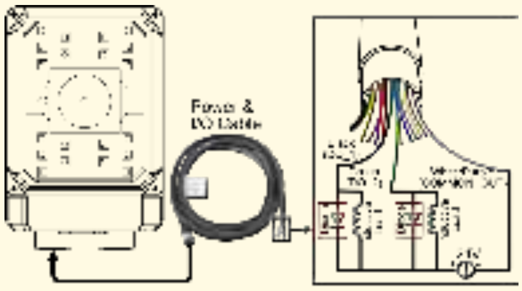
DO Specifications:

- Single-channel load: 0.25A (max).
- Total current for 2 consecutive channels: 0.5A (max).
- Supported load types: Resistive loads, lamp loads, and inductive loads.



WARNING

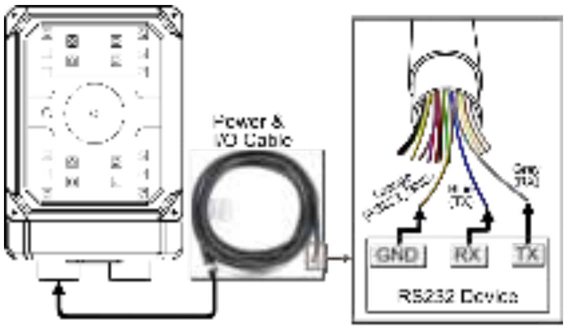
- Do not connect the white/purple wire (COMMON_OUT) to the 24V positive terminal.
- If an inductive load is connected to the DO channel, it is recommended to add a Diode in the circuit (as shown in the figure below) for protection. Select an appropriate Diode based on the specifications of the inductive load.



## 1.6.1.4 RS232 Connection

For the power & I/O cable:

- Orange: RS232_GND
- Blue: RS232_TX
- Gray: RS232_RX
- Compatible with RS232 devices.

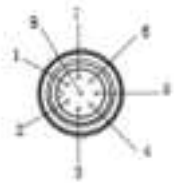


## 1.6.2 Communication interface

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

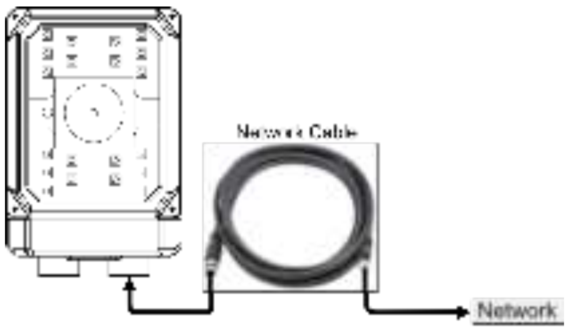
Pin Definition

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- |

Cable Connection

The communication interface is used to connect an Ethernet cable, enabling the device to access the network. One end of the cable is terminated with an M12 connector for connecting to the Camera, while the other end features an RJ45 connector for network integration.

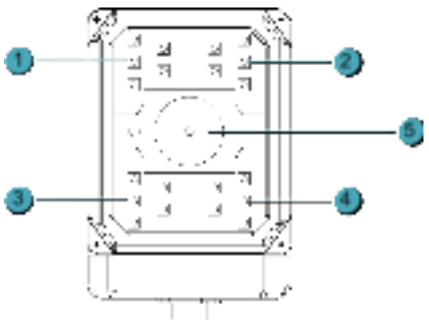


## 1.6.3 USB-C interface

ED-AIC2000 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. |
| 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 | Lens, M12 fixed focal length lens with liquid module zoom. |

## 1.8 RGB Lights

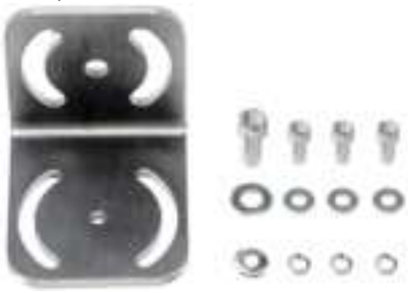
ED-AIC2000 series devices contain 3 RGB lights, 3 RGB lights as a group. It supports to set the lights as red, green, blue, yellow or white through software, the default is off, users can customize according to the actual needs.

