ED-AIC2000 Series

Industrial Smart Camera Based on Raspberry Pi CM4

SDK Development Guide

EDA Technology Co., LTD

May 2024

Contact Us

Thank you very much for purchasing and using our products, and we will serve you wholeheartedly.

As one of the global design partners of Raspberry Pi, we are committed to providing hardware solutions for IOT, industrial control, automation, green energy and artificial intelligence based on Raspberry Pi technology platform.

You can contact us in the following ways:

EDA Technology Co.,LTD

Address: Building 29, No.1661 Jialuo Road, Jiading District, Shanghai

Mail: sales@edatec.cn

Phone: +86-18217351262

Website: https://www.edatec.cn

Technical Support:

Mail: support@edatec.cn

Phone: +86-18627838895

Wechat: zzw_1998-

Disclaimer

Foreword

Reader Scope

This manual is applicable to the following readers:

- Software Engineer
- System Engineer

Related Agreement

Terminology Convention

Terminology	Meaning
CM4	Raspberry Pi Compute Module 4

Symbolic Convention

Symbolic	Instruction
✎	Prompt symbols, indicating important features or operations.
⚠	Notice symbols, which may cause personal injury, system damage, or signal interruption/loss.
⚡	May cause great harm to people.

Safety Instructions

- This product should be used in an environment that meets the requirements of design specifications, otherwise it may cause failure, and functional abnormality or component damage caused by non-compliance with relevant regulations are not within the product quality assurance scope.

- Our company will not bear any legal responsibility for personal safety accidents and property losses caused by illegal operation of products.

- Please do not modify the equipment without permission, which may cause equipment failure.

- When installing equipment, it is necessary to fix the equipment to prevent it from falling.

- If the equipment is equipped with an antenna, please keep a distance of at least 20cm from the equipment during use.

- Do not use liquid cleaning equipment, and keep away from liquids and flammable materials.

- This product is only supported for indoor use.

Content

1 SDK Overview

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

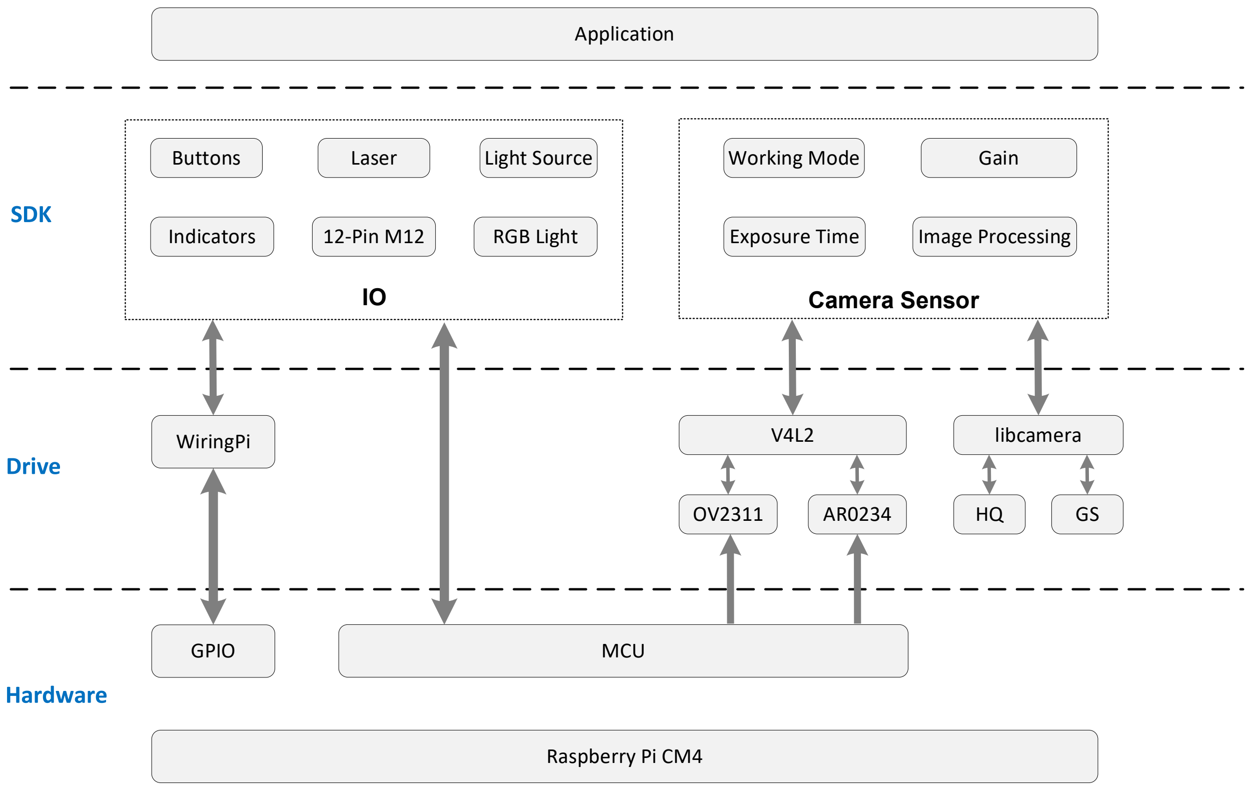
- ✓ SDK Introduction
- ✓ SDK Composition

