

ALINX 黑金开发板配套教程

Ultrascale+ Linux 应用程序开发

2020/11/16 V1.01



版权声明

Copyright ©2012-2019 芯驿电子科技（上海）有限公司

公司网址:

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

技术论坛:

<http://www.heijin.org>

天猫旗舰店:

<https://alinx.tmall.com>

京东旗舰店:

<http://alinx.jd.com>

邮箱:

avic@alinx.com.cn

电话:

021-67676997

传真:

021-37737073

ALINX 微信公众号:



目录

版权声明	2
第一章 极简工作环境搭建	5
1.1 系统环境搭建简介	5
1.2 系统及基本工具安装	5
1.3 编译环境安装	5
1.4 QtCreator 配置	6
第二章 可远程调试的 Hello World	11
2.1 GDB 简介	11
2.2 QtCreator 简介	11
2.3 实验目标	11
2.4 Qt Creator 工程	12
2.5 获取板卡的 IP 地址	12
2.6 QtCreator 配置 GDB 连接板卡	12
2.7 运行 Hello World	15
2.8 运行参数设置	16
第三章 OpenCV 之边缘检测	19
3.1 OpenCV 简介	19
3.2 边缘检测简介	19
3.3 实验目标	20
3.4 软件流程图	20
3.5 运行准备	20
3.6 程序运行	21
3.7 代码解析	21
第四章 OpenCV+Qt 之人脸检测	24
4.1 OpenCV 的级联分类器	24
4.2 积分图	25
4.3 选取特征	26
4.4 Qt 简介	26
4.5 实验目标	27
4.6 软件流程图	27
4.7 运行准备	27
4.8 程序运行	27
4.9 代码解析	28
第五章 GStreamer 的摄像头显示	29
5.1 GStreamer 简介	29
5.2 GStreamer 基本概念	30
5.3 实验目标	31

5.4 Gstreamer 常用工具	31
5.5 gst-launch-1.0 摄像头显示	32
5.6 编程运行程序	33
5.7 代码解析	33
第六章 Qt+DRM+Gstreamer 的摄像头显示	34
6.1 DRM 简介	34
6.2 实验介绍	35
6.3 实验目标	35
6.4 运行程序	35
6.5 代码分析	36
第七章 linux 寄存器操作	37
7.1 linux 内存映射	37
7.2 实验简介	37
7.3 运行程序	37
7.4 代码分析	38

第一章 极简工作环境搭建

1.1 系统环境搭建简介

本教程在 ubuntu 下使用 QtCreator 作为开发工具, 来交叉编译应用程序。所以在 ubuntu 下, 我们需要安装交叉编译工具以及各种需要用到的库文件, 这些可使用 sdk.sh 安装包来完成, 安装包的由来, 可以参考 petalinux 章节的教程, 这里不作介绍。

1.2 系统及基本工具安装

- (1) 在 win10 系统下, 安装虚拟机软件 VMware
可参考《course_s0_Xilinx 开发环境安装教程.pdf》第 2.1 章节
- (2) 使用 ubuntu-18.04.2-desktop-amd64.iso 创建 linux 系统
可参考《course_s0_Xilinx 开发环境安装教程.pdf》第 2.2 章节
- (3) ubuntu 下安装必要的库

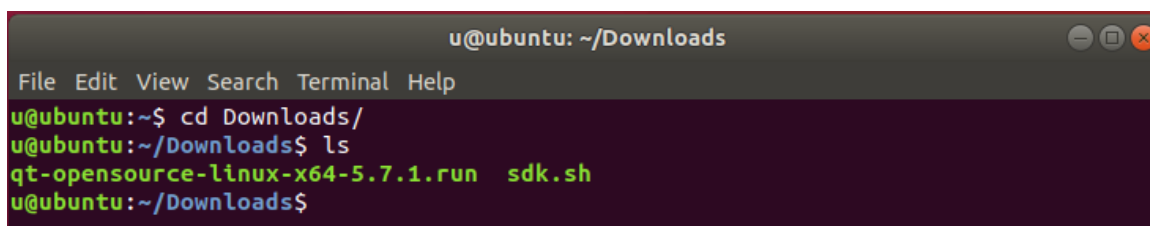
```
sudo apt-get install build-essential
```

可参考《course_s0_Xilinx 开发环境安装教程.pdf》第 4.2 章节

- (4) ubuntu 下安装 QtCreator
可参考《course_s0_Xilinx 开发环境安装教程.pdf》第 6.1 章节

1.3 编译环境安装

- (1) 将 sdk.sh 拷贝至 ubuntu 系统下, 并打开终端

A terminal window titled 'u@ubuntu: ~/Downloads' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
u@ubuntu:~$ cd Downloads/  
u@ubuntu:~/Downloads$ ls  
qt-opensource-linux-x64-5.7.1.run  sdk.sh  
u@ubuntu:~/Downloads$
```

- (2) 在根目录下创建 tools 目录并修改权限

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

- (3) 运行 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) 输入安装目录，这里安装目录设置为“/tools/xilinx_sdk_tool”，并回车

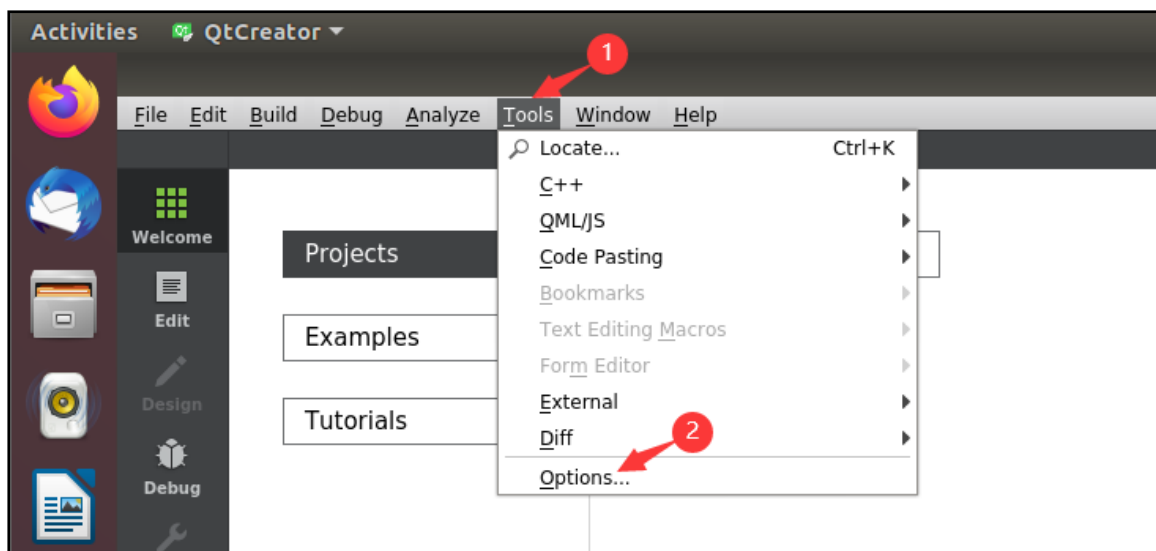
```
u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$ ./sdk.sh
PetaLinux SDK installer version 2019.2
=====
Enter target directory for SDK (default: /opt/petalinux/2019.2): /tools/xilinx_s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