> TIP
>
> 3 RGB lights as a group, only support simultaneous setting.

# 2 Installing Device

This chapter introduces how to install the device.

Preparation：

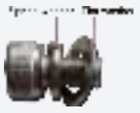
- One M4 and one M6 allen head screwdrivers have been prepared.
- The mounting bracket and mounting screws (3xM4*8 with washers, 1xM6*10 with washers) have been prepared as shown below. 

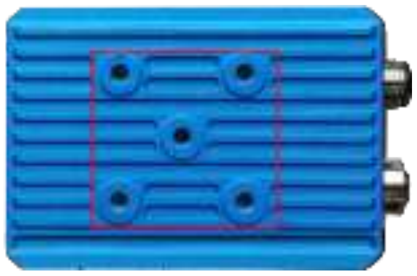
TIP

Mounting bracket and mounting screws are optional and need to be purchased separately.

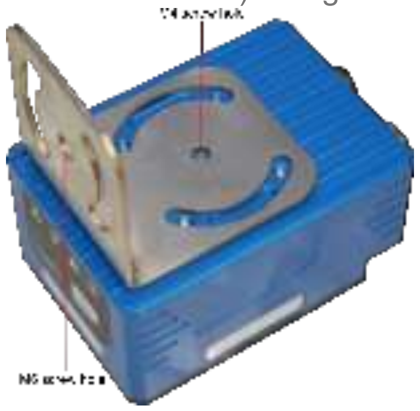
Steps：

TIP

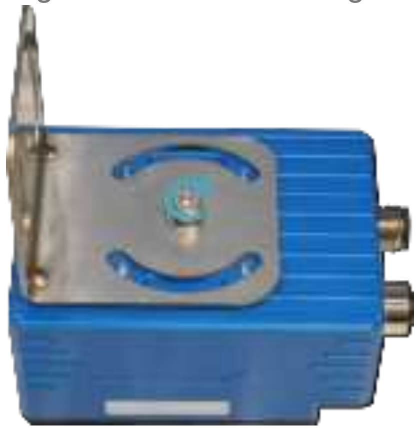
When using M4 and M6 screws, refer to the following diagram to place spring washer and flat washer. 

1. Determine the location of the installation hole on the device, as shown in the red frame in the figure below. 

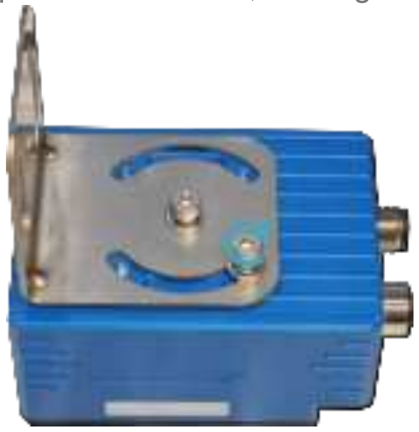
2. Place the Mounting bracket above the camera installation hole, so that the bracket (the side with M4 screw hole) is aligned with the central screw hole of the camera.



3. Insert 1 M4 screw with washer at the location of the center screw hole, then use an M4 hexagonal screwdriver to tighten the M4 screw clockwise to secure the bracket to the device.



4. (Optional) Rotate the Mounting bracket to adjust the mounting direction as required.
5. Insert 1 M4 screw with washer at the location of the peripheral screw hole, then tighten the M4 screw clockwise using an M4 hexagonal screwdriver.



TIP

It is recommended to use a center screw and one peripheral screw for fixing.

4. Use M6 screws to fix the Mounting bracket and other devices.

TIP

Adjust the appropriate installation position and angle according to different project requirements on site.
It is recommended to use a center screw and one peripheral screw for fixing.