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/
	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

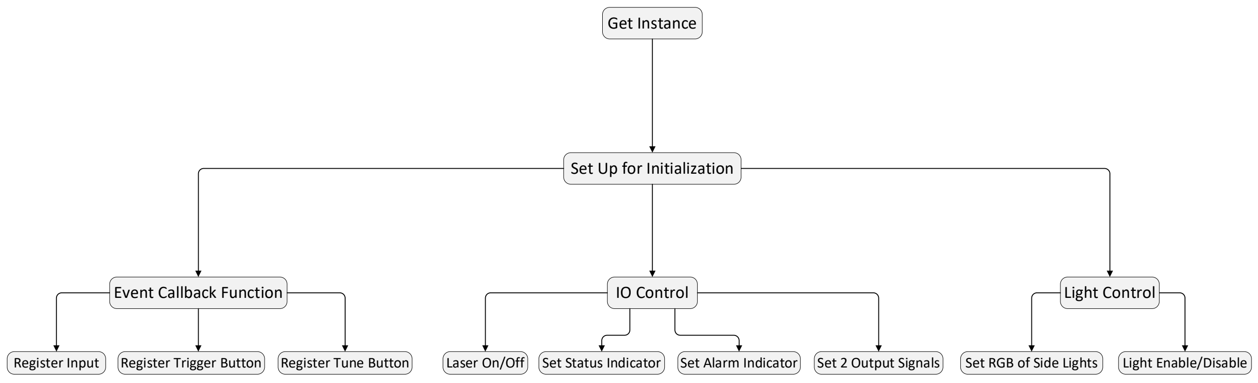
This chapter introduces how to write the code corresponding to each function to help users write the code required for upper-layer applications.

- ✓ IO Control (C++)
- ✓ IO Control (Python)
- ✓ Sensor Control (C++)
- ✓ Sensor Control (Python)

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.

◆ DI1 trigger event

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

The COMMON_IN pin in the 12-Pin M12 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();
    ....
}
```

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
     *
     */
```

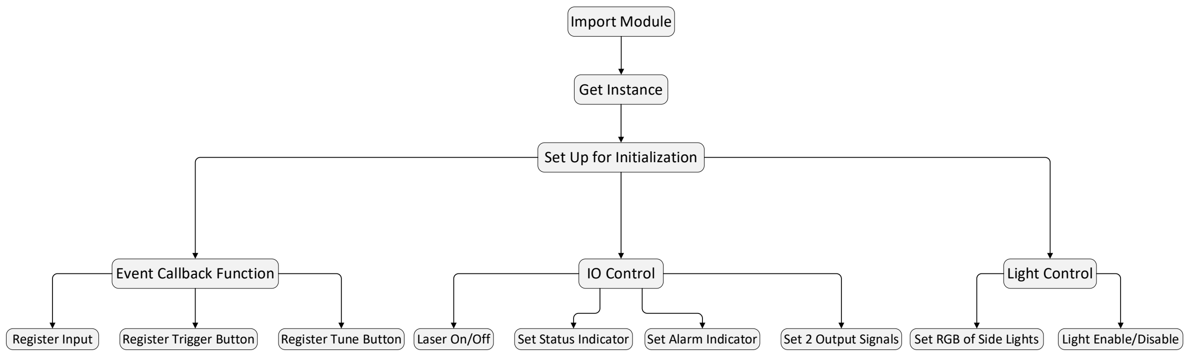
```
void openAlarm();
/**
 * @brief alarm indicator Off
 *
 */
void closeAlarm();
/**
 * @brief
 *
 * @param section 1~4
 * @return int
 */
int enableLightSection(int section);
/**
 * @brief
 *
 * @param section 1~4
 * @return int
 */
int disableLightSection(int section);
/**
 * @brief set output1 to [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.

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

2. Initializing the instance.

   ```
   edaIo.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.

