

ALINX Black Gold Development Board Tutorial

Ultrascale+ Linux Application Development

2020/05/18 V1.01



Copyright Statement

Copyright © 2012-2019 Xinyi Electronics Technology (Shanghai) Co., Ltd. Company

website:

[Http://www.alinx.com.cn](http://www.alinx.com.cn)

Technical forum:

<http://www.heijin.org>

Tmall flagship store:

<https://alinx.tmall.com>

JD Flagship Store:

<http://alinx.jd.com>

Email:

avic@alinx.com.cn

Phone:

021-67676997

Fax:

021-37737073

ALINX WeChat Official Account:



Document Revision Record:

Version	Time	Description
1.01	2020/5/18	Initial Version

We promise that this tutorial is not a one-time effort, a static document. We will continuously revise and optimize the tutorial based on feedback from everyone on the forum, as well as our accumulated practical development experience.

Table of Contents

Copyright Statement	2
Chapter 1: Setting Up a Minimalist Work Environment	6
1.1 Introduction to System Environment Setup	6
1.2 Installation of System and Basic Tools	6
1.3 Installation of the Compilation Environment	6
1.4 Configuration of QtCreator	7
Chapter 2: Hello World with Remote Debugging	12
2.1 Introduction to GDB	12
2.2 Introduction to QtCreator	12
2.3 Experimental Objectives	12
2.4 Qt Creator Project	13
2.5 Get the IP address of the board	13
2.6 Configure GDB in QtCreator to connect to the board	13
2.7 Run Hello World	16
2.8 Parameter Settings	17
Chapter Three Edge Detection in OpenCV	20
3.1 Introduction to OpenCV	20
3.2 Introduction to Edge Detection	20
3.3 Experimental Objectives	21
3.4 Software Flowchart	21
3.5 Preparation for Operation	21
3.6 Program Execution	22
3.7 Code Analysis	22
Chapter 4: Face Detection with OpenCV and Qt	25
4.1 OpenCV's Cascade Classifier	25
4.2 Score Chart	26
4.3 Feature Selection	27
4.4 Introduction to Qt	27
4.5 Experimental Objectives	28
4.6 Software Flowchart	28
4.7 Preparation for Operation	28
4.8 Program Execution	28
4.9 Code Analysis	29
Chapter Five: Displaying the Camera with GStreamer	30
5.1 Introduction to GStreamer	30
5.2 Basic Concepts of GStreamer	31
5.3 Experimental Objectives	32

5.4 Common Tools for Gstreamer	32
5.5 gst-launch-1.0 Camera Display	33
5.6 Running the Program	34
5.7 Code Analysis	34
Chapter Six Camera Display with Qt, DRM, and Gstreamer	35
6.1 Introduction to DRM	35
6.2 Introduction to the Experiment	36
6.3 Experimental Objectives	36
6.4 Running the Program	36
6.5 Code Analysis	37
Chapter 7: Camera Display with Qt and GPU	38
7.1 Introduction to Mali GPU	38
7.2 MPSoc GPU Introduction	38
7.3 V4L2 Introduction	39
7.4 Experimental Objectives	40
7.5 Running the Program	40
7.6 Code Analysis	41
Chapter Eight: Linux Register Operations	42
8.1 Linux Memory Mapping	42
8.2 Introduction to the Experiment	42
8.3 Running the Program	42
8.4 Code Analysis	43

Chapter One: Setting Up a Minimal Work Environment

1.1 Introduction to System Environment Setup

This tutorial uses QtCreator as the development tool on Ubuntu to cross-compile applications. Therefore, on Ubuntu, we need to install cross-compilation tools and various required library files, which can be accomplished using the sdk.sh installation package. The origin of the installation package can be referenced in the PetaLinux chapter's tutorial, which will not be covered here.

1.2 Installation of System and Basic Tools

- (1) In the Windows 10 system, you can refer to the tool installation guide "vmware installation.pdf" for installing the virtual machine software VMware.
- (2) To create a Linux system using ubuntu-18.04.2-desktop-amd64.iso, you can refer to the tool installation guide "ubuntu virtual machine creation.pdf".
- (3) Install the necessary libraries on Ubuntu

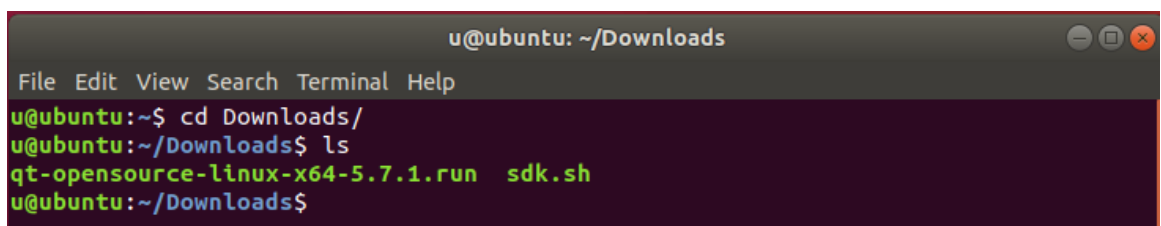
```
sudo apt-get install build-essential
```

Refer to the tool installation guide 'ubuntu library installation.pdf'

- (4) Refer to the tool installation guide "linux 下 Qt 安装.pdf" for installing QtCreator on Ubuntu.

1.3 Installation of the Compilation Environment

- (1) Copy sdk.sh to the Ubuntu system and open the terminal.



```
u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$
```

- (2) Create a tools directory in the root directory and change the permissions.

```
sudo mkdir /tools
sudo chmod 777 -R /tools
```

- (3) Run sdk.sh

```

u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$ ./sdk.sh
PetaLinux SDK installer version 2019.2
=====
Enter target directory for SDK (default: /opt/petalinux/2019.2):

```

- (4) Enter the installation directory, here the installation directory is set to "/tools/xilinx_sdk_tool", and press Enter

```

u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$ ./sdk.sh
PetaLinux SDK installer version 2019.2
=====
Enter target directory for SDK (default: /opt/petalinux/2019.2): /tools/xilinx_s
dk_tool
You are about to install the SDK to "/tools/xilinx_sdk_tool". Proceed[Y/n]? 

```

- (5) Enter "Y" to start the installation, until it is complete

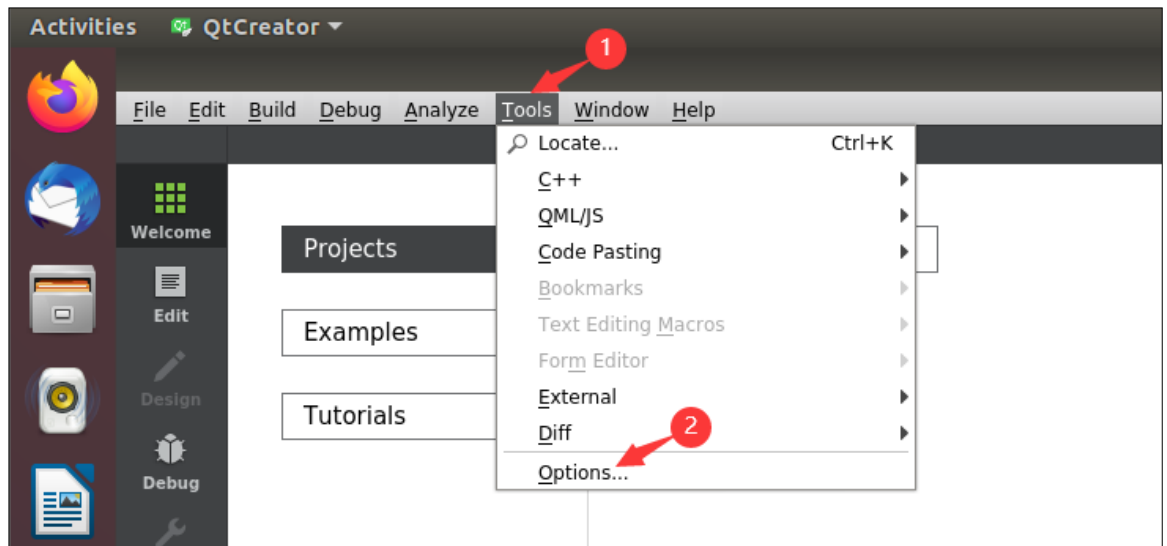
```

u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$ ./sdk.sh
PetaLinux SDK installer version 2019.2
=====
Enter target directory for SDK (default: /opt/petalinux/2019.2): /tools/xilinx_s
dk_tool
You are about to install the SDK to "/tools/xilinx_sdk_tool". Proceed[Y/n]? Y
Extracting SDK.....

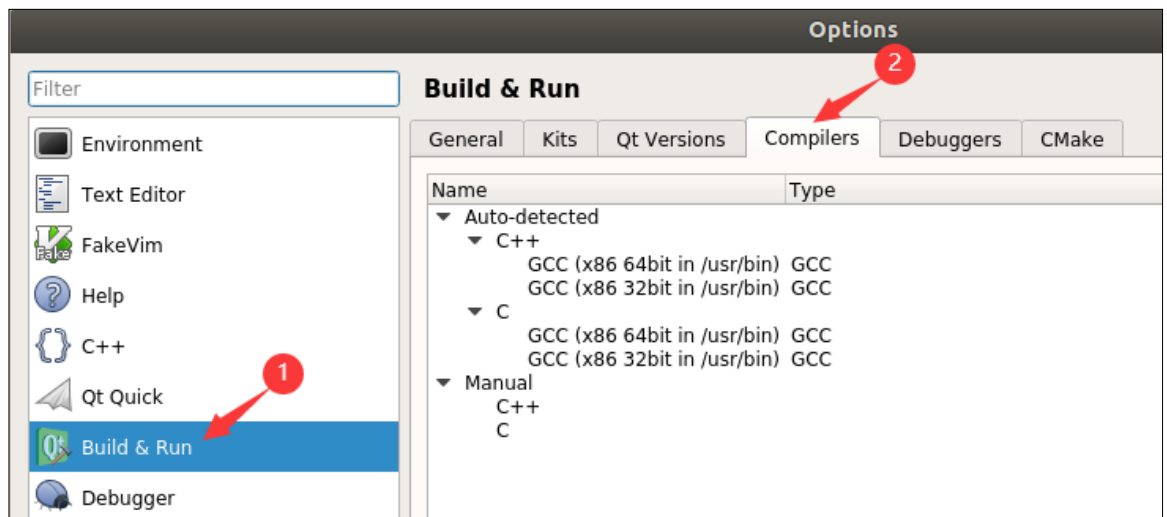
```

1.4 Configuration of QtCreator

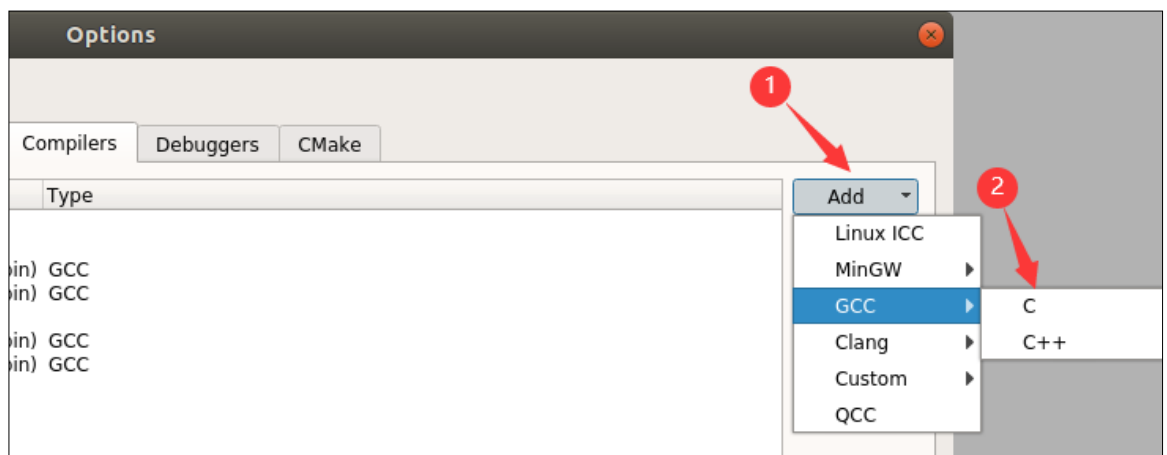
- (1) Open the tool settings in QtCreator