# 3 Booting the Device

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

## 3.1 Connecting Cables

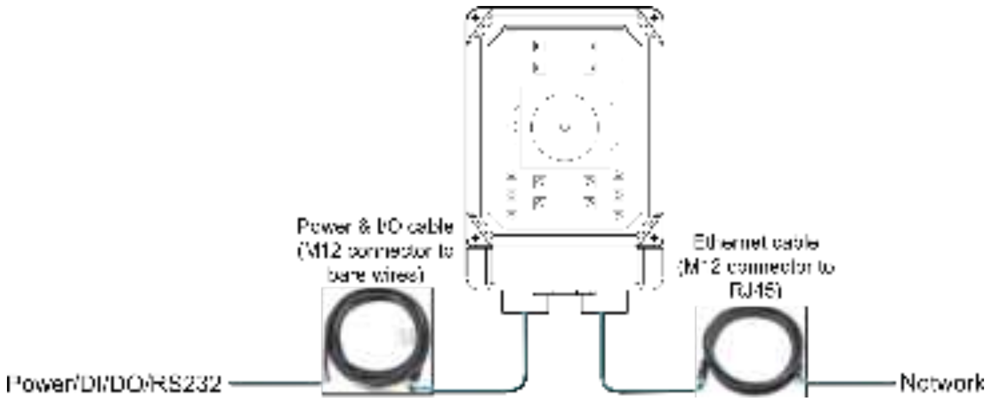
This section describes how to connect cables.

Preparation:

- The Network Cable (M12 connector to RJ45) and Power & I/O Cable (M12 connector to bare wires) to be connected have been obtained.
- A DC 24V 2A power adapter and auxiliary wiring connector have been prepared.
- Prepare the connectors and wiring tools required for the Power & I/O Cable as needed.

Schematic diagram of connecting cables:

For the pin definitions and wiring methods of each interface, see 1.6 Interface.


Power & I/O cable (M12 connector to bare wires)
Ethernet cable (M12 connector to RJ45)
Power/DI/DO/RS232
Network

TIP

The wire colors may vary across different batches of network cables.

## 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 and startup Camera test example.

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

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

2. Compile the test code

- Python

```sh
sudo python test_camera.py
```

- c++

```sh
sudo mkdir test
cd test
sudo cmake .. -DENABLE_HQ=ON
sudo make
sudo ./test_io
```

## 4.2 Camera I/O Control

- The `eda-io.h` file provides the program that operates the control interface (C++) of the AI Camera IO, which is stored at `/usr/include/eda/eda-io.h` .
- The `libedaio.so` file provides the program to operate the control interface (Python) of the AI Camera IO, which is located at `/usr/lib/python3/dist-packages/libedaio.so` .

## 4.3 Camera Sensor Control

The ED-AIC2000 uses the open source `picamera2` library to control the camera, `picamera2` provides a range of API.

## 4.4 Camera API Example

The control functions for the AI Camera Sensor are provided by the `picamera2` library and support turning the sensor on and off, capturing images, etc. Here are some sample descriptions.

| Function | Definition |
|---|---|
| picam2 = Picamera2() | Get Camera Control Instance |
| preview_config = picam2.create_preview_configuration() | Create a preview configuration |
| picam2.configure(preview_config) | Apply the preview configuration to the camera |
| picam2.start_preview(Preview.NULL) | Start the camera preview function, NULL means do not show the picture on the screen |
| picam2.start() | Start the camera hardware and software flow pipeline |
| picam2.capture_file("test.jpg") | Capture an image and save it as test.jpg |
| picam2.close() | Close the camera hardware and software streaming pipeline |

For more details and to check `picamera2` official information Picamera2 Manual (https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf)

## 4.5 I/O API

The I/O API provides control functions for the AI Camera I/O, supporting control of indicators, control of lasers, control of RGB lights, and control of outputs.

### 4.5.1 C++ environment

The control function of AI Camera I/O in C++ environment is as follows:

| Function | Definition |
|---|---|
| eda::EdaIo *em = eda::EdaIo::getInstance() | Get IO control instance |
| void setup() | Initialize IO settings |
| void openLaser() | Turn on the laser |
| void closeLaser() | Turn off the laser |
| void setScanStat(bool good) | Set status indicator |
| void openAlarm() | Turn on the warning light |
| void closeAlarm() | Turn off the warning light |
| void setDo1High(bool high) | Set output1 output |
| void setDo2High(bool high) | Set output2 output |

| Function | Definition |
|---|---|
| void registerInput(IoTrigger callback) | Register input trigger callback function |
| void registerTrigger(IoTrigger callback) | Register trigger button callback function |
| void registerTune(IoTrigger callback) | Register Tune button callback function |
| void setRgbLight(uint8_t light) | Set up RGB lighting |