- (5) 输入“Y”开始安装，直至结束

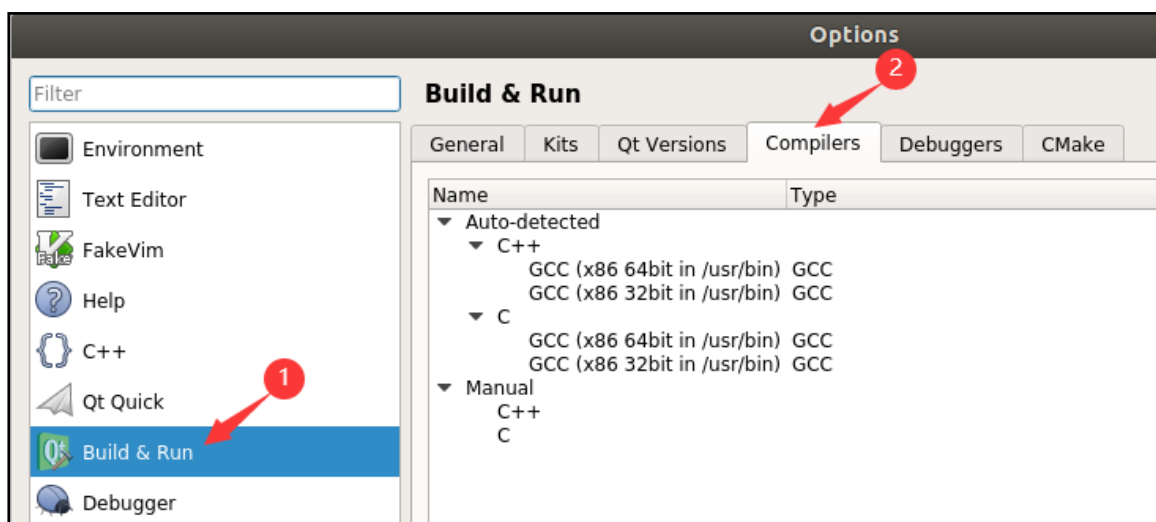
```
u@ubuntu: ~/Downloads
File Edit View Search Terminal Help
u@ubuntu:~$ cd Downloads/
u@ubuntu:~/Downloads$ ls
qt-opensource-linux-x64-5.7.1.run  sdk.sh
u@ubuntu:~/Downloads$ ./sdk.sh
PetaLinux SDK installer version 2019.2
=====
Enter target directory for SDK (default: /opt/petalinux/2019.2): /tools/xilinx_s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

1.4 QtCreator 配置

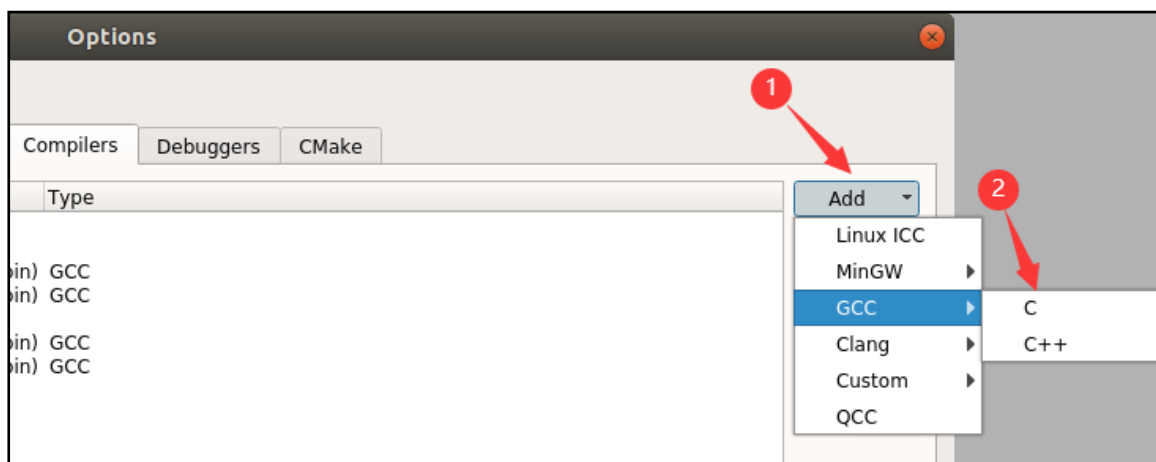
- (1) 打开 QtCreator 工具设置项



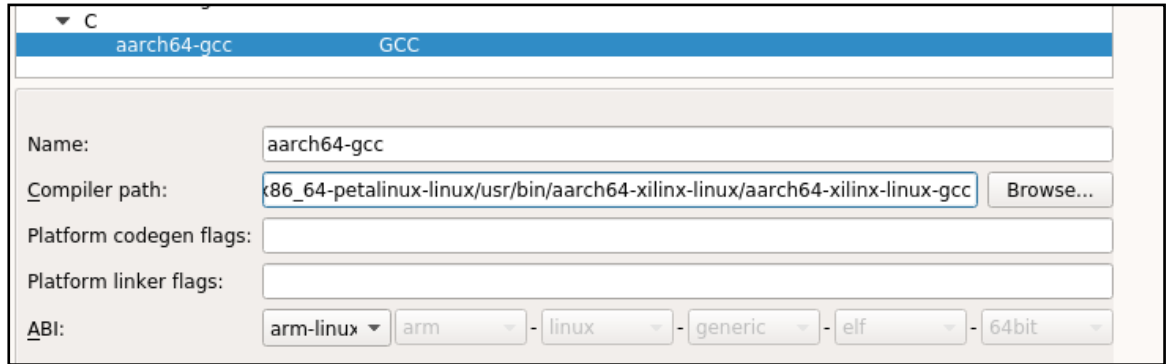
(2) 打开编译工具设置项



(3) 添加 C 编译工具



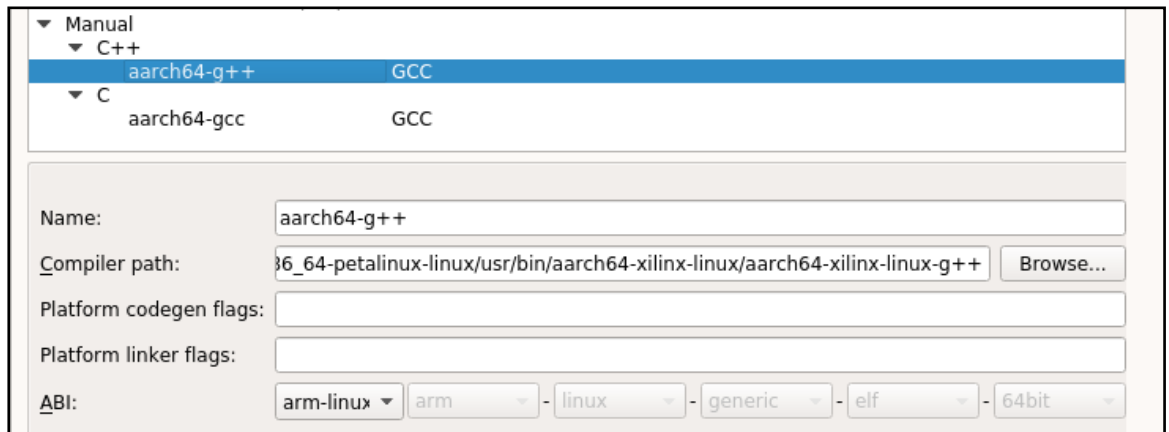
(4) 设置名称及路径



其中路径为

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

(5) 与上面步骤相同，添加 C++编译工具

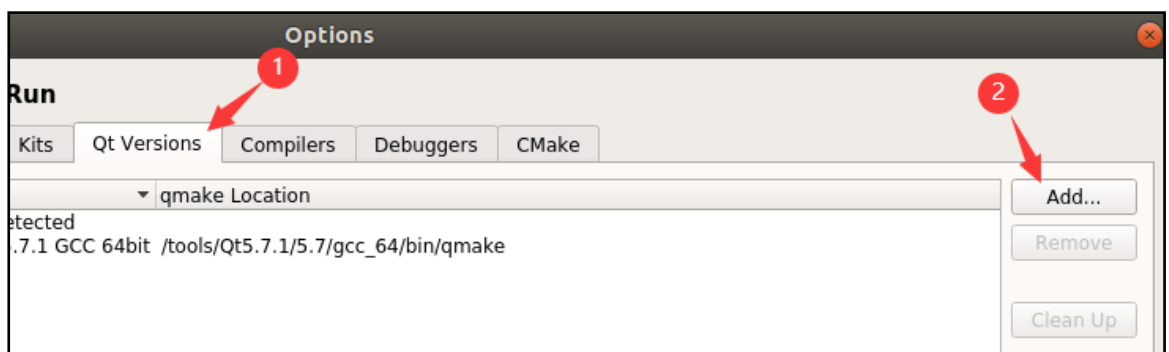


其中路径为

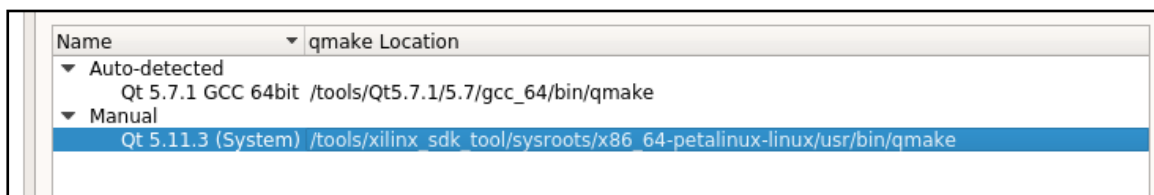
"/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-g++"

(6) 点击按钮 "Apply" 完成此项设置

(7) 切换至 "Qt Versions"，点击添加按钮



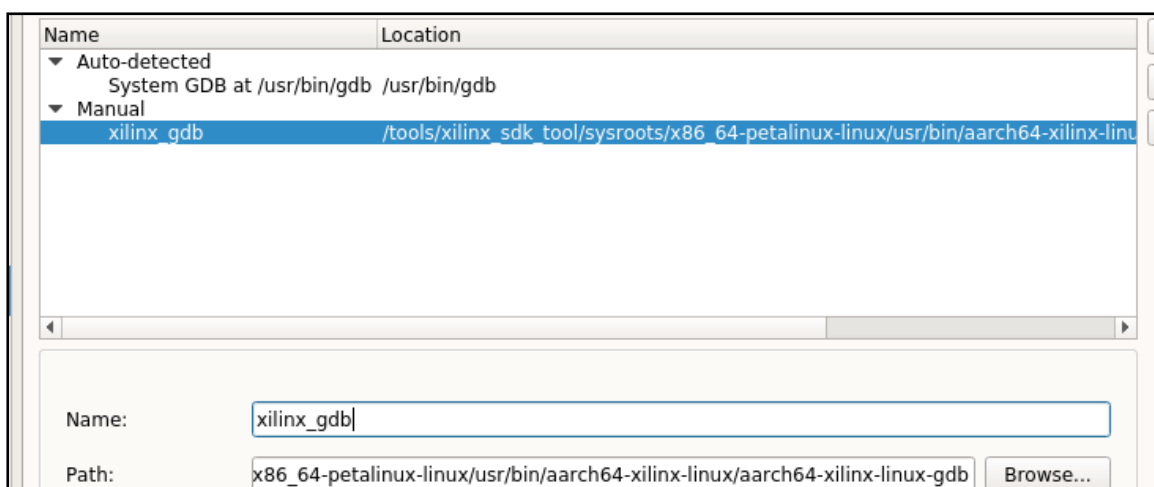
(8) 选择 qmake



路径为 “/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/qmake”

(9) 点击按钮 “Apply” 完成此项设置

(10) 切换至“Debuggers”项，并点击 “Add”按钮

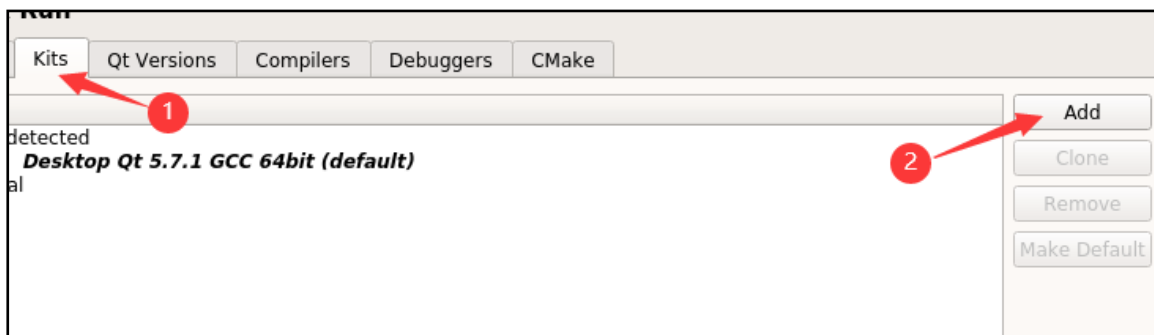


添加项修改内容如图，其中路径

为“/tools/xilinx_sdk_tool/sysroots/x86_64-petalinux-linux/usr/bin/aarch64-xilinx-linux/aarch64-xilinx-linux-gdb”

(11) 点击按钮 “Apply” 完成此项设置

(12) 切换至“Kits”项，并点击 “Add”按钮



(13) 新添加项各项内容修改如图

The screenshot shows the 'Manual' configuration window in the ALINX IDE. The project name is 'xilinx_sdk'. The device type is set to 'Generic Linux Device'. The compiler is 'aarch64-gcc' and the C++ compiler is 'aarch64-g++'. The environment is set to 'No changes to apply'. The debugger is 'xilinx_gdb' and the Qt version is 'Qt 5.11.3 (System)'. The CMake generator is 'CodeBlocks - Unix Makefiles, Platform: <none>, Toolset: <none>'. On the right side, there are buttons for 'Ren', 'Make', 'Mana', 'Brow', 'Mana', 'Chan', 'Mana', and 'Char'.

Manual	xilinx_sdk (default)	Ren
Name:	xilinx_sdk	Make
File system name:		
Device type:	Generic Linux Device	
Device:		Mana
Sysroot:		Brow
Compiler:	C: aarch64-gcc	Mana
	C++: aarch64-g++	
Environment:	No changes to apply.	Chan
Debugger:	xilinx_gdb	Mana
Qt version:	Qt 5.11.3 (System)	Mana
Qt mkspec:		
CMake Tool:		Mana
CMake generator:	CodeBlocks - Unix Makefiles, Platform: <none>, Toolset: <none>	Char

(14) 点击按钮 “OK” 完成设置

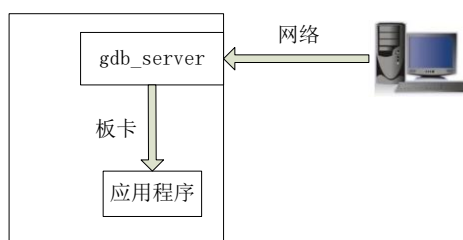
第二章 可远程调试的 Hello World

2.1 GDB 简介

GDB, the GNU Project debugger, 允许您查看程序在执行时“内部”发生了什么，或者一个程序在崩溃时正在做什么。主要功能如下：

- ① 启动程序，指定任何可能影响其行为的内容，如环境变量，运行参数等
- ② 使程序在指定条件下停止运行
- ③ 检查程序停止时发生了什么，如寄存器状态，变量值等
- ④ 手动修改程序运行中的各种值，这样可以更方便的验证程序功能

Linux 下的 GNU 调试工具是 gdb、gdbserver。其中 gdb 和 gdbserver 可完成对目标板上 Linux 下应用程序的远程调试。gdbserver 是一个很小的应用程序，运行于目标板上，可监控被调试进程的运行，可通过网络与上位机上的 gdb 通信。开发者可以通过上位机的 gdb 输入命令，控制目标板上进程的运行，查看内存和寄存器的内容



2.2 QtCreator 简介

Qt Creator 是一个跨平台的集成开发环境 (IDE)，允许开发人员跨桌面、移动和嵌入式平台创建应用程序。

我们通常说的 Qt 库，则是我们开发桌面等应用而调用的库，它将会被链接进我们的运行程序。而 Qt Creator 是一个工具，帮助我们开发应用程序的工具。所以，两者可以认为是完全不同的东西。

2.3 实验目标

- ① 认识 Qt Creator 工程
- ② QtCreator 配置 GDB 连接板卡
- ③ 将可运行的目标程序搬移至板卡
- ④ 给运行程序传递参数

2.4 Qt Creator 工程

Qt Creator 的工程文件是 xxx.pro，文件中“QT=”指定需要用到的 QT 库组件，如果为空，则表示不调用 QT 库。

2.5 获取板卡的 IP 地址

- (1) 将我们的开发板，装好有 linux 系统的 sd 卡并上电
- (2) 查看 ip 地址

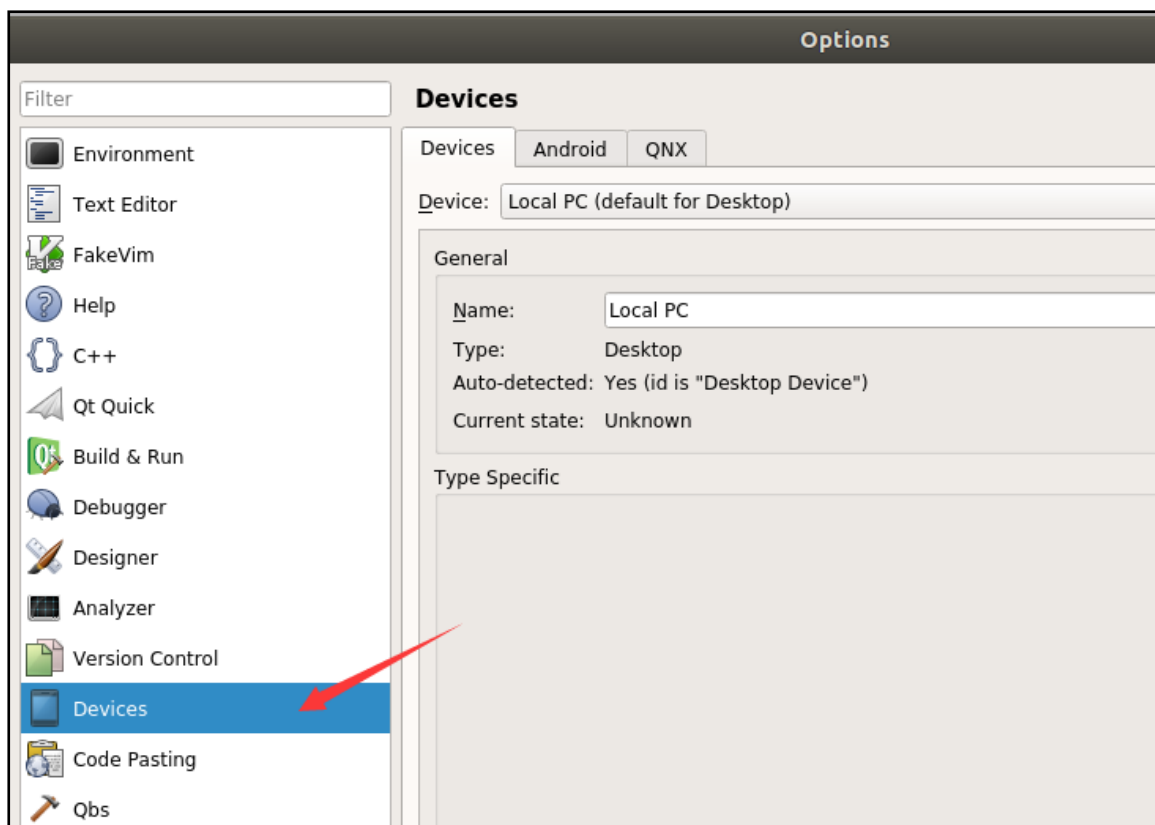
可通过“ipconfig”命令查看板卡的 ip 地址，系统默认启用 DHCP，如果本地不支持 DHCP 服务，则查询结果如下图：