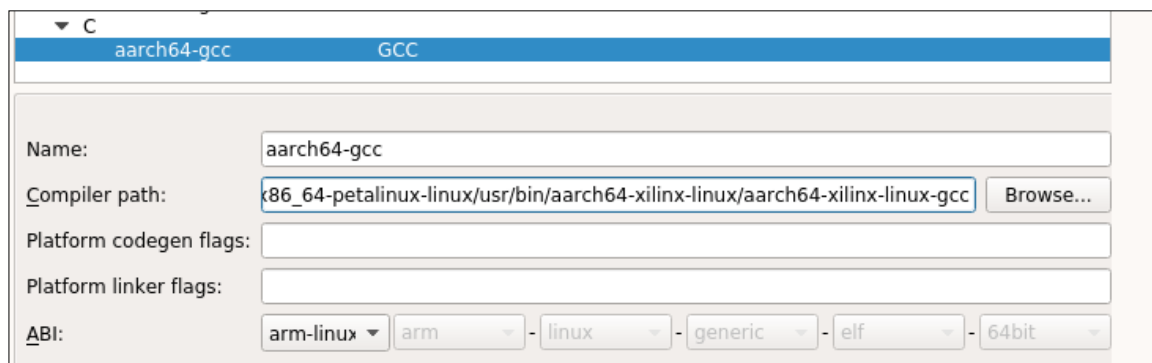
(2) Open the compiler tool settings



(3) Add C compiler tool



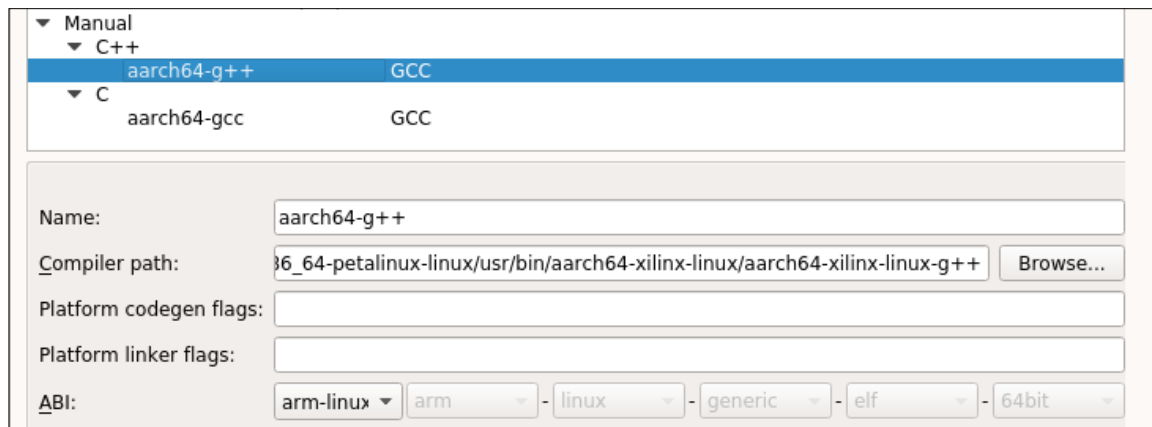
(4) Set name and path



The path is

"/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-gcc"

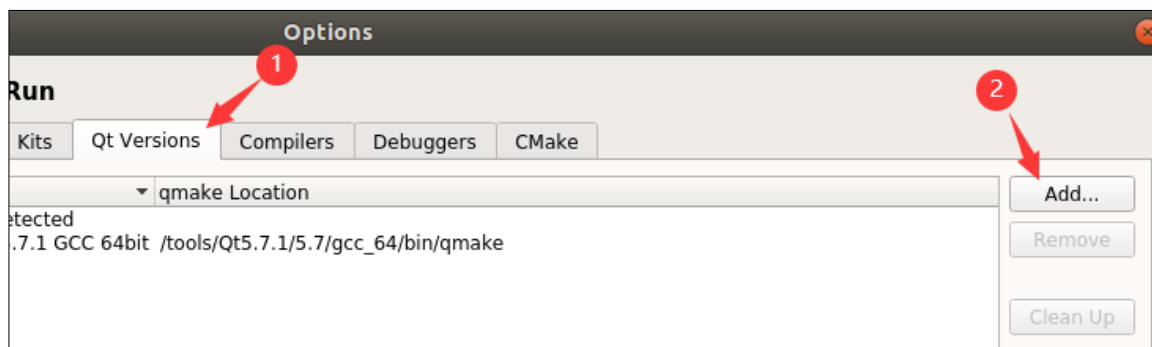
- (5) Add the C++ compilation tool as in the previous steps



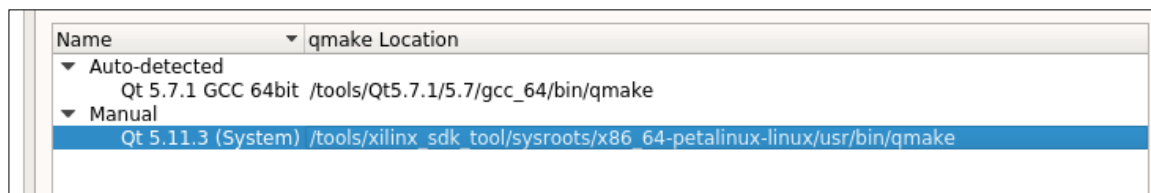
The path is

"/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-g++"

- (6) Click the 'Apply' button to complete this setting
(7) Switch to 'Qt Versions' and click the add button



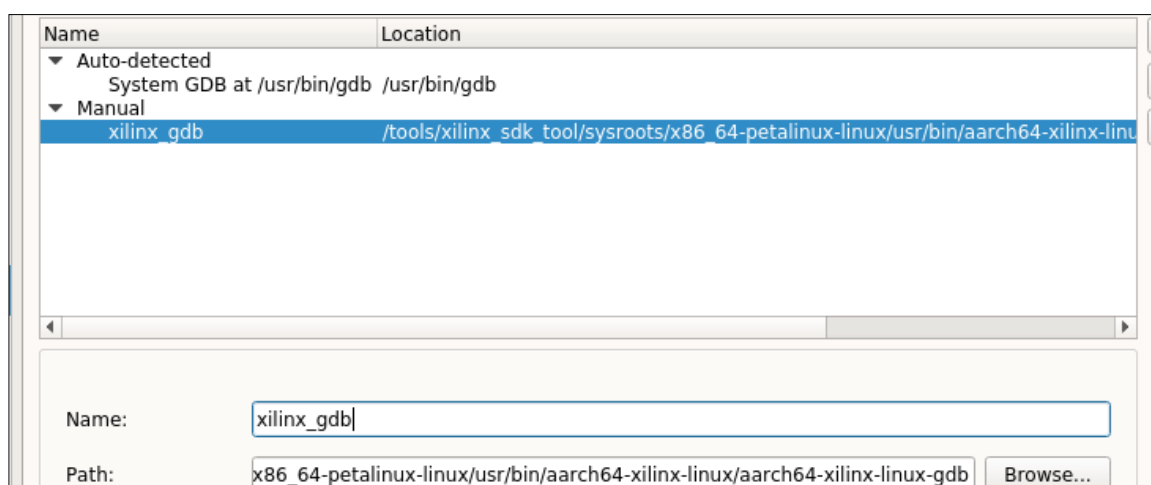
(8) Select qmake



The path is "/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/qmake"

(9) Click the 'Apply' button to complete this setting

(10) Switch to the 'Debuggers' tab and click the 'Add' button



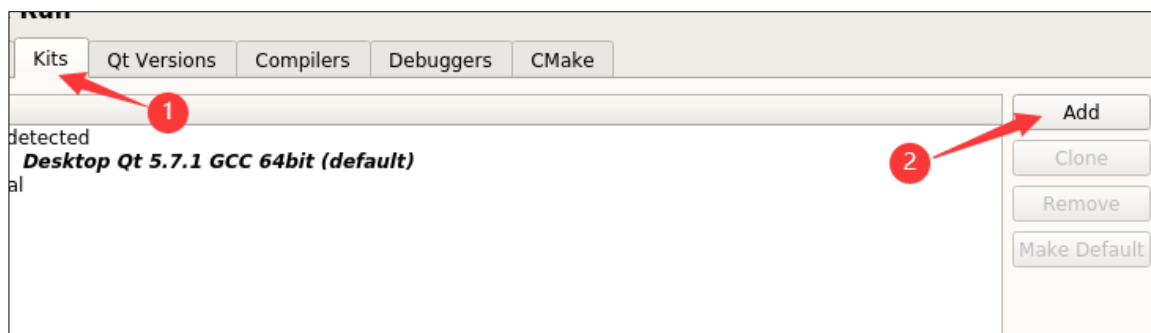
Modify the content of the added item as shown in the figure, where the path

is

'/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-gdb'

(11) Click the 'Apply' button to complete this setting

(12) Switch to the 'Kits' tab and click the 'Add' button



(13) Modify the content of the newly added item as shown in the image

Manual

xilinx_sdk (default)

Name: xilinx_sdk

File system name:

Device type: Generic Linux Device

Device:

Sysroot:

Compiler: C: aarch64-gcc

C++: aarch64-g++

Environment: No changes to apply.

Debugger: xilinx_gdb

Qt version: Qt 5.11.3 (System)

Qt mkspec:

CMake Tool:

CMake generator: CodeBlocks - Unix Makefiles, Platform: <none>, Toolset: <none>

Buttons: Ren, Make, Mana, Brow, Mana, Chan, Mana, Mana, Chan

(14) Click the 'OK' button to complete the settings

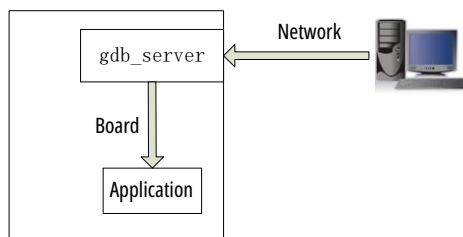
Chapter 2: Hello World with Remote Debugging

2.1 Introduction to GDB

GDB, the GNU Project debugger, allows you to see what is happening 'inside' a program while it is executing, or what a program is doing when it crashes. The main functions are as follows:

- ① Start the program, specifying any factors that may affect its behavior, such as environment variables, runtime parameters, etc. ②
- Make the program stop running under specified conditions.
- ③ Check what happened when the program stopped, such as the state of registers, variable values, etc.
- ④ Manually modify various values during program execution, which can make it easier to verify program functionality.

The GNU debugging tool on Linux is gdb and gdbserver. Both gdb and gdbserver can perform remote debugging of applications running on the target board under Linux. gdbserver is a very small application that runs on the target board, monitors the execution of the debugged process, and communicates with gdb on the host computer over the network. Developers can input commands through gdb on the host computer to control the execution of processes on the target board and view the contents of memory and registers.



2.2 Introduction to QtCreator

Qt Creator is a cross-platform integrated development environment (IDE) that allows developers to create applications across desktop, mobile, and embedded platforms.

The Qt library we usually refer to is the library we call upon to develop desktop applications, which will be linked into our executable program. Qt Creator, on the other hand, is a tool that helps us develop applications. Therefore, the two can be considered completely different things.

2.3 Experiment Objectives

- ① Understand the Qt Creator project
- ② Configure GDB in Qt Creator to connect to the board
- ③ Transfer the executable target program to the board ④

Pass parameters to the running program

2.4 Qt Creator Project

The project file for Qt Creator is xxx.pro. In this file, 'QT=' specifies the QT library components that need to be used. If it is empty, it means that the QT library is not called.

2.5 Obtain the IP address of the board.

- (1) Power on our development board after installing the SD card with the Linux system.
- (2) View IP address

You can check the IP address of the network card using the 'ipconfig' command. The system defaults to enabling DHCP. If the local environment does not support DHCP services,

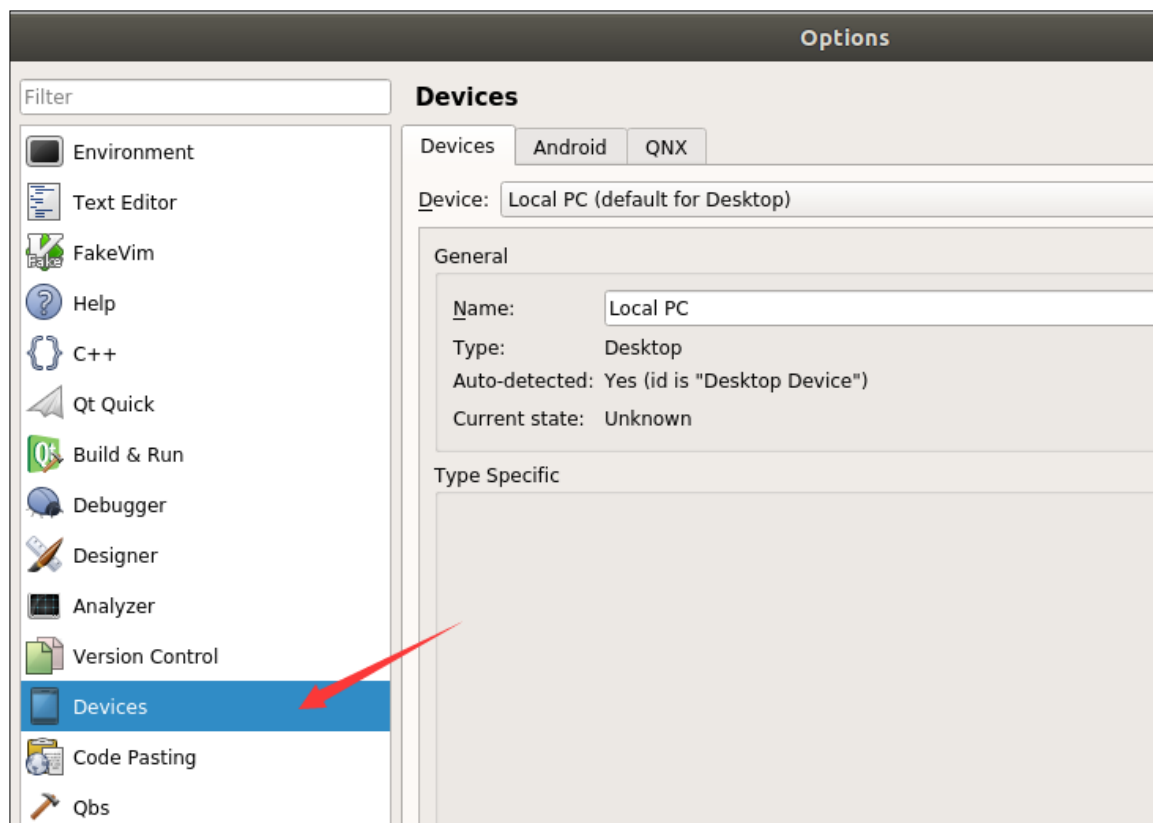
the query result is as shown in the figure below:

```
root@petalinux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:22:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:29
```