### 4.5.2 Python3 language environment

The control function of AI Camera I/O in Python3 language environment is shown below:

| Function | Definition |
|---|---|
| eda = EdaIo.singleton() | Get IO control instance |
| eda.setup() | initialization |
| eda.openLaser() | Turn on the laser |
| eda.closeLaser() | Turn off the laser |
| eda.setScanStat(True) | Set status indicator |
| eda.openAlarm() | Turn on the warning light |
| eda.closeAlarm() | Turn off the warning light |
| eda.open_light() | Open light |
| eda.close_light() | Close light |
| eda.eda.enableLightSection() | Enable light(evaluatable：1，2，3，4) |
| eda.eda.disableLightSection() | Disable light(evaluatable：1，2，3，4) |
| eda.setDo1High(True) | Set output1 output |
| eda.setDo2High(False) | Set output2 output |
| registerInput(func_trigger) | Register input trigger callback function |
| 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 I/O, you need to obtain an I/O instance `eda::EdaIo *em = eda::EdaIo::getInstance();`

- Initialize the instance `em->setup();`

2. I/O 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 I/O (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

### 4.5.3.2 Python

1. Initialization

- Get I/O instance before manipulating IO `eda = EdaIo.singleton()`
- Initialize the instance `eda.setup()` .

2. I/O control supports registering callback functions for events.

- input input event `registerInput(func_input)`
- Trigger keystroke `registerTrigger(func_trigger)`
- Tune button `registerTune(func_tune)` .

3. Control I/O (must be initialized first)

- Control of laser `eda.openLaser()` and `eda.closeLaser()`
- Controlling status indicators `eda.setScanStat(true)` and `eda.setScanStat(false)`
- Controls the alarm indicator `eda.openAlarm()` and `eda.openAlarm();`
- Control of two outputs `eda.setDo1High(false)` and `eda.etDo2High(false);`

4. Control the lights (initialization must be done first)

- Control the sidelight color `eda.setRgbLight(1);`
  - 0: OFF

  - 1: Red
  - 2: Green
  - 3: Blue
- Controlling the light source
  - Enable light source (default is enabled) `eda.enableLightSection(1)` Value range 1~4, corresponding to different partitions.
  - Disable light source `eda.disableLightSection(1)` Range 1~4, corresponding to different partitions.

# 5 Installing OS (optional)

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, the download path is shown in the following table.

TIP

Currently only Raspberry Pi OS (Desktop) 64-bit system is supported.

| Product Model | Download Path |
|---|---|
| ED-AIC2000 | 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:

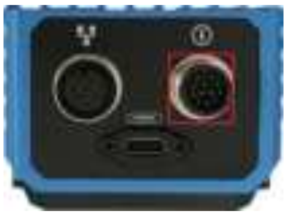
- The download and installation of the flashing tool to the computer has been completed.
- A USB-C to USB-A flashing cable has been prepared.
- An 8-Pin M12 male to RJ45 network cable has been prepared.
- A 12-pin M12 female to bare wire power IO cable has been prepared.
- The image file to be flashsed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

Steps:

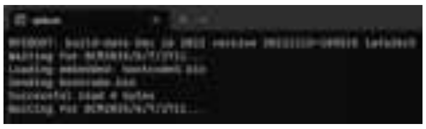
The steps are described using Windows system as an example.

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.