```
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
```

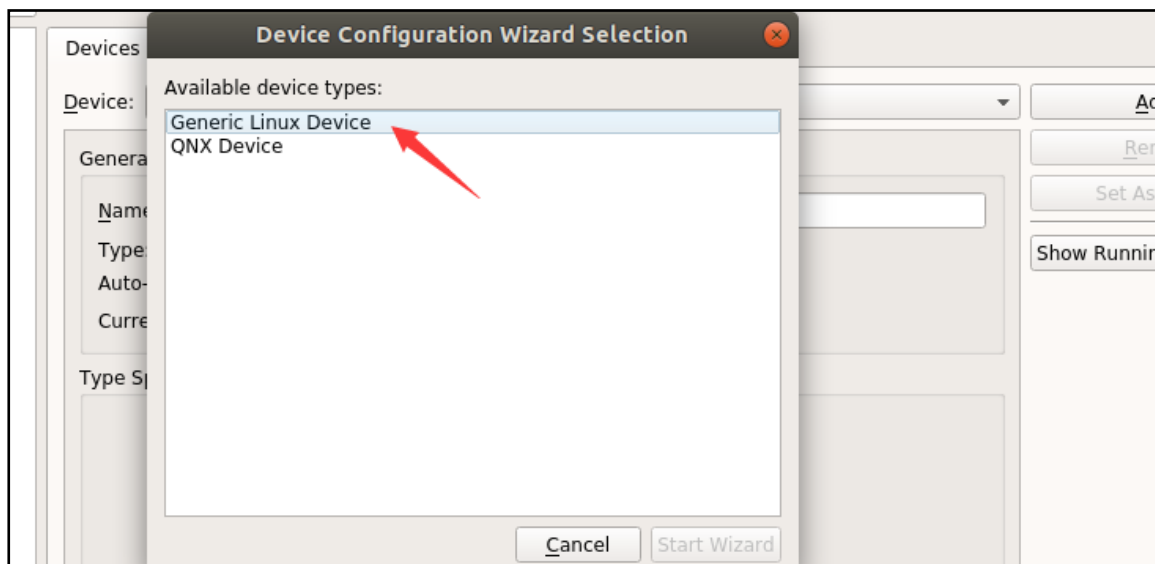
此时需要手动分配一个 ip 地址，可使用命令“ipconfig eth0 192.168.1.45 netmask 255.255.255.0”。这里我们设置板卡的 ip 地址为：192.168.1.45

2.6 QtCreator 配置 GDB 连接板卡

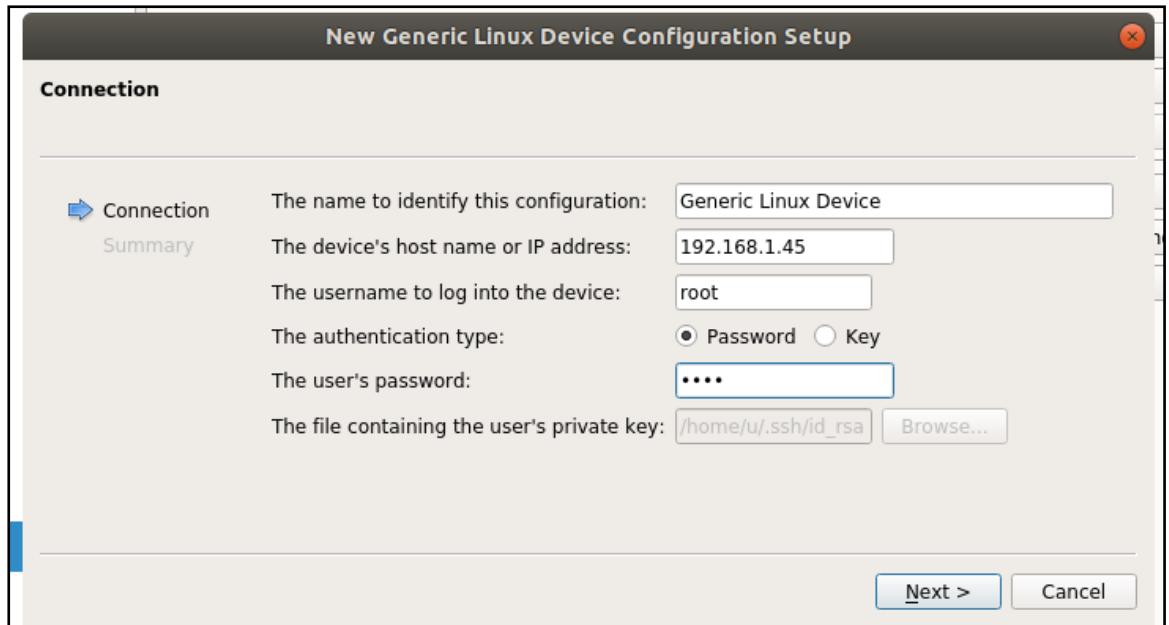
- (1) 打开 QtCreator 的工具设置项



(2) 点击“Add...”按钮，选择“Generic Linux Device”

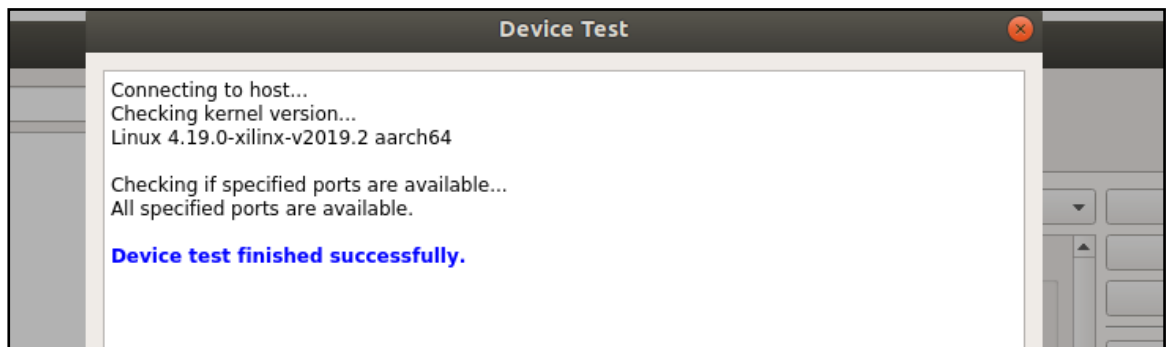


(3) 参数设置如下



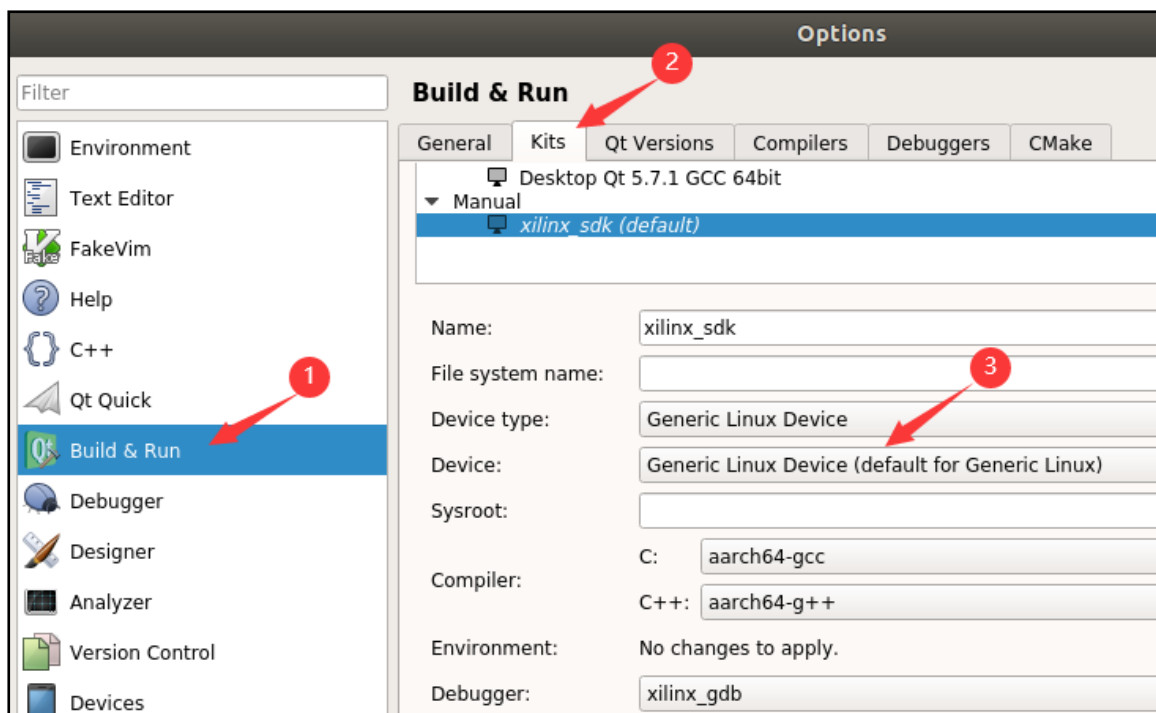
其中密码默认是 root，需要填写

- (4) 点击 “Next” 及 “Finish”，后面出现的测试中，成功后的界面如下



如果没有成功，则需要检测网络连接

- (5) 回到“Device”的设置项，点击 “Apply”，完成此项设置
(6) 切换设置项及配置如下



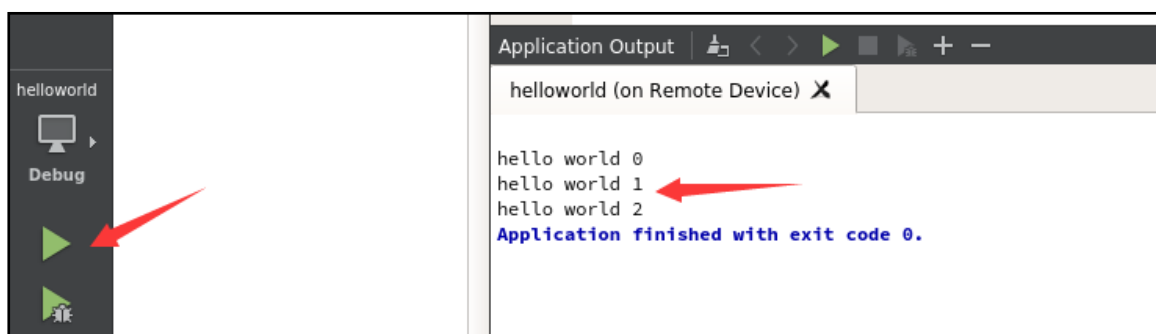
(7) 点击“OK”，完成设置

2.7 运行 Hello World

- (1) 打开 helloworld 工程
- (2) 编译应用程序