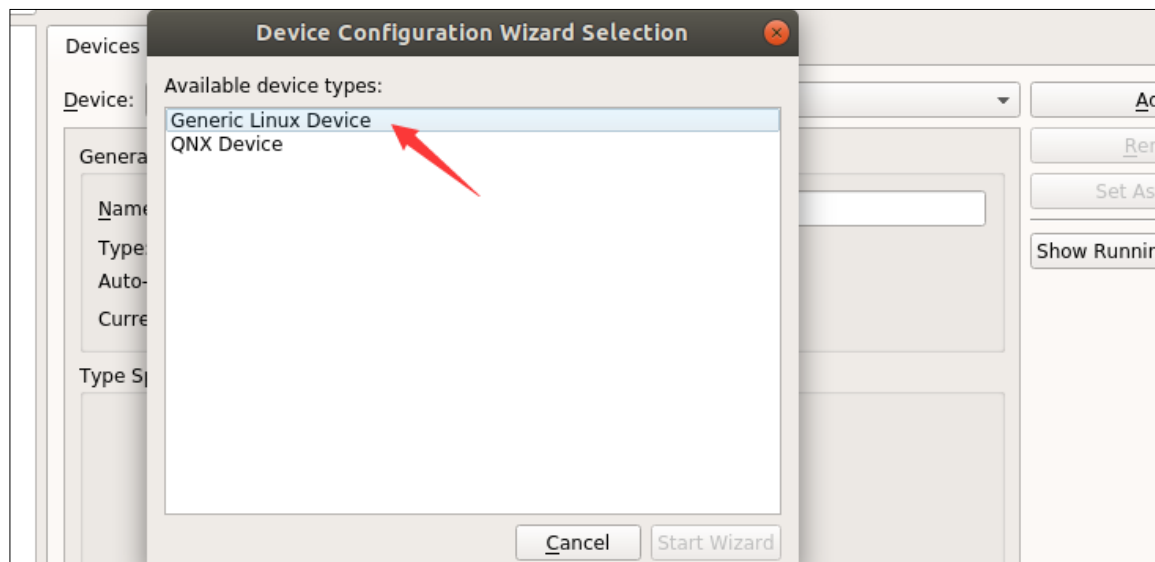
At this point, you need to manually assign an IP address. You can use the command 'ipconfig eth0 192.168.1.45 netmask 255.255.255.0'. Here, we set the IP address of the network card to: 192.168.1.45

2.6 Configure GDB in QtCreator to connect to the board

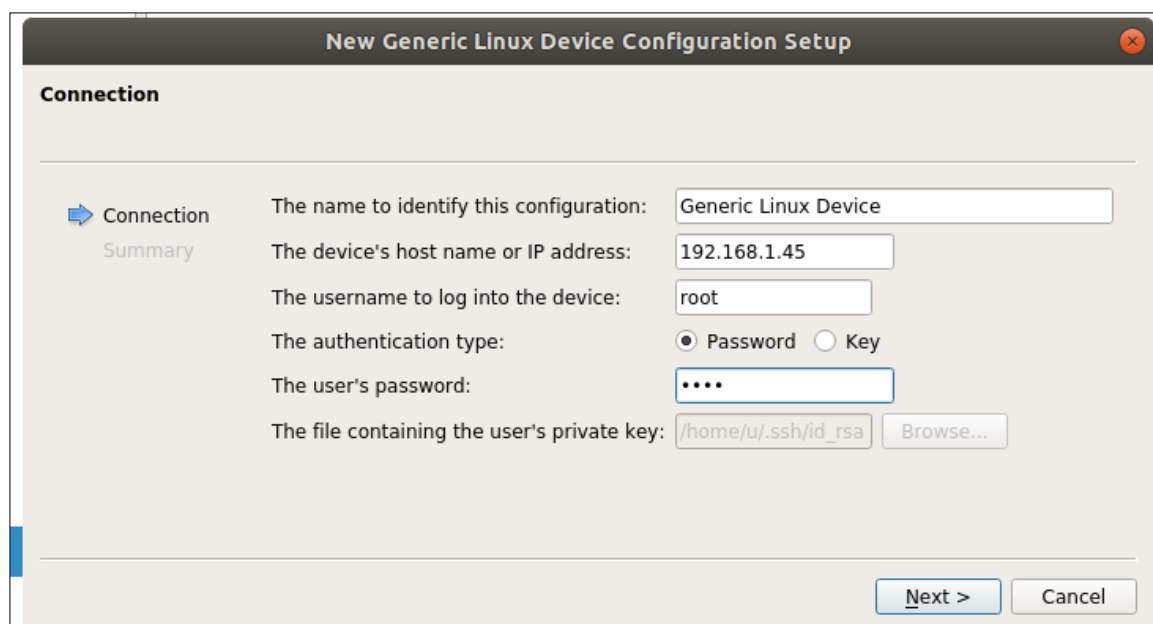
- (1) Open the tool settings in QtCreator



(2) Click the 'Add...' button and select 'Generic Linux Device'



(3) The parameter settings are as follows



The default password is root, which needs to be filled in.

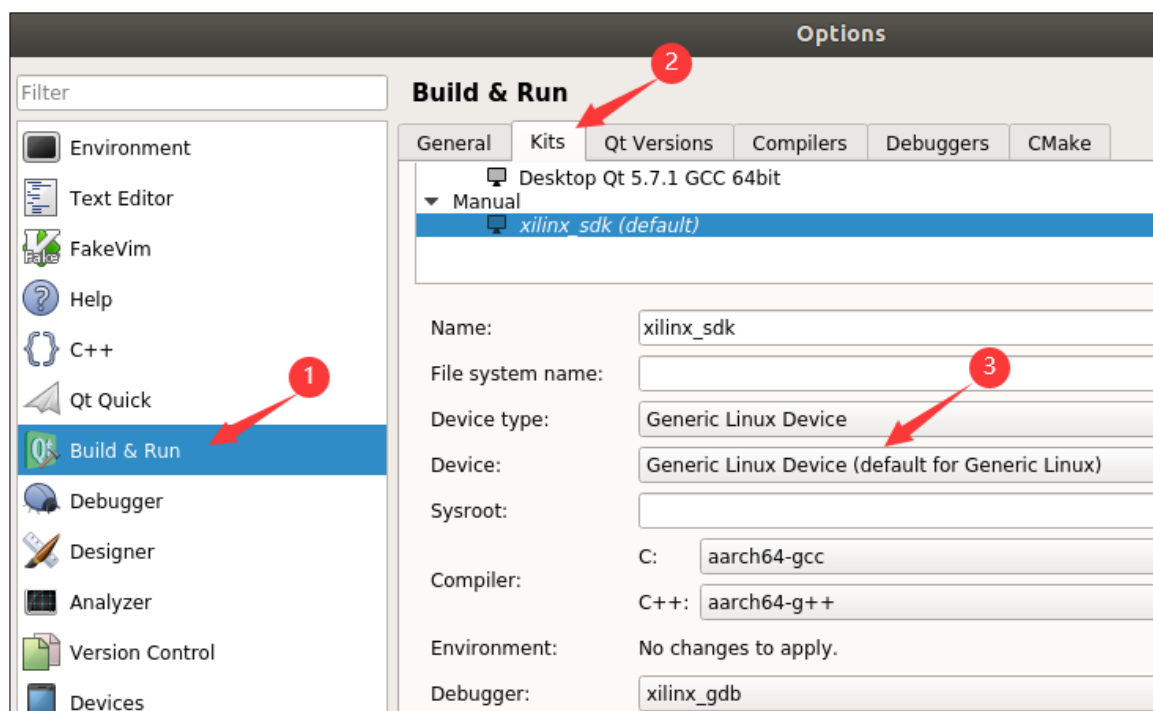
- (4) Click 'Next' and 'Finish', the successful interface that appears in the subsequent test is as follows.



If it is not successful, you need to check the network connection.

- (5) Return to the 'Device' settings, click 'Apply' to complete this setting.

- (6) The settings and configurations to switch are as follows



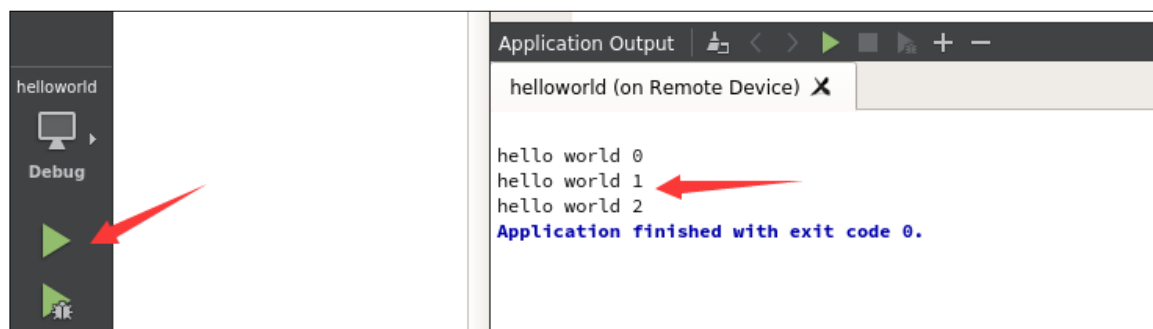
(7) Click 'OK' to complete the settings

2.7 Run Hello World

- (1) Open the helloworld project
- (2) Compile the application



- (3) Click the run button in QtCreator

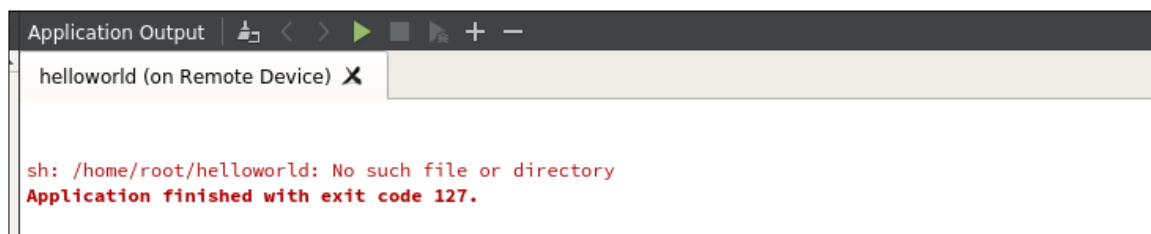


We can see from the running print information on the right that the program has run and exited.

① If the following error is prompted, you need to check whether the board is powered on and has entered the Linux system, and whether the network is functioning properly.

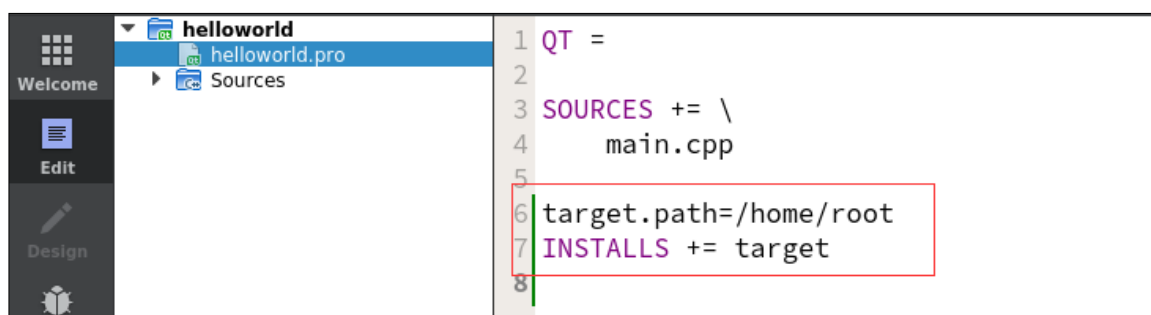


② If the following error is prompted, you need to clear the compilation results and recompile.



(4) At this point, we check the user directory of the board at '/home/root' and find an additional file named helloworld.