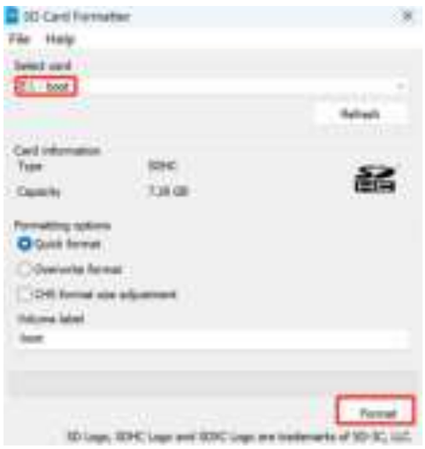


2. Disconnect the device power supply, then press and hold the TRIG button, and power on the device again. The device will automatically enter the flashing mode.
3. Open the installed rpiboot tool to automatically convert the drive to a letter.



```
RPIBOOT: build-date Dec 16 2022 version [illegible]
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.



SD Card Formatter

Select card: E:\ - boot

Card information

| | |
|---|---|
| Type | SDHC |
| Capacity | 7.28 GB |

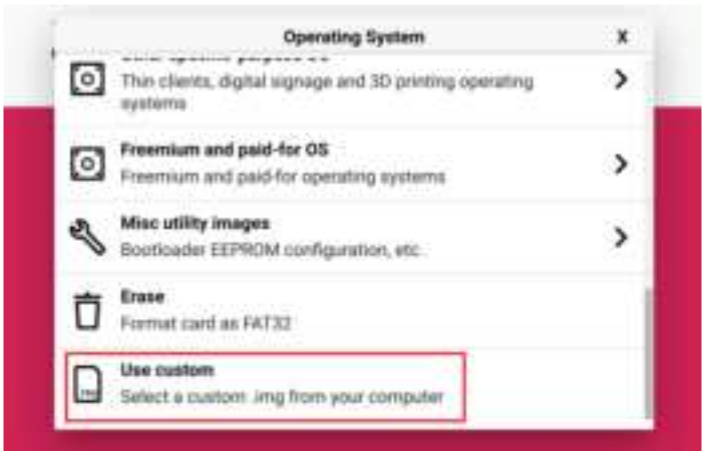
Formatting options: [✓ Quick format] Overwrite format

CHS format size adjustment

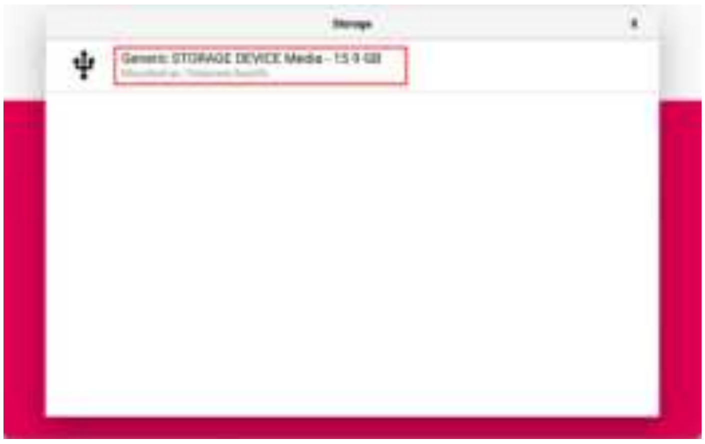
Volume label: boot

SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

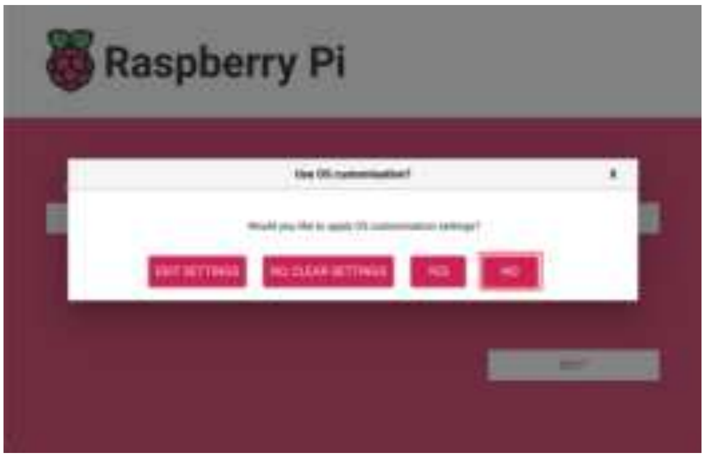
6. In the pop-up prompt box, select "Yes".
7. When the formatting is complete, click "OK" in the prompt box.
8. Close SD Card Formatter.
9. Open Raspberry Pi Imager, select "CHOOSE OS" and select "Use Custom " in the pop-up pane.


Operating System
Thin clients, digital signage and 3D printing operating systems
Freemium and paid-for OS
Freemium and paid-for operating systems
Misc utility images
Bootloader EEPROM configuration, etc.
Erase
Format card as FAT32
Use custom
Select a custom .img from your computer

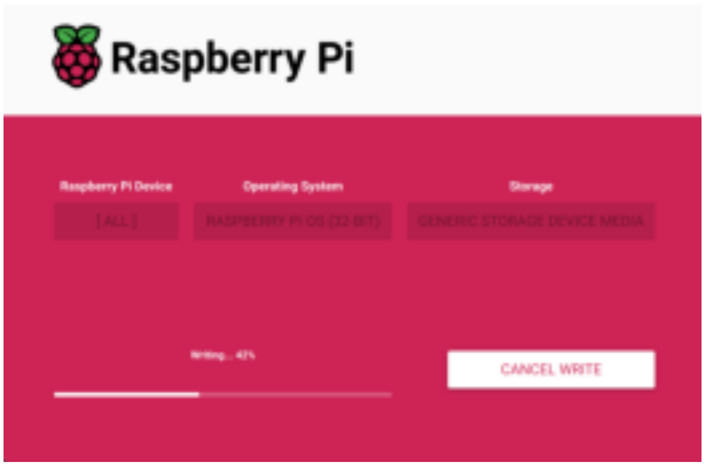
10. According to the prompt, select the OS file under the user-defined path and return to the main page.
11. Click "CHOOSE STORAGE", select the default device in the "Storage" interface, and return to the main page.