- DI1 trigger event

  registerInput(func_input)

  The COMMON_IN pin in the 12-Pin M12 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()
```

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

  edaIo.openLaser();

  edaIo.closeLaser();

- ◆ Status indicator On/Off

  edaIo.setScanStat(true)

  edaIo.setScanStat(false)

- ◆ Alarm indicator On/Off

  edaIo.openAlarm()

  edaIo.openAlarm()

- ◆ 2 outputs enable/disable

  edaIo.setDo1High(false);

  edaIo.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

  edaIo.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)

  ```
  edaIo.enableLightSection(1);
  ```

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

- Disable

  ```
  edaIo.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（Python3）

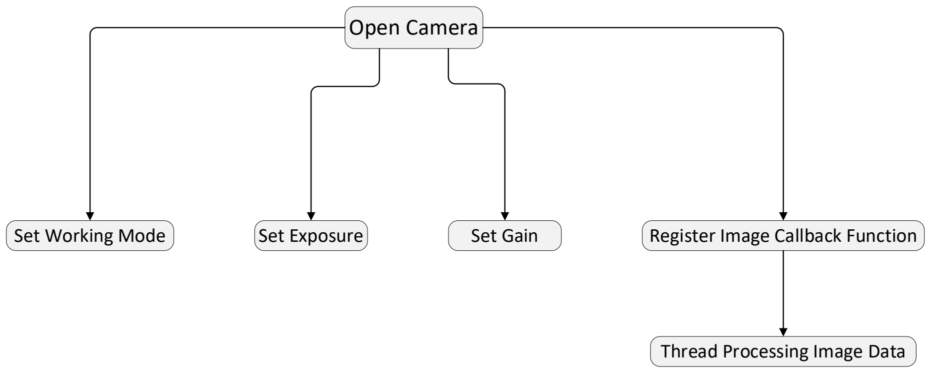
```
from libedaio import EdaIo,registerInput,registerTrigger,registerTune

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
```

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


Open Camera
Set Working Mode
Set Exposure
Set Gain
Register Image Callback Function
Thread Processing Image Data

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**), 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
     *
     * @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;

};
```

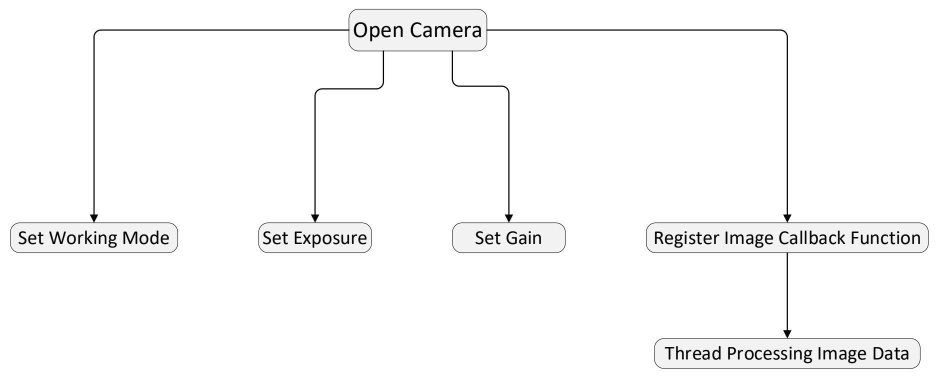
Getting AR0234 instance

```
#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


Open Camera
Set Working Mode
Set Exposure
Set Gain
Register Image Callback Function
Thread Processing Image Data

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** and **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);`

   - ◆ 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.
       `eda.call_trigger();`
   - ◆ width is area width of camera.
   - ◆ 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.

- ✓ Writing Code
- ✓ Compiling 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

```
#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-AIC2020 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:

1. Create a folder on the camera OS and upload the code files written in Chapter 3.1Writing Code to the folder.
2. Execute the following command to view the files in the folder and ensure that the code file has been uploaded successfully.

   ls

3. Execute the following command to compile the written code.

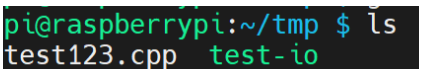
   g++ -leda_io **-o** test-io test123.cpp

   “test123.cpp” means the code file written in Chapter 3.1Writing Code.

   “test-io” means the file name generated after compilation, the file name can be customized.

4. 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
   ```

5. 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.