(5) We open the project's pro file

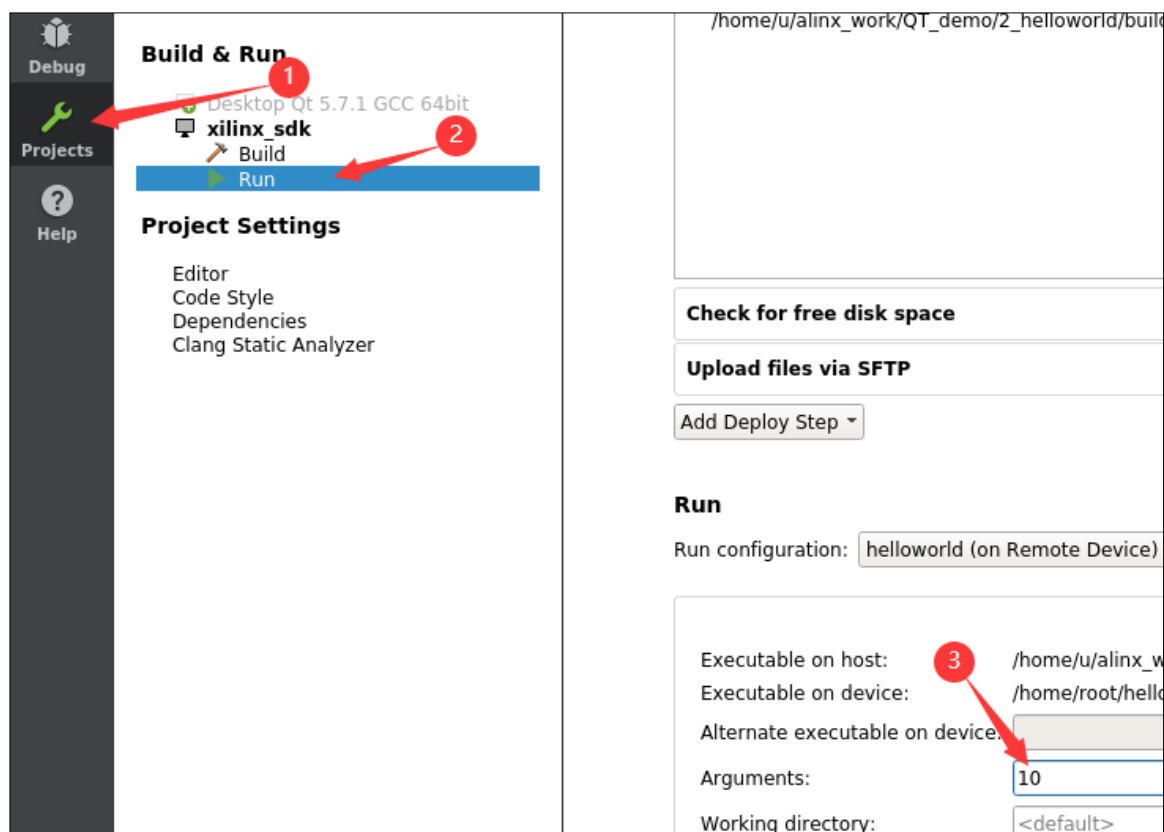


In the red box, it is the directory that the program automatically loads at runtime, which explains why the file helloworld appears in the development board system's '/home/root' directory.

2.8 Runtime Parameter Settings

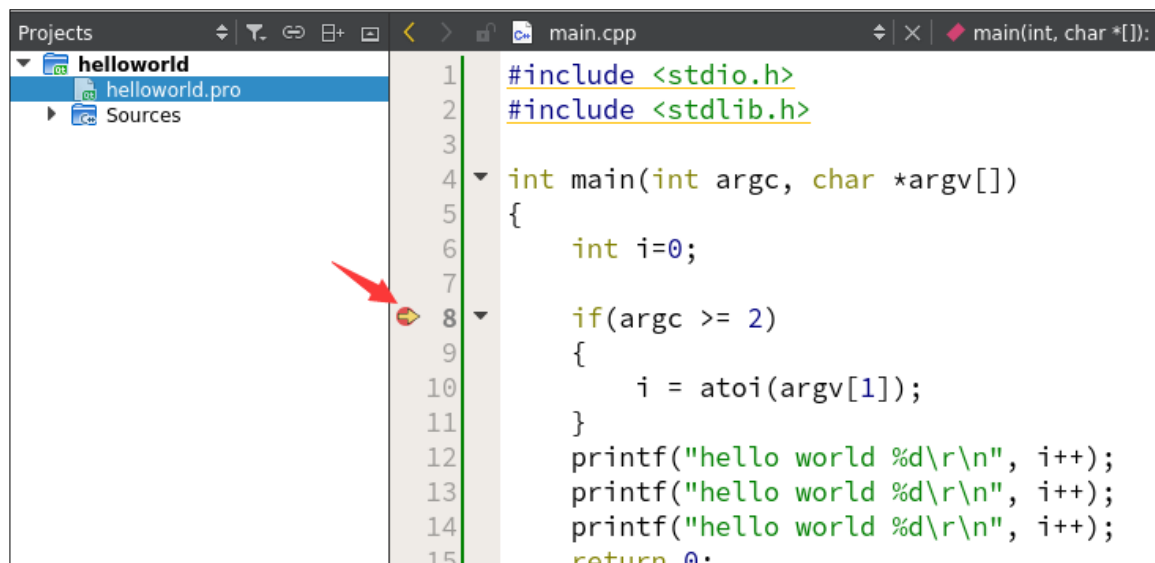
Sometimes, when we run a program, we need to add some runtime parameters at the end. Below, we will introduce how to add runtime parameters when debugging remotely.

(1) Open the following settings item

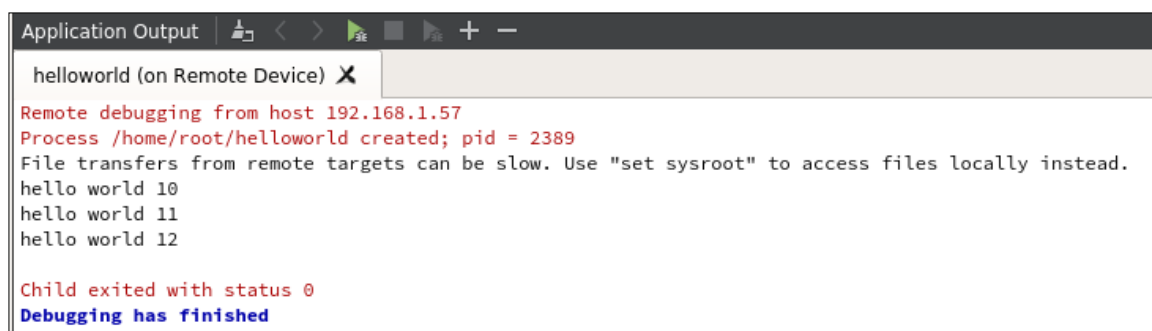


Set the parameter here to 10

- (2) Set a breakpoint at the if location in the main function
- (3) Click the debug button or use the shortcut key F5
- (4) We can see that the program has stopped at the breakpoint we set



- (5) Press F5 to continue running, and we see the printed information as follows



```
Application Output | [Icons] + -
helloworld (on Remote Device) X
Remote debugging from host 192.168.1.57
Process /home/root/helloworld created; pid = 2389
File transfers from remote targets can be slow. Use "set sysroot" to access files locally instead.
hello world 10
hello world 11
hello world 12

Child exited with status 0
Debugging has finished
```

We see that the starting number after 'hello world' has changed to 10, indicating that the parameters we set are now in effect.

Chapter 3: Edge Detection in OpenCV

3.1 Introduction to OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source software library for computer vision and machine learning. The establishment of OpenCV aims to provide a common infrastructure for computer vision applications and accelerate the application of machine perception in commercial products.

As a BSD-licensed product, OpenCV allows enterprises to easily utilize and modify the code.

The library contains over 2,500 optimized algorithms, including a complete set of classical and state-of-the-art computer vision and machine learning algorithms.

These algorithms can be used for detecting and recognizing faces, identifying objects, classifying human behavior in videos, tracking camera motion, tracking moving objects, extracting three-dimensional models of objects, generating three-dimensional point clouds from stereo cameras, stitching images together to create high-resolution images of entire scenes, finding similar images from an image library, removing red-eye from images taken with flash, tracking eye movement, recognizing scenes, and establishing markers for augmented reality, among others. OpenCV has over 47,000 user communities, with an estimated download count exceeding 18 million. It is widely used by companies, research groups, and government agencies.

Many implementations of Xilinx's reVision are based on the OpenCV interface, with the hardware implementation being xfpencv. This will help us achieve the hardware (FPGA) acceleration of the corresponding algorithms.

3.2 Introduction to Edge Detection

Edge detection is a fundamental problem in image processing and computer vision, aimed at identifying points in a digital image where there is a significant change in brightness. Notable changes in image attributes often reflect important events and variations in properties. These include discontinuities in depth, discontinuities in surface orientation, changes in material properties, and variations in scene lighting. Edge detection is a research area in image processing and computer vision, particularly in feature extraction.

Image edge information is primarily concentrated in the high-frequency band. When we talk about image sharpening or edge detection, it essentially refers to high-frequency filtering. We know that differential operations are used to determine the rate of change of a signal, which enhances high-frequency components. In spatial domain operations, sharpening an image involves calculating the derivative. Due to the discrete nature of digital images, differential operations become the calculation of differences or gradients. There are various edge detection (gradient) operators in image processing, commonly including ordinary first-order differences, the Robert operator (cross difference), the Sobel operator, etc., which are based on finding gradient intensity. The Laplacian operator (second-order difference) is based on zero-crossing detection. By calculating the gradient and setting a threshold, we obtain the edge image.

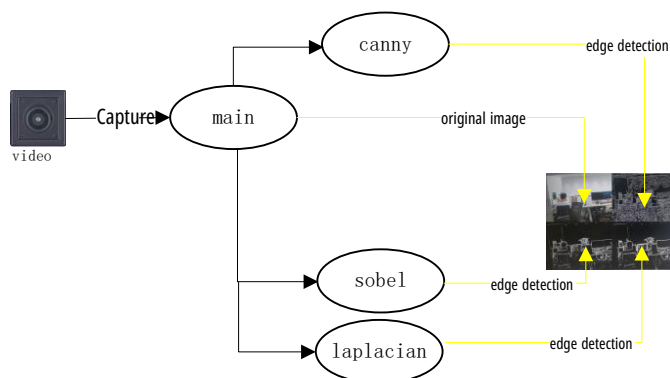
The general steps of edge detection are:

- ① Filtering: Edge detection algorithms are primarily based on the first and second derivatives of image intensity; however, derivatives are very sensitive to noise, so filters need to be used to improve the performance of edge detectors related to noise.
- ② Enhancement: The basis of enhancing edges is to determine the change in intensity values in the neighborhood of each point in the image. Enhancement algorithms can highlight points where the intensity values of neighboring gray points show significant changes.
- ③ Detection: There are many points in the neighborhood with large gradient values, but in specific applications, these points are not necessarily the edge points we are looking for, requiring a selection process.

3.3 Experimental Objectives

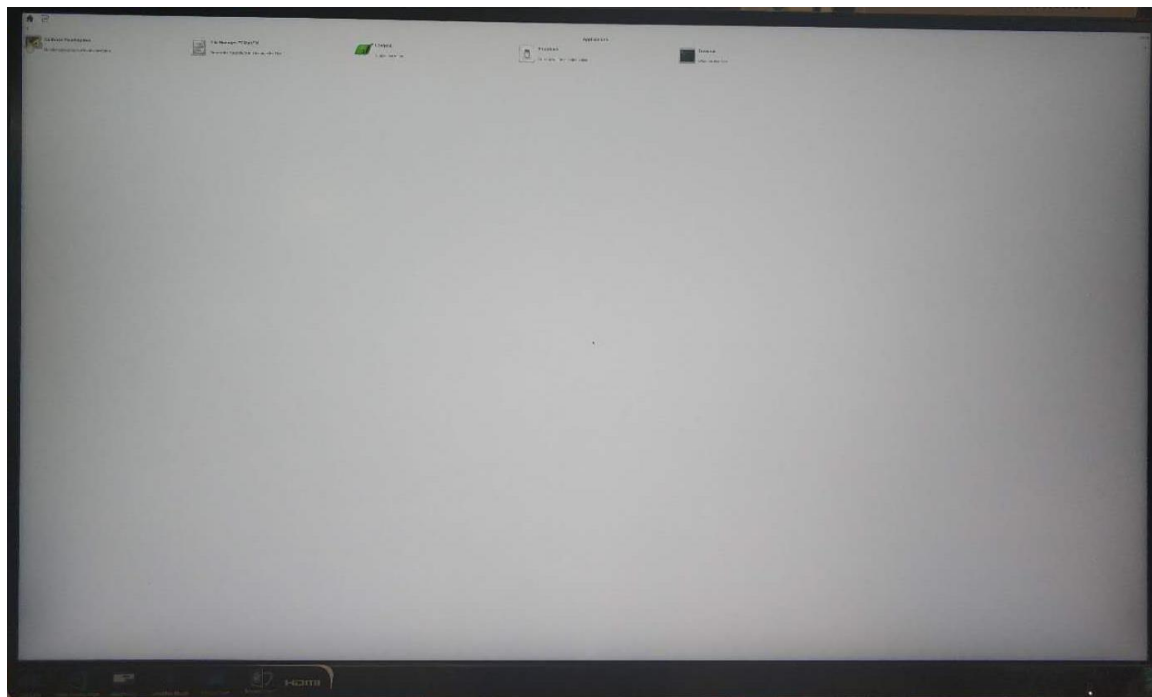
- ① Use OpenCV to capture images from a USB camera
- ② Edge detection using Canny, Sobel, and Laplacian algorithms
- ③ Use OpenCV for image display
- ④ Respond to OpenCV window close operations

3.4 Software Flowchart



3.5 Preparation for operation

- (1) Connect the DP interface to the display
- (2) Plug in the USB camera
- (3) After powering on the development board, wait for the system to complete the startup, and the DP display can correctly show the boot desktop.