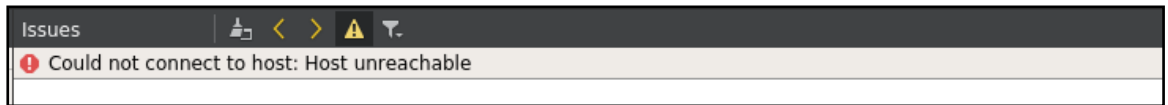


- (3) 点击 QtCreator 的运行按钮

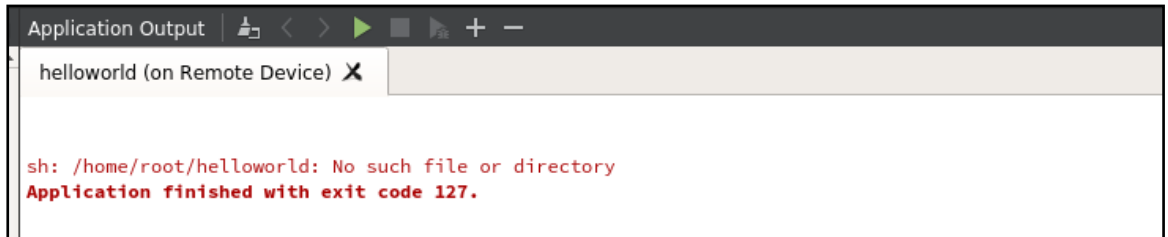


我们可以看到，右边的运行打印信息，说明程序已经运行并退出。

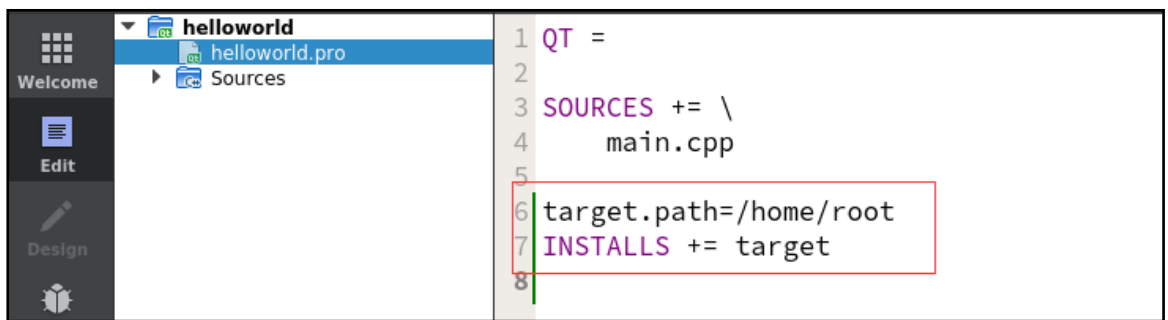
- ① 如果提示如下错误，则需要检查板卡是否上电并进入 linux 系统，网络是否正常



- ② 若提示如下错误，则需要清除编译结果，并重新编译



- (4) 此时我们查看板卡的用户目录“/home/root”，发现多出一个文件 helloworld
(5) 我们打开工程的 pro 文件

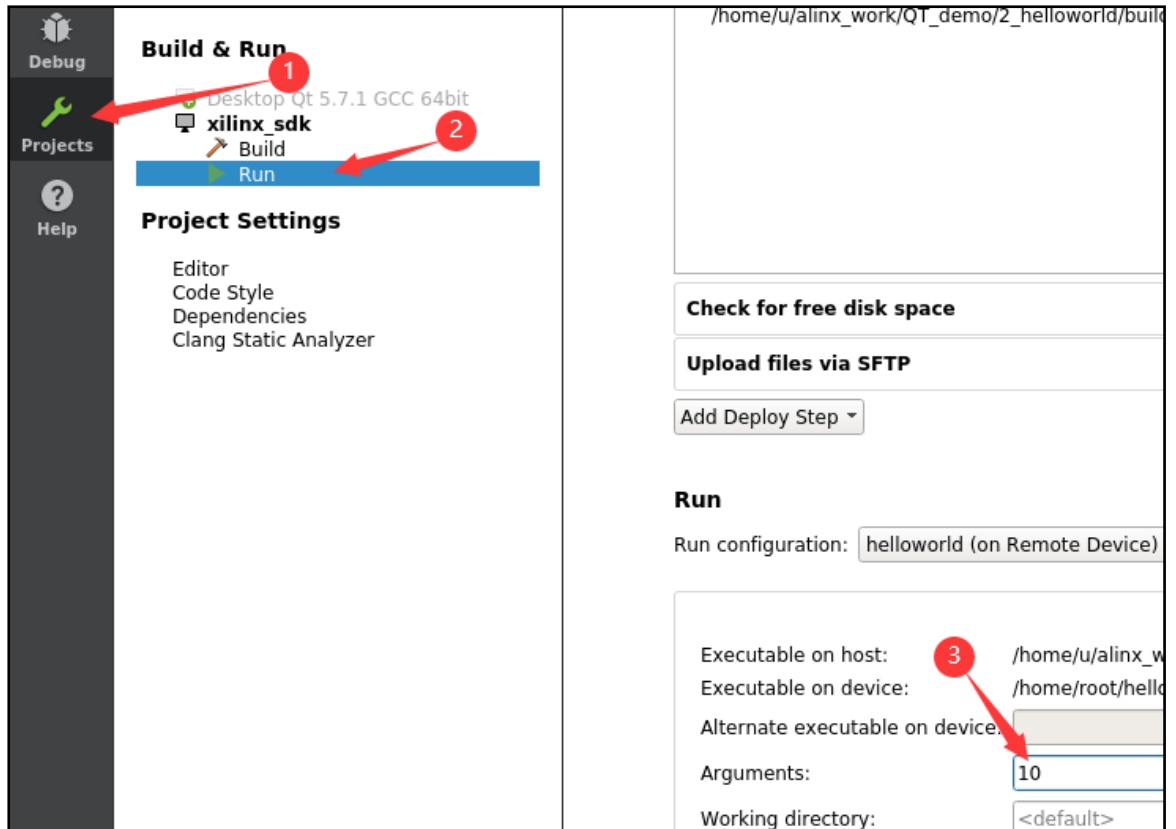


红色框中，是运行时，程序自动加载的目录，这就解释了为什么会在开发板系统“/home/root”目录下出现文件 helloworld

2.8 运行参数设置

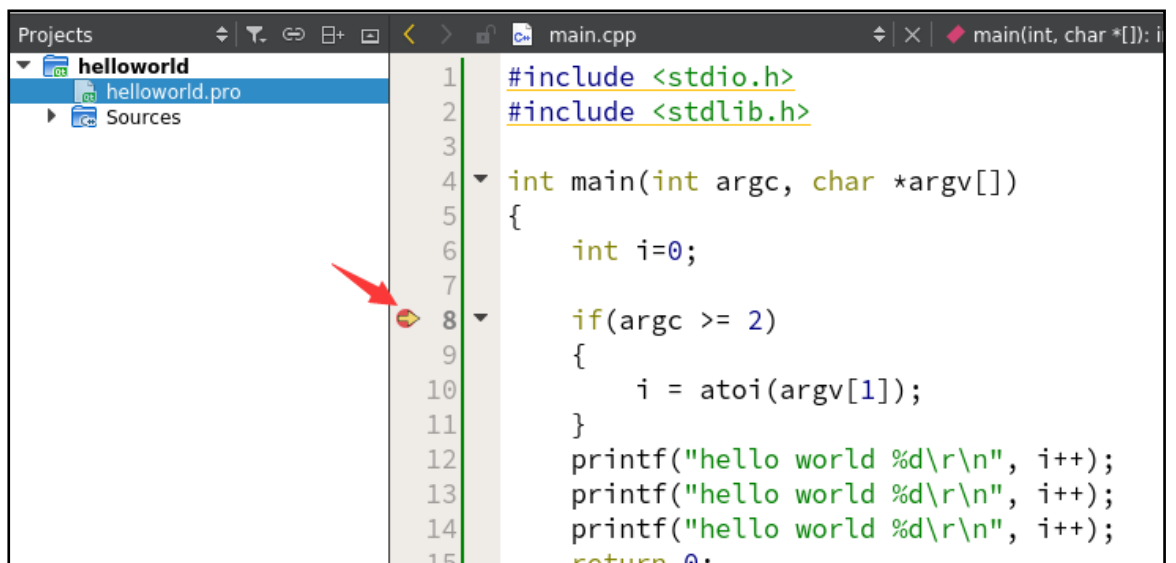
有时候，我们运行程序，需要在后面加上一些运行参数，下面我们介绍如何在远程调试时，为运行程序加上运行参数

- (1) 打开如下设置项

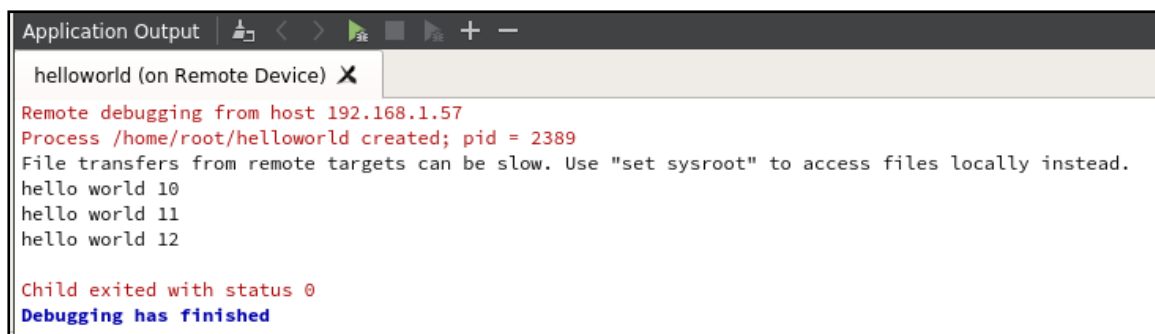


此处参数设置为 10

- (2) 在 main 函数 if 位置设置一个断点
- (3) 点击 debug 调试按钮或快捷键 F5
- (4) 我们看到，程序已经停止在我们设置的断点处



- (5) 按 F5 继续运行，我们看到打印信息如下