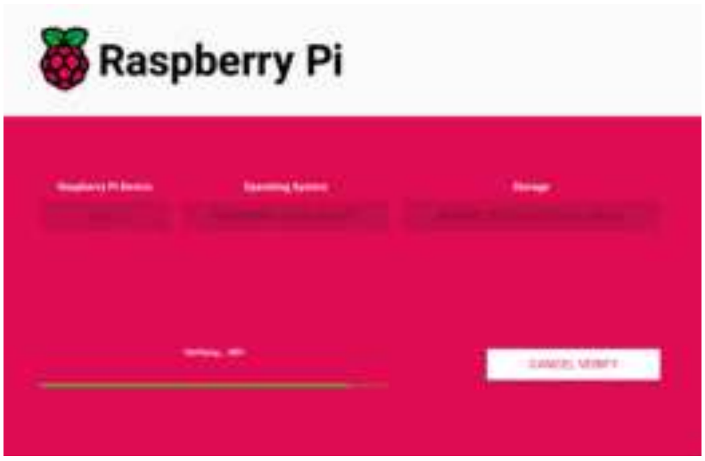
12. Click “NEXT”, select "NO " in the pop-up “Use OS customization?” pane.



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

This chapter introduces the SDK overview, functional description, and development examples.

## 6.1 SDK Overview

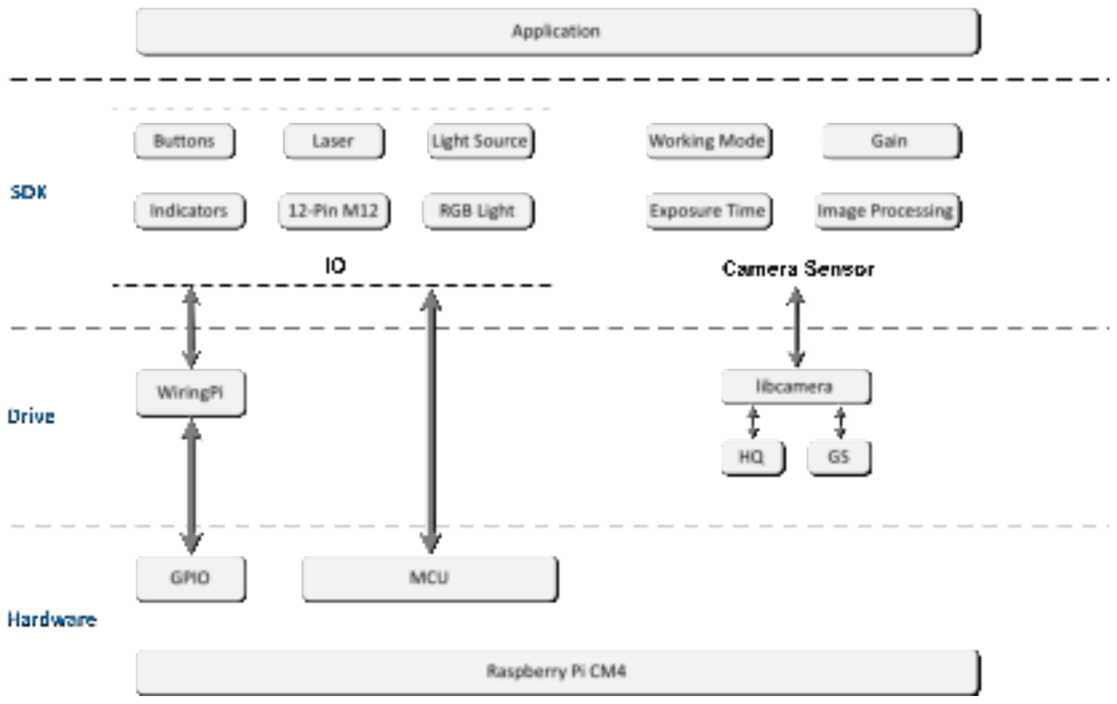
Introducing the definition and composition of SDK to help users understand the SDK better.

### 6.1.1 SDK Introduction

The SDK of ED-AIC2000 series Camera is a set of Software Development Kit (SDK), which provides users with the interfaces required by the upper layer applications, and facilitates the secondary development of the Camera.

The SDK functions of ED-AIC2000 series Camera include the definition of Trigger/Tune button, the definition of DI in 12-Pin M12 interface, the control of laser switch, the control of status indicator, the control of alarm indicator, the control of 2-channel DO, and the control of light and light source.

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



### 6.1.2 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 | Definition | Filename | Installation Path |
|---|---|---|---|
| I/O control | header file | eda-io.h | /usr/include/eda/ |
| | library file | libeda_io.so | /usr/lib/ |

| Function | Definition | Filename | Installation Path |
|---|---|---|---|
|  | Dynamic library files | libedaio.so | /usr/lib/python3/dist-packages/ |