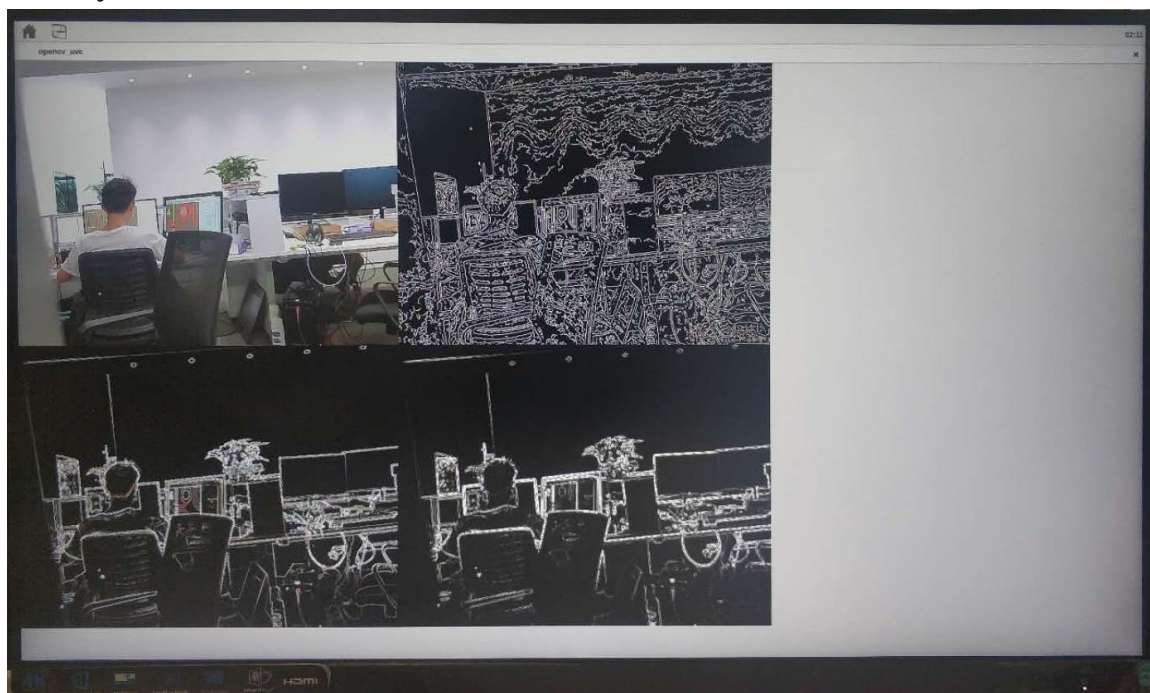


- (4) In the terminal, the USB camera device can be viewed.

```
root@petalinux:~# ls /dev/video*
/dev/video0 /dev/video1
root@petalinux:~#
```

3.6 Program running

- (1) You can refer to the example in the previous chapter, open the 3_opencv_edgedete project, and run the application directly using QtCreator. The running effect is as follows.



3.7 Code Analysis

- (1) Set Environment Variables

Enter the following command in the terminal, which is equivalent to the setenv call in the code.

```
export DISPLAY=:0.0
```

- (2) Start the desktop

```
/etc/init.d/xserver-nodm_xx start
```

- (3) Adjust the display resolution

As in the above experiment, using a 4K monitor, the icons on the desktop are very small. For better display quality, in the application, the display output resolution is set to 1080P through a system call. The command is as follows:

```
xrandr --output DP-1 --mode 1920x1080
```

- (4) Display output management

The system defaults to using power management for the output, which means the monitor will go black after a period of inactivity. At this point, we can use the command,

Cancel this function, the command is as follows:

```
xset s 0 0
xset dpms 0 0 0
```

(5) Multithreading operation

This experiment uses three different algorithms to complete edge detection, with the three algorithms assigned to three thread tasks, which can fully utilize the ARM's quad-core A53.

(6) USB camera capture

The camera resolution is set to 640x480, and the default camera device opened is: /dev/video0. Since the algorithm computation is a bit slow, the displayed image will have a slight delay.

(7) Canny edge detection

The multi-level edge detection algorithm developed by John F. Canny in 1986 is considered by many to be the optimal edge detection algorithm. The three main evaluation criteria for optimal edge detection are:

- Low error rate: Identifying as many actual edges as possible while minimizing false positives caused by noise.
- High localization: The identified edges should be as close as possible to the actual edges in the image.
- Minimum response: Edges in the image should be identified only once.

cv:: Canny function parameters are as follows:

InputArray	image,	Input image
OutputArray	edges	Output edge image
double	threshold1	Threshold 1
double	Threshold2	Threshold 2
int	apertureSize=3	Sobel operator size
bool	L2gradient=false	Is a more precise method used to calculate image gradients?

(8) Sobel edge detection

This is a commonly used edge detection method. The Sobel operator combines Gaussian smoothing and differential derivation, also known as a first-order differential operator. It performs differentiation in both horizontal and vertical directions, resulting in gradient images in the X and Y directions. Disadvantages: It is quite sensitive and easily affected, requiring Gaussian blur (smoothing) to reduce noise, and the edge localization accuracy is not high enough.

The parameters of the cv::Sobel function are as follows:

InputArray	src,	Input image
OutputArray	dst	Output edge image
int	ddepth	Output image depth
int	dx	The order of the derivative of x.
int	dy	The order of the derivative of y.
int	ksize = 3	Size of the Sobel kernel
double	scale = 1	Optional scaling factor for the computed derivative values

double	delta = 0	An optional incremental value to add to the result before storing it in dst.
Int	borderType=BO RDER_DEFAULT	Pixel extrapolation method

(9) Laplacian edge detection

The Laplace operator is a second-order differential operator in n-dimensional Euclidean space, which is more suitable when only the position of the edge is concerned without considering the gray value differences of surrounding pixels. The response of the Laplace operator to isolated pixels is stronger than its response to edges or lines, thus it is only applicable to noise-free images. In the presence of noise, low-pass filtering must be performed before using the Laplacian operator for edge detection. Therefore, typical segmentation algorithms usually combine the Laplacian operator with smoothing operators to generate a new template. Like the Sobel operator, the Laplace operator belongs to spatial sharpening filtering operations.

cv:: The parameters of the Laplacian function are as follows:

InputArray	src,	Input image
OutputArray	dst	Output edge image
int	ddepth	Output image depth
int	ksize = 1	The size of the kernel used to compute the second derivative
double	scale = 1	An optional scaling factor for computing the Laplacian, of type
double	delta = 0	An optional incremental value to add to the result before storing it in dst.
Int	borderType=BO RDER_DEFAULT	Pixel extrapolation method

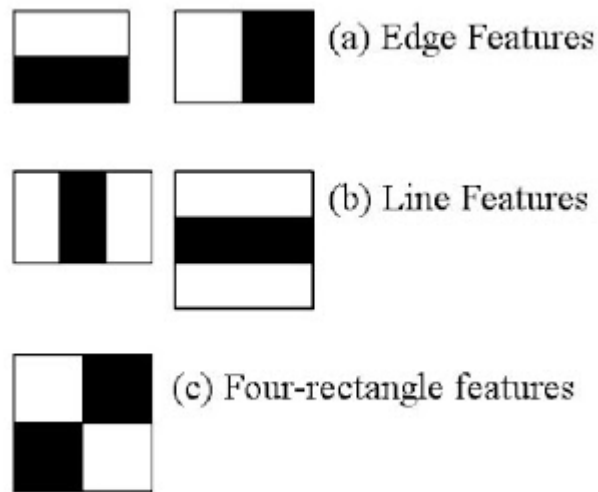
Chapter 4: Face Detection with OpenCV and Qt

4.1 OpenCV's Cascade Classifier

A classifier is a device that determines whether a certain object belongs to a specific category. A cascade classifier can be understood as a series of N single-class classifiers connected in sequence. If an object meets all the criteria of this series of connected classifiers, the final result is deemed to be positive; if any one criterion is not met, the result is deemed negative. For example, a face has many attributes, and we create a separate classifier for each attribute. If a model satisfies all the attributes we define for a face, we consider this model to be a face. So what do these attributes refer to? For example, a face needs to have two eyebrows, two eyes, a nose, a mouth, and a roughly U-shaped chin or contour, etc.

The goal detection based on Haar features using a cascade classifier is an effective object detection method proposed by Paul Viola and Michael Jones in their 2001 paper "Rapid Object Detection using a Boosted Cascade of Simple Features." This is a machine learning-based method where a cascade function is trained from many positive and negative images. It is then used to detect objects in other images.

We will perform face detection here. The algorithm first requires a large number of positive images (face images) and negative images (non-face images) to train the classifier. Then we need to extract features from them.



Haar features are divided into three categories: edge features, line features, center features, and diagonal features, which are combined into feature templates. The feature value of the template is defined as the sum of the white rectangle pixels minus the black rectangle pixels. The Haar feature value reflects the gray level changes in the image. For example, some features of the face can be simply described by rectangular features, such as: the eyes are darker than the cheeks, the sides of the nose are darker than the bridge of the nose, and the mouth is darker than the surrounding area, etc. However, rectangular features are only sensitive to some simple graphic structures, such as edges and line segments, so they can only describe specific orientations (horizontal, vertical, diagonal) of structures.

By changing the size and position of the feature template, a large number of features can be exhaustively generated in the image sub-window. The feature template in the above image is called a 'feature prototype'; the features obtained by expanding (translating and scaling) the feature prototype in the image sub-window are called 'rectangular features'; the value of the rectangular features is referred to as the 'feature value'.

Rectangular features can be located anywhere in the image, and their size can also be changed arbitrarily, so the rectangular feature value is a function of the three factors: the category of the rectangular template, the rectangular position, and the rectangular size.

Therefore, changes in category, size, and position result in a very large number of rectangular features within a very small detection window; for example, the number of rectangular features within a 24*24 pixel detection window can reach 160,000. This raises two questions that need to be addressed: (1) How to quickly compute so many features? (2) Which rectangular features are most effective for the classifier's classification?

4.2 Integral Image

To quickly compute features, we introduce the integral image, which reduces the computation for a given pixel to operations involving only four pixels, regardless of the size of the image. The integral image is a fast algorithm that allows the sum of pixel values in any region of the image to be computed by traversing the image only once, significantly improving the efficiency of image feature value calculations.

The main idea of the integral image is to store the sum of pixel values in rectangular regions formed from the starting point to each point as elements in an array. When calculating the sum of pixels in a specific region, one can directly index the elements of the array without recalculating the pixel sum for that region, thus speeding up the computation (this has a corresponding term known as dynamic programming algorithm). The integral image can compute different features at various scales using the same time (constant time), greatly enhancing detection speed.

The integral image is a matrix representation method that can describe global information. The construction method of the integral image is such that the value at position (i,j) is the sum of all pixels in the upper left direction from the original image at (i,j):

$$ii(i, j) = \sum_{k \leq i, l \leq j} f(k, l)$$

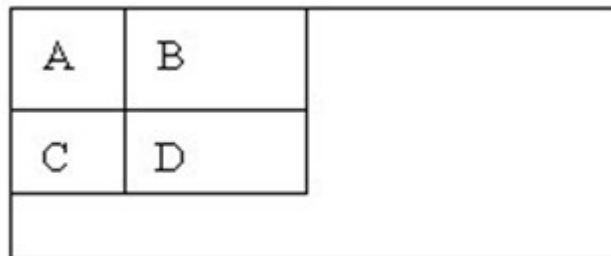
Integral Image Construction Algorithm

- ① Let $s(i, j)$ represent the cumulative sum in the row direction, initializing $s(i, -1) = 0$; ② Let $ii(i, j)$ represent an integral image, initializing $ii(-1, i) = 0$; ③ Scan the image row by row, recursively calculating the cumulative sum $s(i, j)$ in the row direction and the value of the integral image $ii(i, j)$ as $s(i, j) = s(i, j - 1) + f(i, j)$

$$ii(i, j) = ii(i-1, j) + s(i, j)$$

- ④ Scan the image once; when you reach the pixel at the bottom right corner of the image, the integral image ii is constructed.

Once the integral image is constructed, the sum of pixel values in any rectangular region of the image can be obtained through simple calculations as shown in the figure.



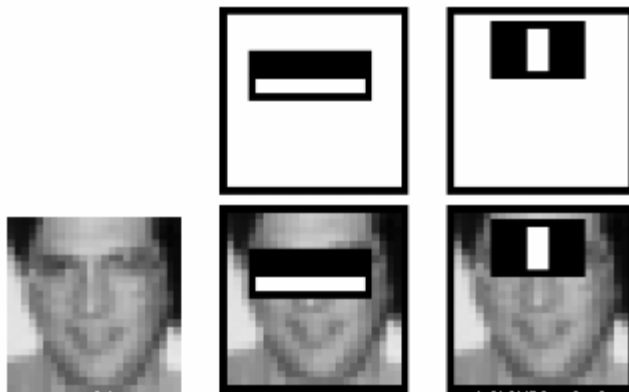
Let the four vertices of D be α , β , γ , and δ , then the sum of the pixels in D can be expressed

$$\text{as } D_{\text{sum}} = ii(\alpha) + ii(\beta) - (ii(\gamma) + ii(\delta));$$

The Haar-like feature value is nothing more than the difference between the sums of the pixel values of two matrices, which can also be completed in constant time. Therefore, the calculation of the feature value for rectangular features only depends on the integral image of the endpoints of this feature rectangle. Thus, regardless of how the scale of this feature rectangle changes, the time consumed for calculating the feature value is constant. As long as we traverse the image once, we can obtain the feature values for all sub-windows.

4.3 Selecting Features

Among all the features we have calculated, most are irrelevant. For example, consider the image below. The top row shows two good features. The first selected feature seems to focus on the property that the area around the eyes is usually darker than the nose and cheek areas. The second selected feature relies on the property that the eyes are darker than the bridge of the nose. However, applying the same window to the cheeks or any other area is ineffective. So how do we select the best features from over 160,000 features? This can be done through Implementing the AdaBoost algorithm.