```
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
```

我们看到，“hello world”后面的开始数字已经变为 10，表示我们设置的参数已经起作用

第三章 OpenCV 之边缘检测

3.1 OpenCV 简介

OpenCV (开源计算机视觉库) 是一个开源的计算机视觉和机器学习软件库。OpenCV 的建立是为了为计算机视觉应用提供一个通用的基础设施, 并加速机器感知在商业产品中的应用。作为一个 BSD 许可的产品, OpenCV 使企业可以很容易地利用和修改代码。

该库有 2500 多种优化算法, 其中包括一整套经典和最先进的计算机视觉和机器学习算法。这些算法可用于检测和识别人脸、识别物体、对视频中的人类行为进行分类、跟踪摄像机运动、跟踪运动物体、提取物体的三维模型、从立体摄像机生成三维点云、将图像缝合在一起以生成整个场景的高分辨率图像, 从图像库中查找相似的图像, 从使用闪光灯拍摄的图像中移除红眼, 跟踪眼球运动, 识别景物并建立标记以覆盖增强现实等。OpenCV 拥有超过 47000 个用户社区, 估计下载量超过 1800 万。广泛应用于公司、研究团体和政府机构。

Xilinx 的 reVision 的很多实现, 就是基于 opencv 的接口, 其中的硬件实现是 xfopencv。这将有助于我们实现相应算法的硬件(FPGA)加速的实现。

3.2 边缘检测简介

边缘检测是图像处理和计算机视觉中的基本问题, 边缘检测的目的是标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化。这些包括深度上的不连续、表面方向上不连续、物质属性变化和场景照明变化。边缘检测是图像处理和计算机视觉中, 尤其是特征提取中的一个研究领域。

图像边缘信息主要集中在高频段, 通常说图像锐化或检测边缘, 实质就是高频滤波。我们知道微分运算是求信号的变化率, 具有加强高频分量的作用。在空域运算中来说, 对图像的锐化就是计算微分。由于数字图像的离散信号, 微分运算就变成计算差分或梯度。图像处理中有多种边缘检测(梯度)算子, 常用的包括普通一阶差分, Robert 算子(交叉差分), Sobel 算子等等, 是基于寻找梯度强度。拉普拉斯算子(二阶差分)是基于过零点检测。通过计算梯度, 设置阈值, 得到边缘图像。

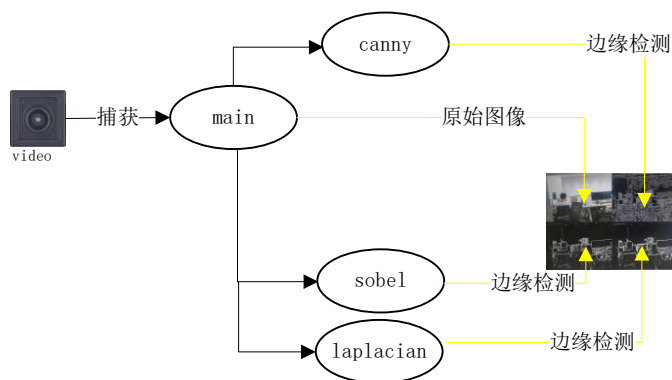
边缘检测一般步骤有:

- ①滤波: 边缘检测算法主要是基于图像强度的一阶和二阶导数, 但是导数对于噪声很敏感, 因此需要采用滤波器来改善与噪声有关的边缘检测器的性能
- ②增强: 增强边缘的基础是确定图像各点邻域强度的变化值。增强算法可以将灰度点邻域强度值有显著变化的点凸显出来
- ③检测: 邻域中有很多的点的梯度值较大, 但是在特定的应用中, 这些点并不是要找的边缘点, 需要取舍

3.3 实验目标

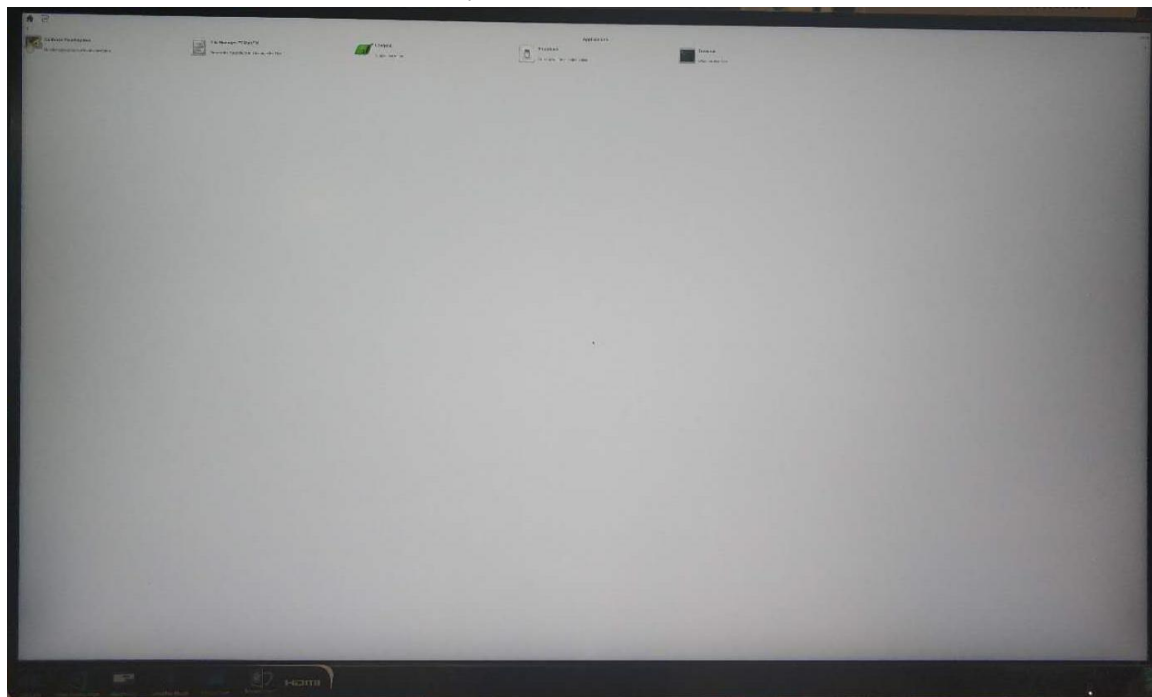
- ① 使用 opencv 来捕获 usb 摄像头图像
- ② canny、sobel、laplacian 算法的边缘检测
- ③ 使用 opencv 来图像显示
- ④ 响应 opencv 窗口关闭操作

3.4 软件流程图



3.5 运行准备

- (1) dp 接口连接至显示器
- (2) 插好 usb 摄像头
- (3) 开发板上电，等待系统启动完成后，dp 显示器可以正确显示开机桌面

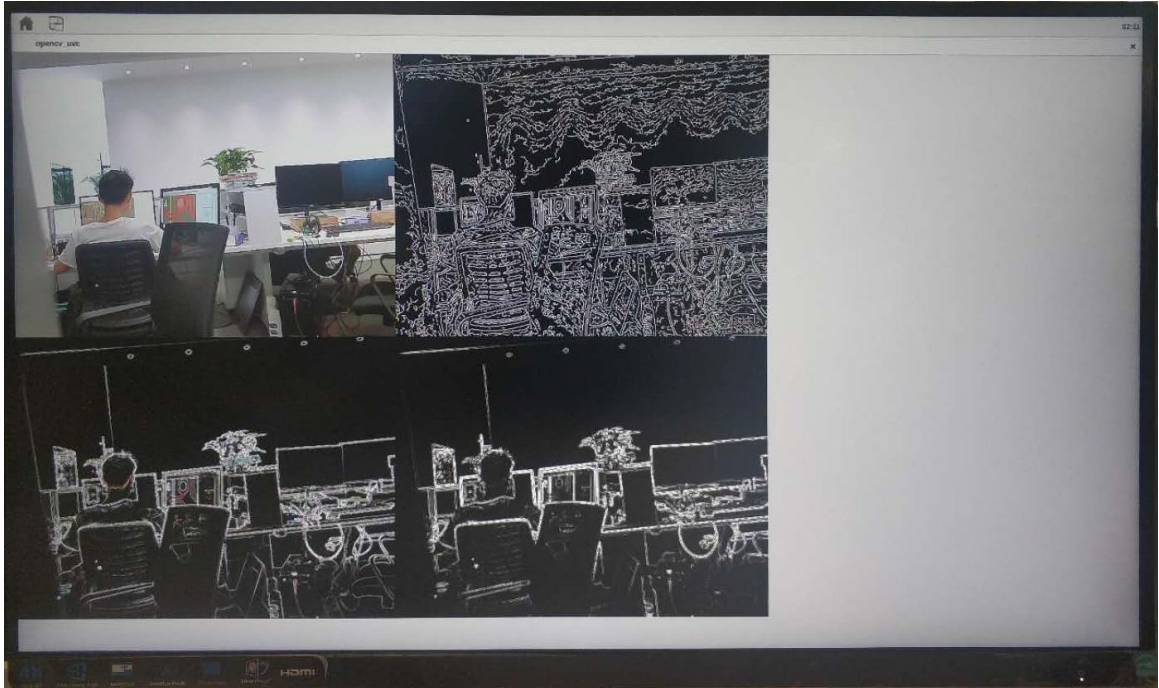


- (4) 在终端下，可以查看到 usb 摄像头设备

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

3.6 程序运行

- (1) 可以参照上一章例程，打开 3_opencv_edgedete 工程，使用 QtCreator 直接运行应用程序，运行效果如下



3.7 代码解析

- (1) 设置环境变量

在终端中输入如下命令，效果等同于代码中的 setenv 调用。