| Camera Sensor Control | Open source library | picamear2 | picamera2 User Manual (https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf) |

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.

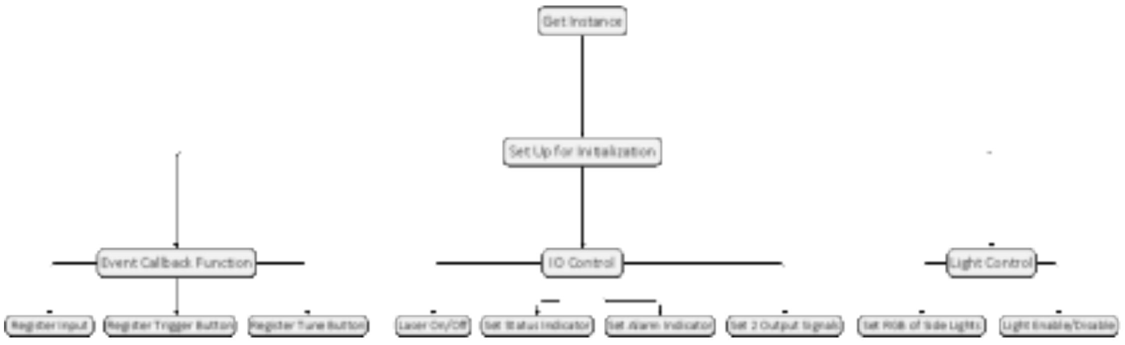
## 6.2 Function Description

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

### 6.2.1 I/O Control(C++)

This section describes the specific operations of indicator control, laser control, event listening, and output control.

#### 6.2.1.1 Flow Diagram



#### 6.2.1.2 Getting Instance and Initializing

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

1. Getting an I/O instance.

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

2. Initializing the instance.

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

### 6.2.1.3 Event Callback Function

I/O 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:

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

### 6.2.1.4 Controlling I/O

The I/O is used to control the laser on/off, status indicator on/off, alarm indicator on/off and 2 output signals enable/disable.

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->closeAlarm();`

- 2 outputs enable/disable

`em->setDo1High(false);`

`em->setDo2High(false);`

### 6.2.1.5 Controlling light

Both the camera side lights and area lights can be controled 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
- 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.

### 6.2.1.6 Code Examples

I/O Control Class (C++)

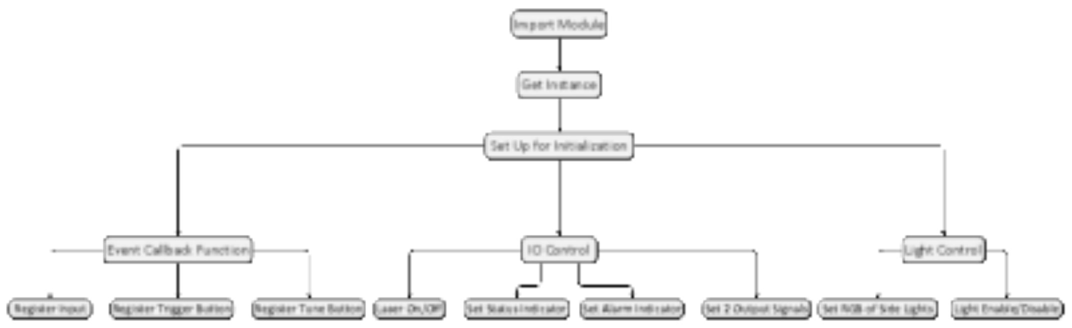
```c++
typedef void (*IoTrigger)(int level);