We apply each feature to all training images. For each feature, it will find the best threshold to classify faces into positive and negative categories. Clearly, there will be correct or incorrect classifications. We choose the features with the smallest error rate, which means they are the most accurate in classifying face and non-face images. (The process is not as simple as this. Each image starts with an equal weight. After each classification, the weight of incorrectly classified images increases. Then the same process is repeated. The new error rate is calculated. There are also new weights. This process will continue until the desired accuracy or error rate is achieved or the required number of features is found.)

4.4 Introduction to Qt

Qt is a cross-platform C++ development library primarily used for developing graphical user interface (GUI) programs, but it can also be used to develop command line (CUI) programs without a graphical interface. Qt also has bindings for scripting languages such as Python, Ruby, and Perl, which means that programs based on Qt can be developed using scripting languages.

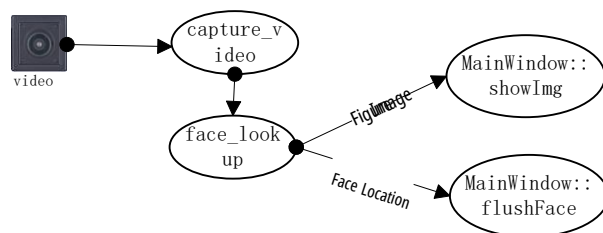
Qt supports many operating systems, such as general-purpose operating systems like Windows, Linux, and Unix, mobile operating systems like Android, iOS, and WinPhone, as well as embedded systems like QNX and VxWorks, among others. Although Qt is often regarded as a GUI library for developing graphical user interface applications, this is not all that Qt offers; in addition to creating beautiful interfaces (including controls, layouts, and interactions), Qt also includes many other features, such as multithreading, database access, image processing, audio and video processing, network communication, and file operations, all of which are integrated into Qt.

Although Qt also supports mobile operating systems, its market share in the mobile sector is almost negligible because Android already has Java and Kotlin, and iOS already has Objective-C and Swift.

4.5 Experimental Objectives

- ① Using the Qt library to display video images and face bounding boxes
- ② Using the CascadeClassifier class from the OpenCV library

4.6 Software flowchart

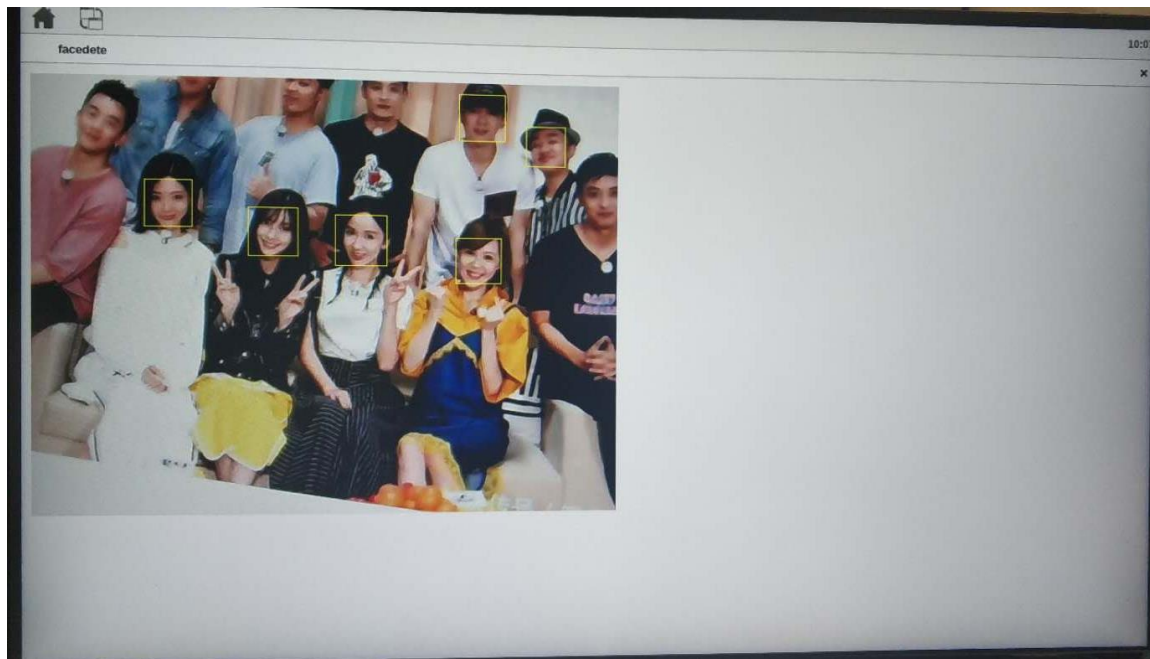


4.7 Preparation for Running

- (1) Prepare the work as per the previous chapter
- (2) Copy the haar_train folder to the development board's /home/root directory

4.8 Program Execution

- (1) Run the application, and the effect is as follows



4.9 Code Analysis

- (1) The detection of faces is mainly done in the face_lookup.cpp file using face_cascade.detectMultiScale. During initialization, face_cascade.load is called to load the trained model haarcascade_frontalface_alt.xml.
- (2) In MainWindow::flushFace, the face bounding box will be displayed based on the detection results mentioned above.
- (3) Due to the model training requirements, the detection needs the face to be neither tilted nor in profile.
- (4) .pro file

Because the Q display interface is called, in the facedete.pro file, I added several Qt components after 'QT=' including core, gui, and widgets.

- (5) Qt's signals and slots

The program binds the signals and slots of face_lookup and MainWindow, allowing MainWindow to handle the images that need to be displayed and mark the positions of faces.

- (6) Similar to the above example, the display effect is relatively choppy.

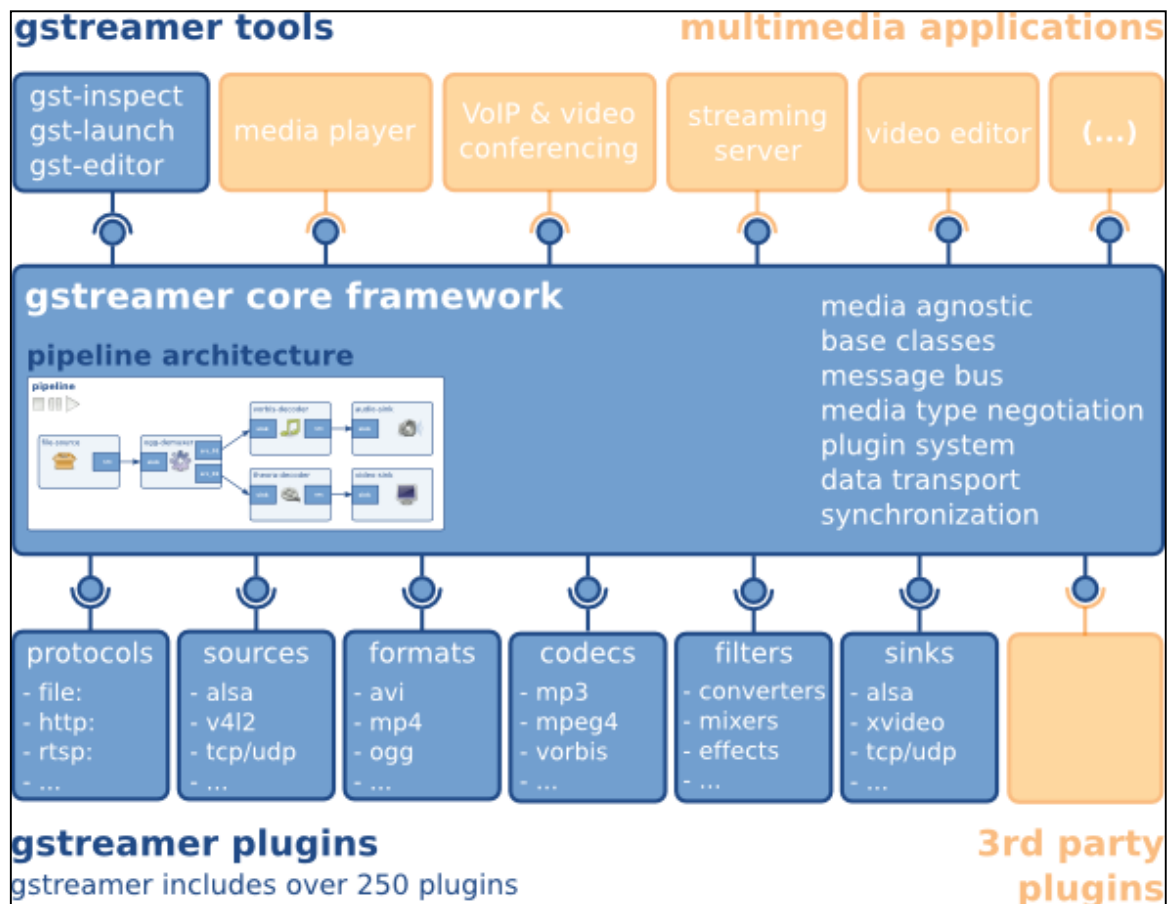
Chapter Five: GStreamer Camera Display

5.1 Introduction to GStreamer

GStreamer is a framework for creating streaming media applications. Its basic design philosophy is inspired by the ideas of the Oregon Graduate Institute regarding video pipelines, and it also draws on the design concepts of DirectShow. Its features are as follows:

- ① Clear structure, powerful functionality
- ② Object-oriented programming philosophy
- ③ Flexible and extensible features
- ④ High performance
- ⑤ Separation of core libraries and plugins

The architecture of GStreamer is as follows, supporting applications ranging from simple Ogg/Vorbis playback and audio/video streaming to complex audio (mixing) and video (non-linear editing) processing. Applications can transparently utilize advanced codec and filtering technologies.



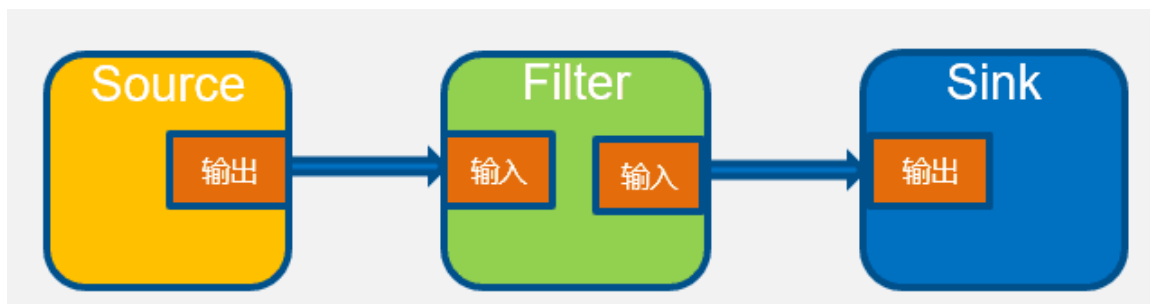
5.2 Basic Concepts of GStreamer

(1) Element

Elements are the most basic components that make up a pipeline. Several elements can be connected together to create a pipeline to accomplish a specific task, such as media playback or recording. Elements are classified by function:

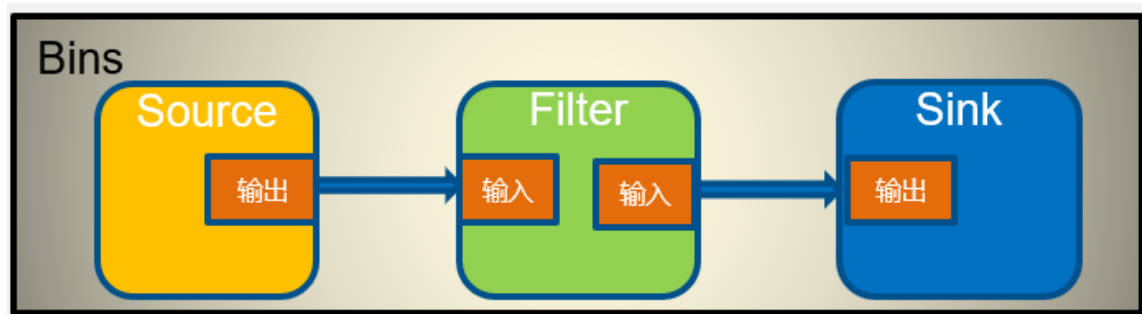
- ① Source elements
- ② Sink elements
- ③ Filter elements

The application block diagram is as follows:



(2) Bin and Pipeline

Bins are containers that can hold elements. Pipelines are a special subclass of bins; pipelines can operate on all elements contained within themselves. As shown in the figure below.



(3) Pads

Pads serve as interfaces for elements, allowing various elements to connect through this interface, enabling data flow between them. Pads are classified as follows:

- ① Always
- ② Sometimes
- ③ On request

(4) Function (Cap)

Caps describe the data flow format that can be processed through pads or the current data flow format through pads. The media types supported by the element can be viewed using the `gst-inspect-1.0` command.

(5) Bus

Each pipeline contains a default bus, and the application does not need to create a bus. The application sets up on the bus.

A message handler. When the main loop is running, the bus will poll this message handler for new messages. When a message is collected, the bus will call the corresponding callback function to complete the task.

(6) Buffer

The buffer contains the data stream in the created pipeline. Typically, a source element will create a new buffer, and the element will pass the buffer's data to the next element. When creating a media pipeline using GStreamer low-level constructs, there is no need to handle the buffer manually; the elements will automatically manage these buffers. The buffer is primarily composed of the following components:

- ① A pointer to a certain block of memory②

The size of the memory

- ③ The timestamp of the buffer

- ④ A reference count that indicates the number of components using the buffer. When there are no components to reference, this reference will be used to destroy the buffer

(7) Events

The events include control information in the pipeline, such as information seeking and flow termination signals.