```
export DISPLAY=:0.0
```

- (2) 启动桌面

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

- (3) 显示器分辨率调整

如上述实验，使用一台 4K 显示器，所以开机桌面上的图标很小，为了更好的显示效果，在应用程序中，通过系统调用，将显示输出分辨率设置为 1080P。命令如下：

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

- (4) 显示器输出管理

系统默认使用输出电源管理，即一段时间后，显示器黑屏。这时，我们可以使用命令，

取消该功能，命令如下：

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

(5) 多线程操作

本实验使用三种不同算法完成边缘检测，三种算法被分配到三个线程任务，可以利用 ARM 的四核 A53

(6) usb 摄像头捕获

摄像头分辨率被设置为 640x480，默认打开的摄像头设备为：/dev/video0，因为这里算法运算有点慢，显示的图像会有一点延迟。

(7) canny 边缘检测

John F. Canny 于 1986 年开发出来的一个多级边缘检测算法，被很多人认为是边缘检测的最优算法，最优边缘检测的三个主要评价标准是：

- 低错误率：标识出尽可能多的实际边缘，同时尽可能的减少噪声产生的误报。
- 高定位性：标识出的边缘要与图像中的实际边缘尽可能接近。
- 最小响应：图像中的边缘只能标识一次。

cv::canny 函数参数如下：

InputArray	image,	输入图像
OutputArray	edges	输出边缘图像
double	threshold1	阈值 1
double	Threshold2	阈值 2
int	apertureSize=3	Sobel 算子大小
bool	L2gradient=false	是否采用更精确的方式计算图像梯度

(8) sobel 边缘检测

这是一种较为常用的边缘检测方法，Soble 算子的功能集合了高斯平滑和微分求导，又被称为一阶微分算子，求导算子，在水平和垂直两个方向上求导，得到的是图像在 x 方向与 y 方向梯度图像。缺点：比较敏感，容易受影响，要通过高斯模糊（平滑）来降噪，边缘定位精度不够高。

cv::sobel 函数参数如下：

InputArray	src,	输入图像
OutputArray	dst	输出边缘图像
int	ddepth	输出图像深度
int	dx	导数 x 的阶数
int	dy	导数 y 的阶数
int	ksize = 3	Sobel 内核的大小
double	scale = 1	计算的导数值的可选比例因子

double	delta = 0	在将结果存储到 dst 之前添加到结果中的可选增量值。
Int	borderType=BO RDER_DEFAULT	像素外推法

(9) Laplacian 边缘检测

Laplace 算子是 n 维欧几里德空间中的一个二阶微分算子，在只关心边缘的位置而不考虑其周围的像素灰度差值时比较合适。Laplace 算子对孤立像素的响应要比对边缘或线的响应要更强烈，因此只适用于无噪声图像。存在噪声情况下，使用 Laplacian 算子检测边缘之前需要先进行低通滤波。所以，通常的分割算法都是把 Laplacian 算子和平滑算子结合起来生成一个新的模板。Laplace 算子和 Sobel 算子一样，属于空间锐化滤波操作。

cv::Laplacce 函数参数如下：

InputArray	src,	输入图像
OutputArray	dst	输出边缘图像
int	ddepth	输出图像深度
int	ksize = 1	用于计算二阶导数内核的大小
double	scale = 1	用于计算 Laplace 的可选比例因子
double	delta = 0	在将结果存储到 dst 之前添加到结果中的可选增量值。
Int	borderType=BO RDER_DEFAULT	像素外推法

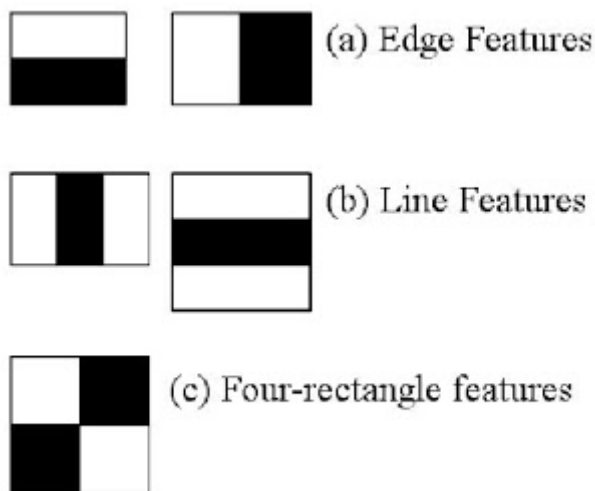
第四章 OpenCV+Qt 之人脸检测

4.1 OpenCV 的级联分类器

分类器，是判断某个事物是否属于某种分类的器件。级联分类器：可以理解为将 N 个单类的分类器串联起来。如果一个事物能属于这一系列串联起来的所有分类器，则最终结果就判定为是，若有一项不符，则判定为否。比如人脸，它有很多属性，我们将每个属性做一个分类器，如果一个模型符合了我们定义的人脸的所有属性，则我们人为这个模型就是一个人脸。那么这些属性是指什么呢？比如人脸需要有两条眉毛，两只眼睛，一个鼻子，一张嘴，一个大概 U 形状的下巴或者是轮廓等等。

基于 Haar 特征的级联分类器的目标检测是 Paul Viola 和 Michael Jones 在 2001 年的论文《Rapid Object Detection using a Boosted Cascade of Simple Features》中提出的一种有效的目标检测方法。这是一种基于机器学习的方法，其中一个级联函数是从许多正面和负面图像中训练出来的。然后，它被用于检测其他图像中的对象。

我们将在这里进行面部检测。该算法首先需要大量的正图像（人脸图像）和负图像（无人脸图像）来训练分类器。然后我们需要从中提取特征。



Haar 特征分为三类：边缘特征、线性特征、中心特征和对角线特征，组合成特征模板。特征模板内有白色和黑色两种矩形，并定义该模板的特征值为白色矩形像素和减去黑色矩形像素和。Haar 特征值反映了图像的灰度变化情况。例如：脸部的一些特征能由矩形特征简单的描述，如：眼睛要比脸颊颜色要深，鼻梁两侧比鼻梁颜色要深，嘴巴比周围颜色要深等。但矩形特征只对一些简单的图形结构，如边缘、线段较敏感，所以只能描述特定走向（水平、垂直、对角）的结构。

通过改变特征模板的大小和位置，可在图像子窗口中穷举出大量的特征。上图的特征模板称为“特征原型”；特征原型在图像子窗口中扩展（平移伸缩）得到的特征称为“矩形特征”；矩形特征的值称为“特征值”。

矩形特征可位于图像任意位置，大小也可以任意改变，所以矩形特征值是矩形模版类别、

矩形位置和矩形大小这三个因素的函数。故类别、大小和位置的变化，使得很小的检测窗口含有非常多的矩形特征，如：在 24*24 像素大小的检测窗口内矩形特征数量可以达到 16 万个。这样就有两个问题需要解决了：(1) 如何快速计算那么多的特征？(2) 哪些矩形特征才是对分类器分类最有效的？

4.2 积分图

为了快速计算特征，这里引入了积分图，无论图像有多大，它都会将给定像素的计算减少到仅涉及四个像素的操作。积分图就是只遍历一次图像就可以求出图像中所有区域像素和的快速算法，大大的提高了图像特征值计算的效率。

积分图主要的思想是将图像从起点开始到各个点所形成的矩形区域像素之和作为一个数组的元素保存在内存中，当要计算某个区域的像素和时可以直接索引数组的元素，不用重新计算这个区域的像素和，从而加快了计算（这有个相应的称呼，叫做动态规划算法）。积分图能够在多种尺度下，使用相同的时间（常数时间）来计算不同的特征，因此大大提高了检测速度。

积分图是一种能够描述全局信息的矩阵表示方法。积分图的构造方式是位置 (i,j) 处的值 $ii(i,j)$ 是原图像 (i,j) 左上角方向所有像素的和：

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

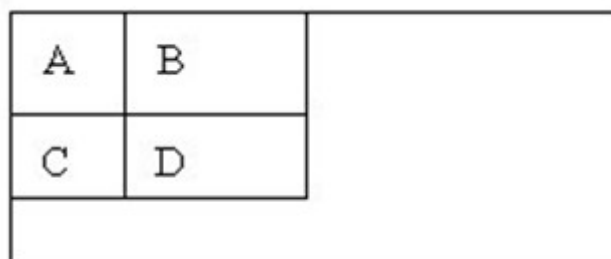
积分图构建算法

- ① 用 $s(i,j)$ 表示行方向的累加和，初始化 $s(i,-1)=0$;
- ② 用 $ii(i,j)$ 表示一个积分图像，初始化 $ii(-1,i)=0$;
- ③ 逐行扫描图像，递归计算每个像素 (i,j) 行方向的累加和 $s(i,j)$ 和积分图像 $ii(i,j)$ 的值