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 setup();
};
```

### 6.2.2 I/O Control (Python)

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

### 6.2.2.1 Flow Diagram



### 6.2.2.2 Import Module

Before operating I/O, you need to import modules.

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

### 6.2.2.3 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 I/O instance.

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

2. Initializing the instance.

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

### 6.2.2.4 Event Callback Function

I/O 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

```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()
```

### 6.2.2.5 Controlling I/O

Using I/O 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.closeAlarm();`

- 2 outputs enable/disable

`eda.setDo1High(false);`

`eda.setDo2High(false);`

### 6.2.2.6 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.

### 6.2.2.7 Code Examples

I/O Control (Python 3)

```py
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
eda.setDo1High(True); # Set output1
eda.setDo2High(False); # Set output2
eda.setRgbLight(1); # Set side light，0: Off; 1: Red; 2: Green; 3: Blue 4: Yellow 5: White
```

## 6.2.3 Sensor Control Examples

Camera Sensor control software based on open source libraries `picamera2` , official information Picamera2 Manual (https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf) , here are some simple instructions and examples。

### 6.2.3.1 Operating Steps

Before operating Camera, you need to import the IO module and then get the IO instance and initialise it (for details, see 2.2.2 Importing Module and 2.2.3 Getting Instance and Initialising), and then do the following.

1. Importing modules

```
from picamera2 import Picamera2, Preview
```

2. Get the Camera instance

```
picam2 = Picamera2()
```

2. Create the preview configuration

```
preview_config = picam2.create_preview_configuration()
```

3. Apply the preview configuration

```
picam2.configure(preview_config)
```

4. Start the camera preview function, NULL means do not show the picture on the screen

```
picam2.start_preview(preview.NULL)
```

5. Turn on the camera

```
picam2.start()
```

6. Capture the picture

```
picam2.capture_file("test.jpg")
```

8. Close the camera

`picam2.close()` 8. close the camera

## 6.3 Examples

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

### 6.3.1 Code Example

The following is an example of how to implement the function 'Turn on the camera and wait for 2s to capture a picture', using Python lines to write the code.

The detailed code is as follows

```py
from picamera2 import Picamera2, Preview # Import Picamera2 library and preview function.
import time # Import the time module for time-delay operations.

picam2 = Picamera2() # Create a Picamera2 object instance.
camera_config = picam2.create_preview_configuration() # Create the camera's preview configurat
picam2.configure(camera_config) # configure the camera
picam2.start_preview(Preview.NULL) # start the camera's preview function (NULL preview is sele
picam2.start() # start the camera
time.sleep(2) # wait for 2 seconds to make sure the camera is stable
picam2.capture_file('test.jpg') # capture the photo and save it to the current directory, the
picam2.close() # shut down the camera, freeing up resources
```

After writing is completed, save it as `test123.py` file.

> TIP
>
> The file name can be customized.

### 6.3.2 Compiling and Running Code

After the Python 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 Boot device
- 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 Code Example 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.

```sh
sudo python test123.py
```

- `test123.py` : means the code file written in Chapter 3.1 Code Example.

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