5.3 Experimental Objectives

- ① Use of gstreamer tools②

gstreamer programming

5.4 Common tools for Gstreamer

(1) gst-inspect-1.0

Without parameters, it will list all available elements, which are all the elements you can use. If a filename is provided, it will attempt to open that file as a GStreamer plugin and list all the internal elements. If a GStreamer element is provided, it will list all the information about that element.

Print the list of plugins and their information, such as:

Print the list of supported plugins	gst-inspect-1.0 -a
Print the information of the filesrc plugin	gst-inspect-1.0 filesrc

Pad Templates: This section will list all types of Pads and their Caps. Through these, you can confirm whether you can connect to a certain element.

(2) gst-discoverer

This tool can be used to view the Caps contained within a file; it is a wrapper around the GstDiscoverer object. It accepts a URI input from the command line and prints out all the information. This is useful for examining how media is encoded and multiplexed, allowing us to determine what element to place in the pipeline.

(3) gst-launch-1.0

This tool can create a pipeline, initialize it, and run it. It allows you to quickly test whether it works before formally writing code to implement the pipeline. If you need to view the generated caps of an element in the pipeline,


```
command: gst-launch-1.0 -v v4l2src device=/dev/video0
```

```

root@petalinux:~# gst-launch-1.0 -v v4l2src device=/dev/video0
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
/GstPipeline:pipeline0/GstV4l2Src:v4l2src0.GstPad:src: caps = video/x-raw, format=(string)YUY2, width=(int)2592, height=(int)1944, pixel-aspect-ratio=(fraction)1/1, interlace-mode=(string)progressive, framerate=(fraction)25/1, colorimetry=(string)bt709
ERROR: from element /GstPipeline:pipeline0/GstV4l2Src:v4l2src0: Intern[ 3714.942617] xhci-hcd xhci-hcd.0.auto: ERROR unknown event type 37
al data stream error.
Additional debug info:
../.../.../git/libs/gst/base/gstbasesrc.c(3055): gst_base_src_loop (): /GstPipeline:pipeline0/GstV4l2Src:v4l2src0:
streaming stopped, reason not-linked (-1)
Execution ended after 0:00:00.435641850
Setting pipeline to PAUSED ...
Setting pipeline to READY ...
Setting pipeline to NULL ...
Freeing pipeline ...
root@petalinux:~#

```

```
root@petalinux:~# ./uvc_fmt 0
query/dev/video0 format
1.YUYV 4:2:2
1920x1080@60
640x480@60
800x600@60
1024x768@60
1600x1200@25
2048x1536@25
2592x1944@25
root@petalinux:~#
```



5.6 Running Programs

Through commands, we can quickly verify various scenarios we envision, but if we want more state monitoring or user control, this needs to be implemented through programming.

The examples in this chapter are implemented through programs to achieve the effects of the above commands.

- (1) It is important to note that the considerations are the same as above, and the resolution and format supported by the USB camera need to be modified.
- (2) The program monitors and handles various state changes by adding a message event callback function called `sink_message`.
- (3) After running the program, you can see that the displayed image is the same as the effect of running the command. At this point, if you unplug the camera, the information printed by the program is as follows, and the program exits.

```
get error: Could not read from resource.
play error
playing out
Application finished with exit code 0.
```

5.7 Code Analysis

(1) `gst_parse_launch`

This function creates a pipeline, and the function parameter is as follows:

<code>const gchar *</code>	<code>pipeline_description</code>	Describe the pipeline
<code>GError **</code>	<code>error</code>	Error return

The content of the `pipeline_description` is almost the same as what was discussed earlier regarding `gst-launch-1.0` verification.

(2) `gst_bus_add_watch`

Adds a callback function to the bus to handle necessary messages.

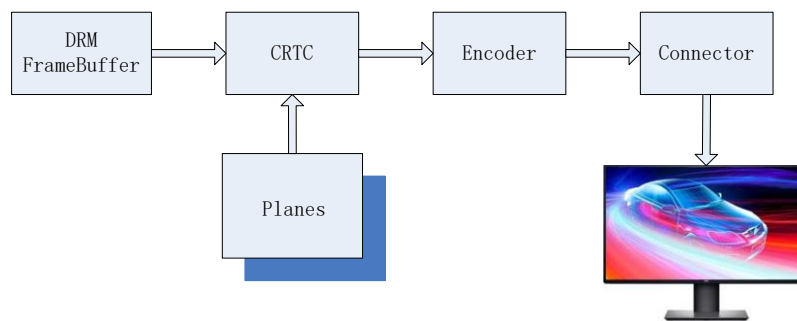
Chapter Six: Camera Display with Qt, DRM, and Gstreamer

6.1 Introduction to DRM

DRM, which stands for Direct Rendering Manager, is designed to address the issue of multiple programs sharing resources of the Video Card. It provides a set of APIs to user space for output management.

DRM has now encompassed many functions that were previously handled by user space programs, such as framebuffer management and mode setting, memory shared objects, and memory synchronization. Some of these extensions have specific names, such as Graphics Execution Manager (GEM) or Kernel Mode Setting (KMS), all of which belong to the DRM subsystem.

DRM display mainly involves the following five modules



(1) DRM Framebuffer

From a developer's perspective, a FrameBuffer is a block of memory, and writing data in a specific format to this memory means outputting content to the screen. Therefore, the FrameBuffer is like a canvas. For example, in a FrameBuffer initialized for 16-bit color, two bytes in the FrameBuffer represent a single point on the screen, with a linear relationship between screen positions and memory addresses from top to bottom and left to right.

(2) CRTC

The task of the CRTC is to read the image to be displayed from the framebuffer and output it to the encoder in the corresponding format. Although CRTC literally means Cathode Ray Tube Controller, CRTs have long been obsolete in ordinary display devices. In DRI, the main functions of the CRTC are as follows:

- ① Configure the resolution suitable for the display (kernel) and output the corresponding timing (hardware logic)
- ② Scan the framebuffer to display on one or more display devices

(3) Planes

With the continuous updates in software technology, the performance requirements for hardware are becoming increasingly high. Under the premise of meeting normal functional use, the demands for power consumption are also becoming more stringent. Originally, the GPU could handle all graphical tasks, but due to its excessively high power consumption during operation, designers decided to delegate some simple tasks to the Display Controller (such as composition) while allowing the GPU to focus on its primary task of drawing (i.e., rendering), thereby reducing the burden on the GPU and achieving the goal of lowering power consumption and enhancing performance. Thus, the Plane (hardware layer unit) was born. Planes mainly output multiple layers stacked together, and some hardware supports layer translation, scaling, cropping, alpha channels, and so on.

(4) Encoder

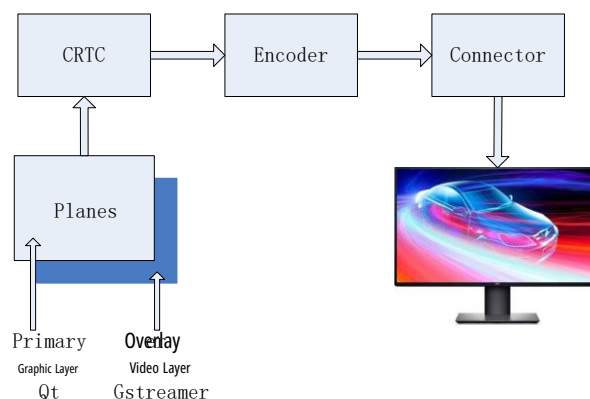
Encodes image signals in a certain format (such as RGB, YUV, etc.) into the signals that the connector needs to output. Taking HDMI as an example, frame/line synchronization/display content is output through the serial bus of TMDS data, so the task of the Encoder is to encode the parallel timing into a serial sequence according to the HDMI standard.

(5) Connector

A connector is essentially a physical interface that connects to a display, commonly including VGA, HDMI, DVI, DP, etc., and can read the parameters supported by the display through the connector;

6.2 Experiment Introduction

The DP output of MPSoc supports two layers, with the upper layer (Primary) supporting the alpha channel. In this chapter's experiment, we use Qt to output content to the Primary layer, while Gstreamer captures video data to the lower layer (Overlay). Ultimately, these two layers are mixed and output by the hardware. The working block diagram is as follows:



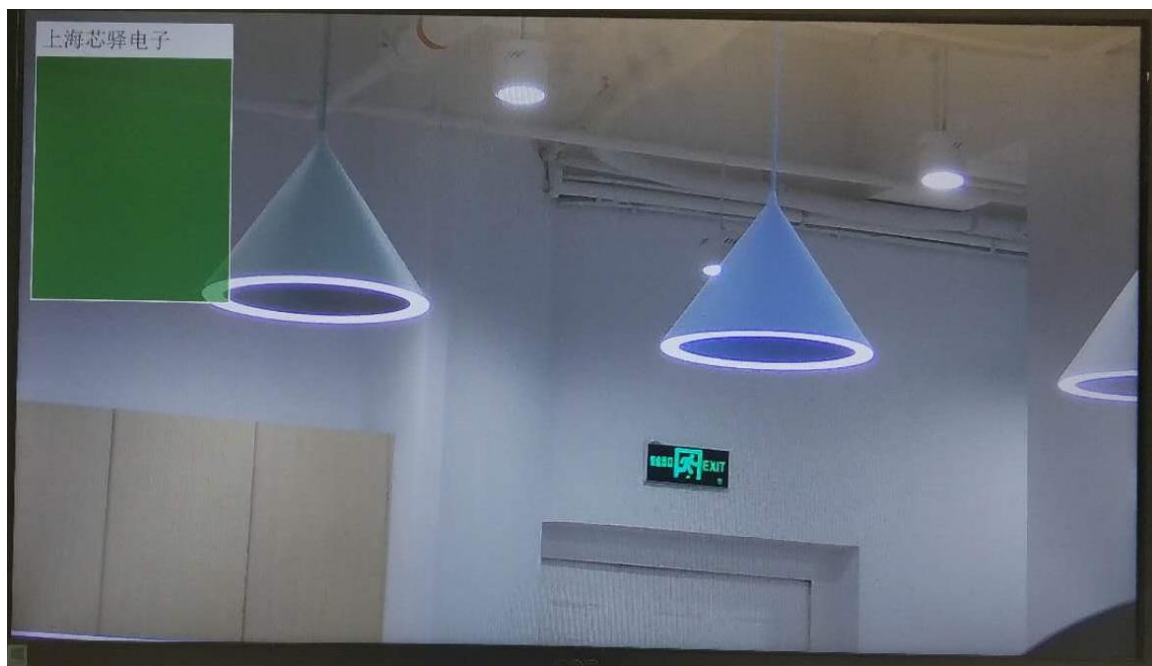
6.3 Experiment Objectives

- ① DRM Output Management
- ② DRM Layer Management
- ③ Layered Display of Qt and GStreamer

6.4 Run Program

- (1) It should be noted that this experiment requires the USB camera to support 1080 resolution (referring to the raw stream, such as the YUYV422 format mentioned above).
- (2) Experimental running effect.

It can be seen that a Qt interface has been overlaid on the video.



6.5 Code Analysis

(1) Check if the monitor is connected

In the main function, call `init_drm` to determine if the initialization is successful. If it returns -2, indicating no supported display format, it means the monitor is not connected.

(2) Set the display output resolution

Through the function `drmModeSetCrtc`, set the output resolution and refresh rate. Here, the buffer is not set, so the third parameter is set to -1.

(3) Set the transparency attribute of the graphic layer.

The graphic layer supports both global and per-pixel alpha value settings. Here, it is set to per-pixel, meaning `g_alpha_en` is set to 0. Set to 0.

(4) Qt Display

Qt defaults to displaying on the Primary layer

(5) GStreamer Display

The plane information, including the plane ID number, can be obtained through `drmModeGetPlaneResources`. Here, we obtain

The ID of the Overlay layer is 35. Through the code, we can see that the properties of `kmssink` are set as follows: `kmssink sink-type=dp bus-id=fd4a0000.zynqmp-display fullscreen-overlay=false plane-id=35`

Chapter Seven: Camera Display with Qt and GPU

7.1 Introduction to Mali GPU

GPU stands for Graphics Processing Unit, which translates to '图形处理器' in Chinese. We usually refer to it as a graphics card. There are many manufacturers of GPUs, with the three largest being Intel, NVIDIA, and AMD, primarily used in the PC domain.

ARM CPUs occupy 90% of the mobile market, but its Mali GPUs can only be considered a niche presence in the mobile sector. The Mali GPU was originally developed by the Falanx company, which was spun off from a project at the Norwegian University of Science and Technology. It was acquired by ARM in 2006, becoming a GPU division of ARM, and in 2007, Mali was released as part of ARM with the mali-200 GPU.