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

$$ii(i,j)=ii(i-1,j)+s(i,j)$$
- ④ 扫描图像一遍，当到达图像右下角像素时，积分图像 ii 就构造好了

积分图构造好之后，图像中任何矩阵区域的像素累加和都可以通过简单运算得到如图所示。



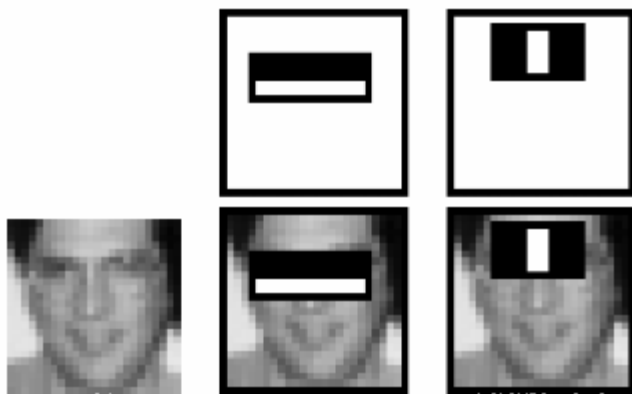
设 D 的四个顶点分别为 α 、 β 、 γ 、 δ ，则 D 的像素和可以表示为

$$Dsum = ii(\alpha) + ii(\beta) - ii(\gamma) - ii(\delta);$$

而 Haar-like 特征值无非就是两个矩阵像素和的差，同样可以在常数时间内完成。所以矩形特征的特征值计算，只与此特征矩形的端点的积分图有关，所以不管此特征矩形的尺度变换如何，特征值的计算所消耗的时间都是常量。这样只要遍历图像一次，就可以求得所有子窗口的特征值。

4.3 选取特征

在我们计算的所有这些特征中，大多数都是无关的。例如，考虑下面的图片。最上面一排显示了两个很好的特性。选定的第一个特征似乎集中在眼睛区域通常比鼻子和脸颊区域暗的属性上。选择的第二个特征依赖于眼睛比鼻梁暗的属性。但同样的窗口应用于脸颊或任何其他地方都是没有作用的。那么我们如何从 160000 多个功能中选择最好的功能呢？这里可以通过 AdaBoost 算法来实现。



我们将每个特征应用于所有训练图像。对于每一个特征，它都会找到最佳的阈值，将人脸分为正反两类。显然，会有正确或错误的分类。我们选择错误率最小的特征，这意味着它们是最准确地分类人脸和非人脸图像的特征。（过程并不像这么简单。每幅图像在开始时都有一个相等的权重。每次分类后，错误分类图像的权重都会增加。然后做同样的过程。计算新的错误率。还有新的权重。该过程将继续进行，直到达到所需的精度或错误率或找到所需数量的特征）。

4.4 Qt 简介

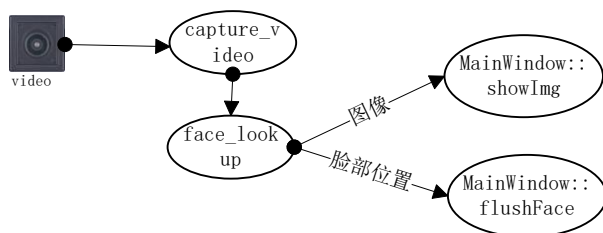
Qt 是一个跨平台的 C++ 开发库，主要用来开发图形用户界面（Graphical User Interface, GUI）程序，当然也可以开发不带界面的命令行（Command User Interface, CUI）程序。Qt 还存在于 Python、Ruby、Perl 等脚本语言的绑定，也就是说可以使用脚本语言开发基于 Qt 的程序。Qt 支持的操作系统有很多，例如通用操作系统 Windows、Linux、Unix，智能手机系统 Android、iOS、WinPhone，嵌入式系统 QNX、VxWorks 等等。Qt 虽然经常被当做一个 GUI 库，用来开发图形界面应用程序，但这并不是 Qt 的全部；Qt 除了可以绘制漂亮的界面（包括控件、布局、交互），还包含很多其它功能，比如多线程、访问数据库、图像处理、音频视频处理、网络通信、文件操作等，这些 Qt 都已经集成了。

Qt 虽然也支持手机操作系统，但是由于 Android 本身已经有 Java 和 Kotlin，iOS 本身已经有 Objective-C 和 Swift，所以 Qt 在移动端的市场份额几乎可以忽略。

4.5 实验目标

- ① 调用 Qt 库显示视频图像与脸部框图
- ② opencv 库的 CascadeClassifier 类的使用

4.6 软件流程图

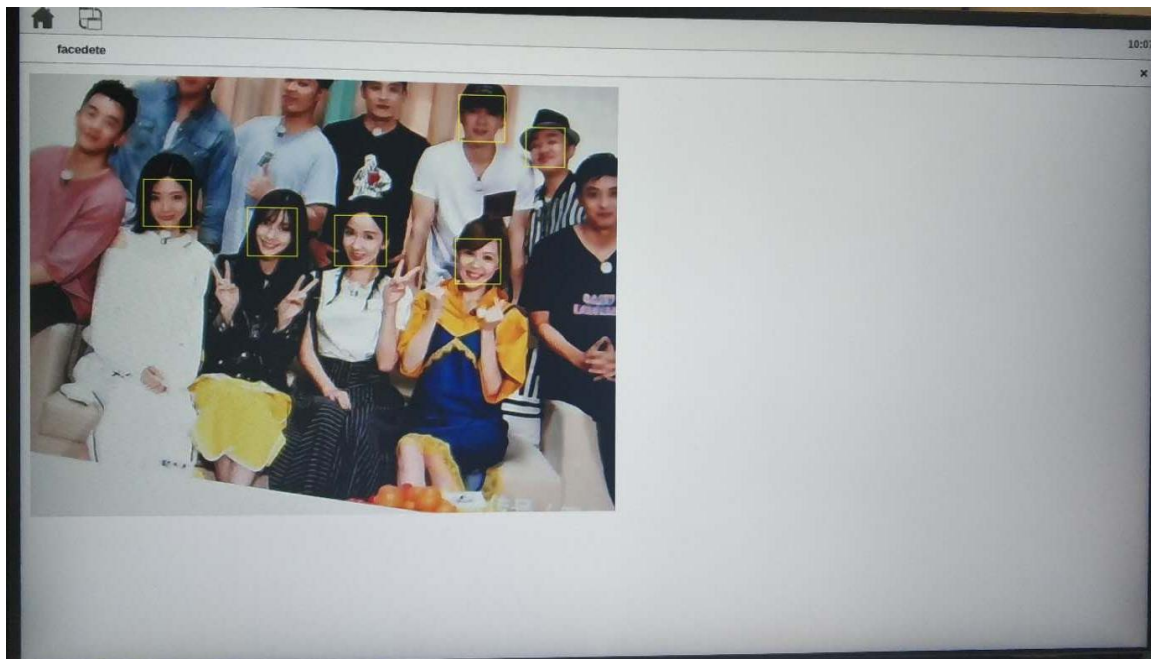


4.7 运行准备

- (1) 按上章节做好准备工作
- (2) 将 haar_train 文件夹复制到开发板/home/root 目录

4.8 程序运行

- (1) 运行应用程序，效果如下



4.9 代码解析

- (1) 检测人脸主要是 face_lookup.cpp 文件中的 face_cascade.detectMultiScale, 初始化时, 调用 face_cascade.load, 加载训练好的模型 haarcascade_frontalface_alt.xml。
- (2) 在 MainWindow::flushFace 中, 将根据上面检测结果, 显示人脸框图。
- (3) 因为训练模型的原因, 这里检测, 需要人脸不能是倾斜的或侧面的。
- (4) .pro 文件
因为调用 Q 显示界面, 在 facedete.pro 文件中, "QT=" 后面添加了 core gui widgets 几个 Qt 组件
- (5) Qt 的信号与槽
程序将 face_lookup 和 MainWindow 的信号与槽绑定, 由 MainWindow 处理需要显示的图片 and 标志出脸的位置
- (6) 与上面例程相同, 显示效果比较卡顿。

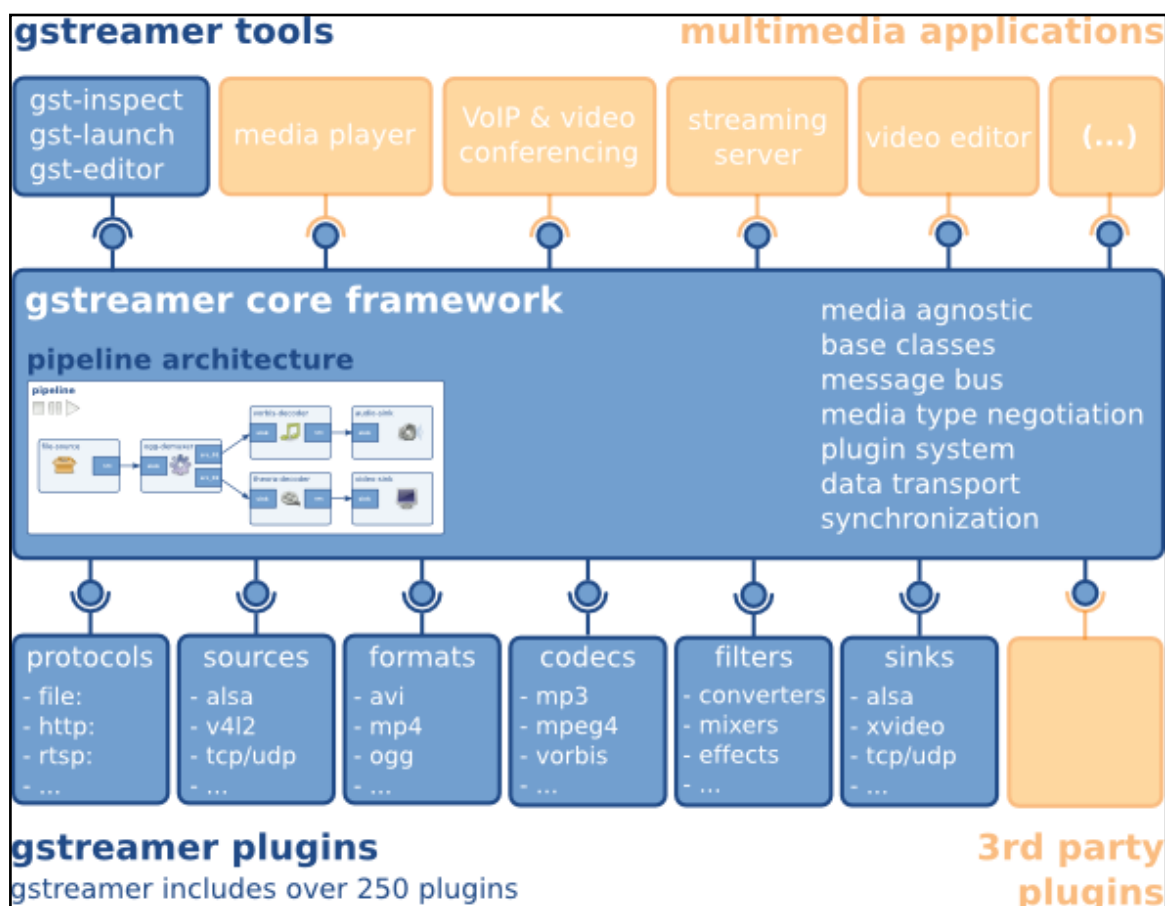
第五章 GStreamer 的摄像头显示

5.1 GStreamer 简介

GStreamer 是一个创建流媒体应用程序的框架。其基本设计思想来自于俄勒冈(Oregon)研究生学院有关视频管道的创意,同时也借鉴了 DirectShow 的设计思想。它的特点如下:

- ① 结构清晰, 功能强大
- ② 面向对象的编程思想
- ③ 灵活的可扩展功能
- ④ 高性能
- ⑤ 核心库与插件分离

GStreamer 的架构如下, 它支持的应用范围从简单的 Ogg/Vorbis 回放、音频/视频流到复杂的音频(混合)和视频(非线性编辑)处理。应用程序可以透明地利用先进的编解码器和过滤技术。



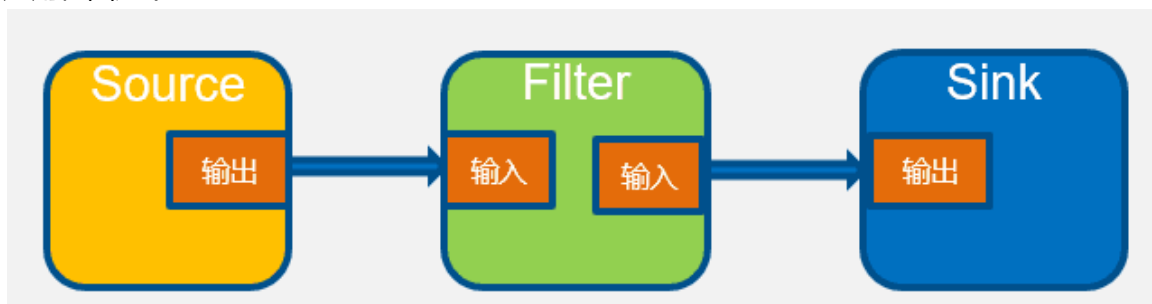
5.2 GStreamer 基本概念

(1) 元件 (element)

Elements 是组成管道的最基本的构件。可以将若干个元件(Elements)连接在一起,从而创建一个管道(pipeline)来完成一个特殊的任务,例如,媒体播放或者录像。Elements 根据功能分类:

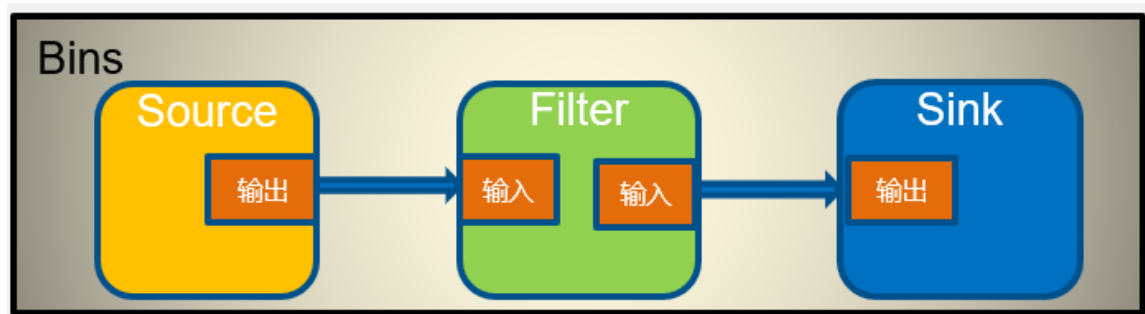
- ① Source elements
- ② Sink elements
- ③ Filter elements

应用框图如下:



(2) 箱柜 (bin) 和管道 (Pipeline)

箱柜(Bins)是一个可以装载元件(element)的容器。管道(pipelines)是箱柜(Bins)的一个特殊的子类,管道(pipelines)可以操作包含在它自身内部的所有元件(element)。如下图



(3) 衬垫 (Pad)

衬垫 (Pads) 相当于一个元件的接口, 各个元件 (element) 通过这个接口进行连接, 这样数据流就可以在这些元件中进行传输。Pads 分类:

- ① 永久型 (always)
- ② 随机型 (sometimes)
- ③ 请求型 (on request)

(4) 功能 (Cap)

Caps 描述了能够通过衬垫或当前通过衬垫的数据流格式。通过 `gst-inspect-1.0` 命令可以查看元件所支持的媒体类型。

(5) 总线 (Bus)

每一个管道默认包含一条总线, 应用程序不需要再创建总线。应用程序在总线上设置

一个消息处理器。当主循环运行的时候，总线将会轮询这个消息处理器是否有新的消息，当消息被采集到后，总线将调用相应的回调函数来完成任务

(6) 缓冲区 (Buffer)

缓冲区包含了创建的管道里的数据流。通常一个源元件会创建一个新的缓冲区，同时元件还将会把缓冲区的数据传递给下一个元件。当使用 GStreamer 底层构造来创建一个媒体管道的时候，不需要自己来处理缓冲区，元件将会自动处理这些缓冲区。缓冲区主要由以下组件构成：

- ① 指向某块内存的指针
- ② 内存的大小
- ③ 缓冲区的时间戳
- ④ 一个引用计数，指出了缓冲区所使用的元件数。没有元件可引用的时候，这个引用将用于销毁缓冲区