The Mali GPU has gone through three generations of architecture as follows:

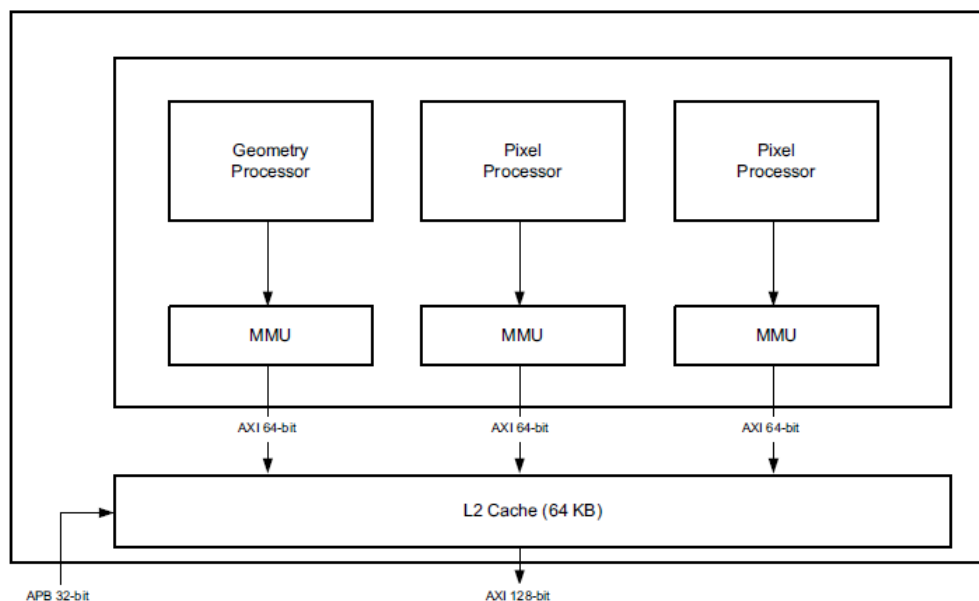
First generation: Utgard	Second generation: Midgard	Third Generation Bifrost
Mali-200/300/400/450/470	Mali-T600/T700/T800 Series	Mali-G3/G5/G7 Series

Divided by performance:

Low power	Medium	High performance
Mali-400/450/470 Mali-G31	Mali-G51/G52 Mali-T820/T830 Mali-T720	Mali-G71/72 Mali-T860/T880 Mali-T760

7.2 MPSoc GPU Introduction

The GPU used here is a 2D and 3D acceleration subsystem based on Mali™-400 MP2 hardware. Its component resource block diagram is as follows:



The supported features are as follows:

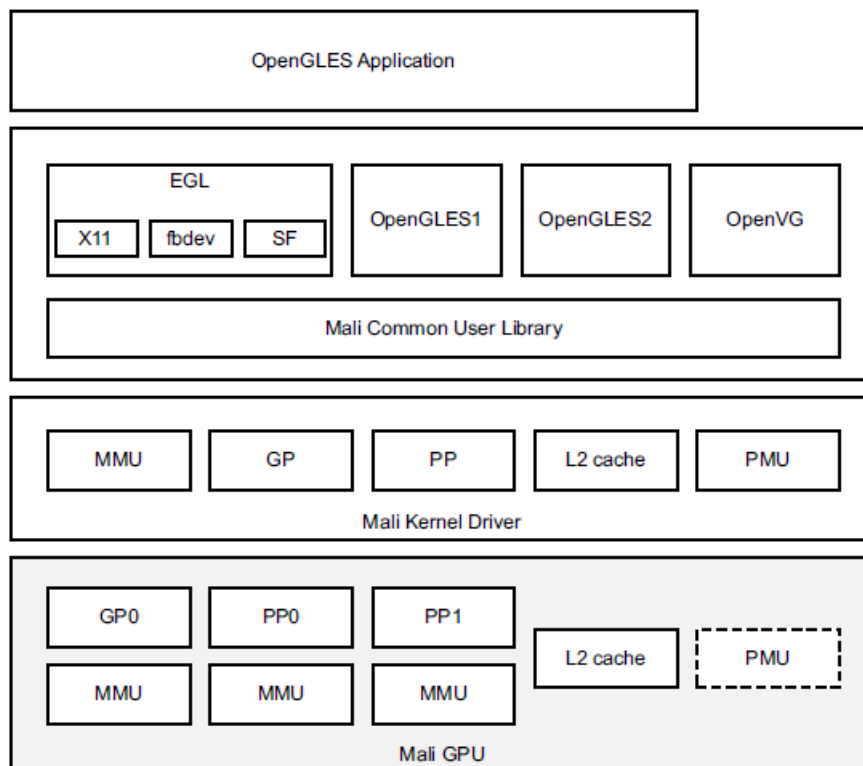
Supports OpenGL ES 1.1 and 2.0

Supports OpenVG 1.1

The software architecture diagram for Linux systems supports

32-bit floating-point operations in accordance with IEEE standards,

allowing for texture sizes of up to 4096 x 4096 pixels.



At a working frequency of 400M, the peak performance it can achieve is as follows:

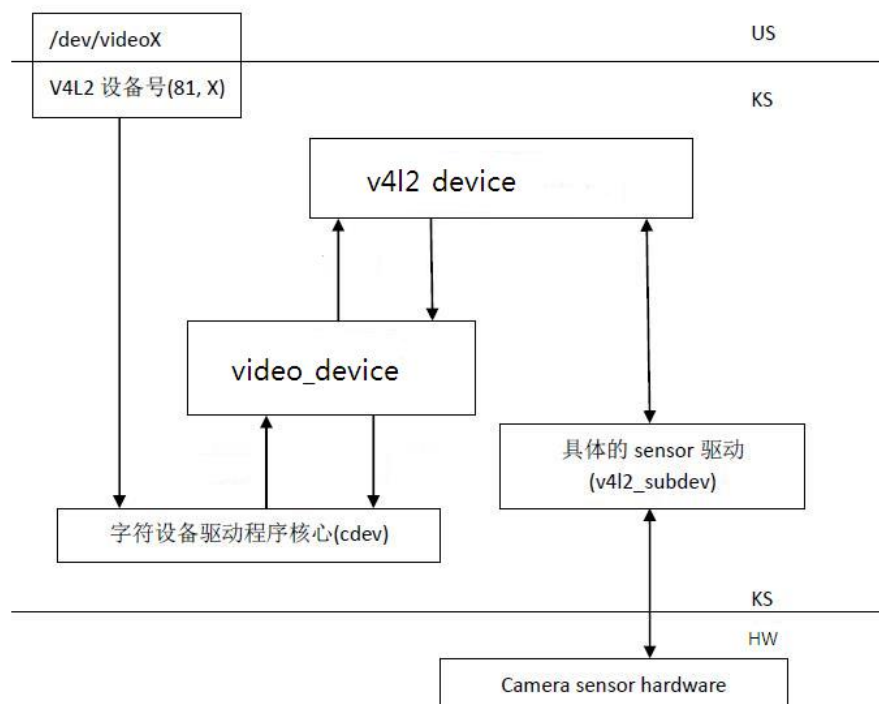
Pixel fill rate: 800 Mpixel/sec, vertex

processing speed: 40 Mvertex/sec.

7.3 V4L2 Introduction

V4L2 stands for Video For Linux Two, which is a unified interface provided by the kernel for applications to access audio and video drivers. One of its most basic applications is to obtain video data from devices such as cameras. The device name is /dev/video, with a major device number of 81 and minor device numbers ranging from 0 to 63.

The structure diagram in the Linux system is as follows:

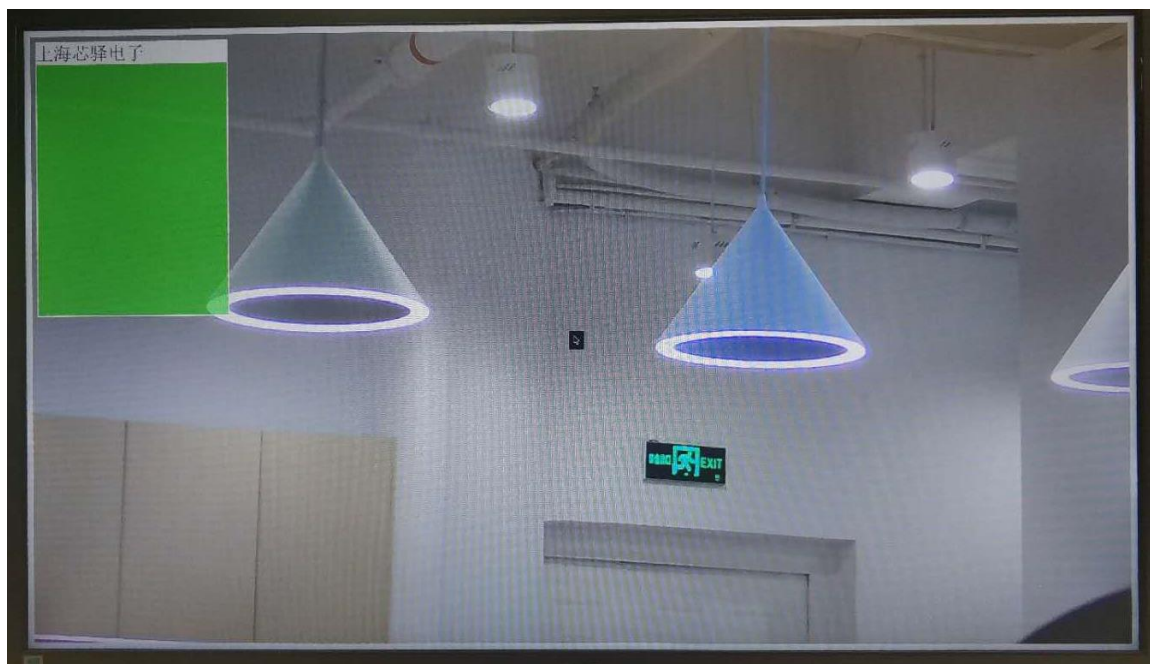


7.4 Experiment Objectives

- ① OpenGL operations
- ② V4L2 operations

7.5 Run the program

The program runs as follows. You can see that, compared to the display effect in the previous chapter, the image here is not displayed in full screen, leaving a few pixels of white border.



7.6 Code Analysis

The display processing part is mainly encapsulated in the class uOpenglYuv. Here, we only demonstrate how to display images in YUYV422 format.

(1) How to obtain a pixel in the string

fragmentStr for YUYV422:

Texture Y retrieves the Y value `yuv.x = texture2D(tex_y, textureOut).x;`

Texture UV retrieves the U value `yuv.y = texture2D(tex_uv, textureOut).y;`

Texture UV retrieves the V value `yuv.z = texture2D(tex_uv, textureOut).w;`

Other operations convert yuv to rgb.

(2) Image Refresh

In the MainWindow, after receiving a frame of image, `pOpenglYuv->update()` is called, and then `uOpenglYuv` calls `paintGL` to load the image as a texture and display it.

(3) Camera Capture

The camera capture is completed by calling the `v4l2` interface.

Chapter Eight: Linux Register Operations

8.1 Linux Memory Mapping

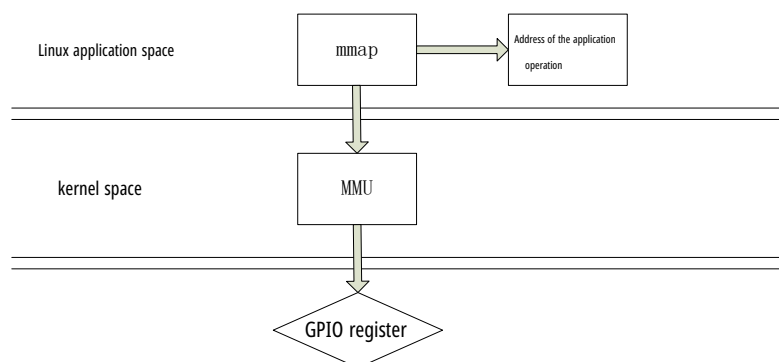
Memory mapping refers to mapping a segment of memory in user space to kernel space. Once the mapping is successful, modifications made by the user to this memory area can directly reflect in kernel space, and similarly, modifications made by kernel space to this area can directly reflect in user space. This is highly efficient for transferring large amounts of data between kernel space and user space.

For example, when reading a file from the hard disk into memory, it must go through the file system for data copying, and the data copying operation is implemented by the file system and hardware drivers. Theoretically, the efficiency of copying data is the same. However, accessing files on the hard disk through memory mapping is more efficient than using read and write system calls. Why is this? The reason is that `read()` is a system call that involves data copying. It first copies the file content from the hard disk to a buffer in kernel space, and then copies this data to user space, resulting in two data copies in this process. On the other hand, `mmap()` is also a system call, but it does not perform data copying. The actual data copying occurs during page fault handling. Since `mmap()` directly maps the file to user space, the interrupt handler can directly copy the file from the hard disk to user space based on this mapping relationship, resulting in only one data copy. Therefore, the efficiency of memory mapping is higher than that of read/write.

8.2 Introduction to the Experiment

Sometimes, the functionality of a peripheral has already been verified under bare metal. If this peripheral does not use interrupts, we can directly port the bare metal program to Linux by mapping registers, thus saving the effort of developing a driver.

This example has ported the GPIO example from Vitis, driving the PS_LED to blink once per second. Other operations follow a similar process.



8.3 Run the program

After running the application `register_opt`, you can see that the LED light of the PS blinks once every second

8.4 Code Analysis

- (1) Regarding the register addresses of GPIO, these are all copied from the Vitis GPIO example. Our development approach should also be like this, first verifying with Vitis's bare-metal program. For many peripherals and FPGA-side IPs, Vitis will help us generate the necessary operation methods and addresses, so we don't need to find the corresponding relationships ourselves.

- (2) When opening /dev/mem, use the O_SYNC option.

After writing data to the external device, the data is usually written to the cache buffer. O_SYNC will ensure that the data is written to the peripheral before returning. It is important to note that O_SYNC here only affects write operations.

- (3) The call of msync

If a large amount of data needs to be written to a device at once, calling O_SYNC will severely impact system performance. In this case, if we do not use O_SYNC but instead call msync after writing the data, it will improve the writing performance.

- (4) Read operation consistency issues

When reading data from a device, due to the presence of cache, the data obtained in the application may be from the cache rather than the latest state of the device, which may result in reading an incorrect value.