(7) 事件 (Events)

事件包括管道里的控制信息，如寻找信息和流的终止信号。

5.3 实验目标

- ① gstreamer 工具使用
- ② gstreamer 编程

5.4 Gstreamer 常用工具

(1) gst-inspect-1.0

不带参数，它会列出所有可用的 element，也就是你所有可以使用的元素，带一个文件名，它会把这个文件作为 GStreamer 的一个插件，试着打开，然后列出内部所有的 element，带一个 GStreamer 的 element，会列出该 element 的所有信息。

打印插件列表及其信息，如：

打印支持的插件列表	gst-inspect-1.0 -a
打印插件 filesrc 信息	gst-inspect-1.0 filesrc

Pad Templates：这部分会列出所有的 Pad 的种类以及它们的 Caps。通过这些你可以确认是否可以和某一个 element 连接

(2) gst-discoverer

这个工具可以用来查看文件里面包含的 Caps，它是对 GstDiscoverer 对象的一个封装。接受从命令行输入一个 URI，然后打印出所有的信息。这个在查看媒体是如何编码如何复用是很有用的，这样我们可以确定把什么 element 放到 pipeline 里面。

(3) gst-launch-1.0

这个工具可以创建一个 pipeline，初始化然后运行。它可以让你在正式写代码实现 pipeline 之前先快速测试一下，看看是否能工作。若需要查看 pipeline 里面一个 element



5.6 编程运行程序

通过命令，我们可以快速的验证各种我们设想的方案，但如果想要更多的状态监控或用户控制，这就需要通过编程来实现。

本章的例程，是通过程序来实现上面的命令效果

- (1) 需要注意的事项与上面的一样，需要修改 usb 摄像头支持的分辨率与格式
- (2) 程序通过添加消息事件的回调函数 sink_message，监控与处理各种状态变化。
- (3) 运行程序后，可以看到显示的图像与命令运行的效果一样，此时拔掉摄像头，程序打印的信息如下且程序退出运行。

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

5.7 代码解析

- (1) gst_parse_launch

该函数创建一个 pipelin，函数参数如下：

const gchar *	pipeline_description	描述管道
GError **	error	错误返回

pipeline_description 内容与前面讲的 gst-launch-1.0 验证的内容几乎相同。

- (2) gst_bus_add_watch

添加总线的回调函数，处理必要的消息。

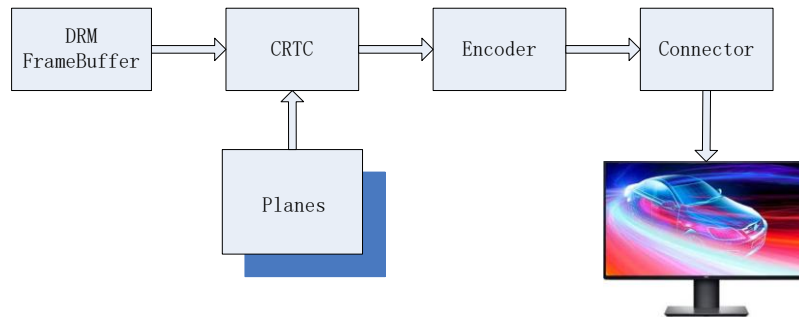
第六章 Qt+DRM+Gstreamer 的摄像头显示

6.1 DRM 简介

DRM, 英文全称 Direct Rendering Manager, 即直接渲染管理器。它是为了解决多个程序对 Video Card 资源的协同使用问题而产生的。它向用户空间提供了一组 API, 用于输出管理。

DRM 到现在已经涵盖了以前由用户空间程序处理的很多功能, 比如 帧缓存区的管理和模式设置, 内存共享对象和内存同步。其中一些拓展具有特定的名称, 比如图形执行管理器 GEM 或者内核模式设置 KMS, 这些都是属于 DRM 子系统。

DRM 显示, 主要涉及以下 5 个模块



(1) DRM Framebuffer

在开发者看来, FrameBuffer 是一块内存, 往内存中写入特定格式的数据就意味着向屏幕输出内容。所以说 FrameBuffer 就是一块画布。例如对于初始化为 16 位色的 FrameBuffer 来说, FrameBuffer 中的两个字节代表屏幕上一个点, 从上到下, 从左至右, 屏幕位置与内存地址是顺序的线性关系。

(2) CRTC

CRTC 的任务是从 Framebuffer 中读出待显示的图像, 并按照相应的格式输出给 Encoder。CRTC 虽然字面上意思为阴极射线显像管控制器, 但 CRT 在普通显示设备中早已被淘汰, DRI 中 CRTC 主要承担的作用如下:

- ① 配置适合显示器的分辨率 (kernel) 并输出相应时序 (hardware logic)
- ② 扫描 framebuffer 送显到一个或多个显示设备中

(3) Planes

随着软件技术的不断更新, 对硬件的性能要求越来越高, 在满足功能正常使用的前提下, 对功耗的要求也越来越苛刻。本来 GPU 可以处理所有图形任务, 但是由于它运行时的功耗实在太高, 设计者们决定将一部分简单的任务交给 Display Controller 去处理 (比如合成), 而让 GPU 专注于绘图 (即渲染) 这一主要任务, 减轻 GPU 的负担, 从而达到降低功耗提升性能的目的。于是, Plane (硬件图层单元) 就诞生了。Planes 主要是将多个图层进行叠加输出, 有些硬件支持图层的平移、缩放、裁剪、alpha 通道等。

(4) Encoder

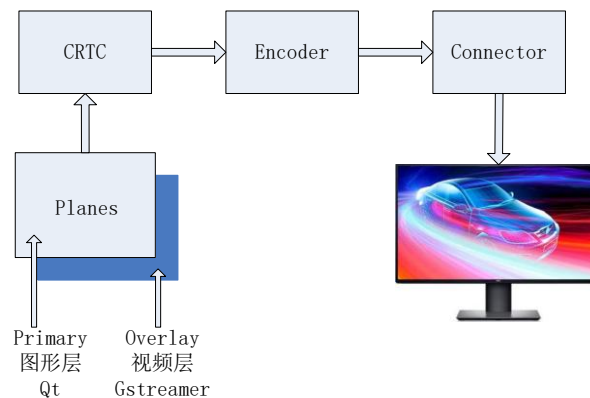
将一定格式的图像信号 (如 RGB、YUV 等) 编码成 connector 需要输出的信号。以 HDMI 为例, 帧/行同步/显示内容都是通过 TMDS data 的串行总线输出的, 那么并行的时序按照 HDMI 的标准编码为串行顺序则是 Encoder 的任务;

(5) Connector

Connector 其实就是和显示器连接的物理接口, 常见的有 VGA/HDMI/DVI/DP 等, 可以通过 connector 读出的显示器支持的参数;

6.2 实验介绍

MPSoc 的 DP 输出, 支持两个图层, 上面的图层(Primary)支持 alpha 通道。本章实验, 使用 Qt 输出内容至 Primary 图层, Gstreamer 捕获视频数据至下面的图层(Overlay)。最终这两个图层由硬件混合输出。工作框图如下:

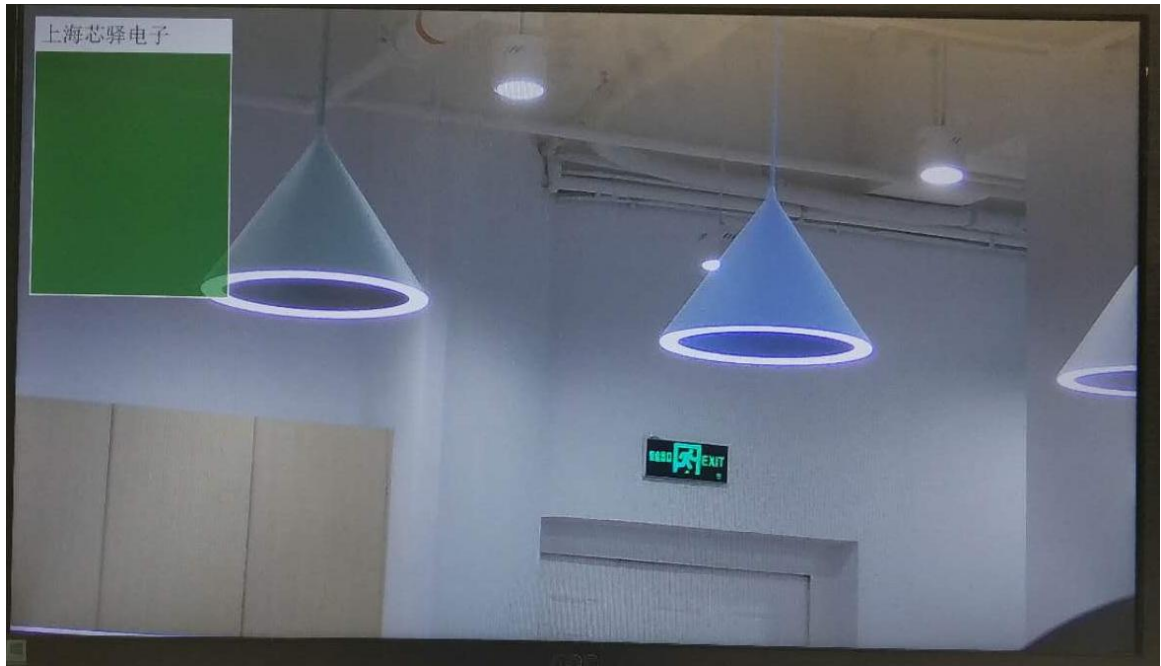


6.3 实验目标

- ① DRM 输出管理
- ② DRM 图层管理
- ③ Qt 与 GStreamer 分层显示

6.4 运行程序

- (1) 需要注意, 本实验需要 usb 摄像头支持 1080 分辨率(指的是裸流, 如上面的 YUYV422 格式)
- (2) 实验运行效果
可以看到在视频上面叠加了一个 Qt 界面



6.5 代码分析

(1) 检测显示器是否连接

在 main 函数中, 调用 `init_drm` 判断初始化是否成功, 返回-2, 即没有支持的显示格式, 说明没有连接显示器。

(2) 设置显示器输出分辨率

通过函数 `drmModeSetCrtc`, 设置输出的分辨率及刷新率, 这里没有设置 buffer, 所以第三个参数设置为-1

(3) 设置图形层的透明属性

图形层支持全局的和单像素的 alpha 值设置。这里将其设置为单像素的, 即将 `g_alpha_en` 设置为 0

(4) Qt 显示

Qt 默认显示在 Primary 层

(5) GStreamer 显示

通过 `drmModeGetPlaneResources` 可以获取 plane 的信息, 包括 plane 的 id 号, 这里获取到 Overlay 图层的 ID 为 35。通过代码可以看到, `kmssink` 的属性设置如下: `kmssink sink-type=dp bus-id=fd4a0000.zynqmp-display fullscreen-overlay=false plane-id=35`

第七章 linux 寄存器操作

7.1 linux 内存映射

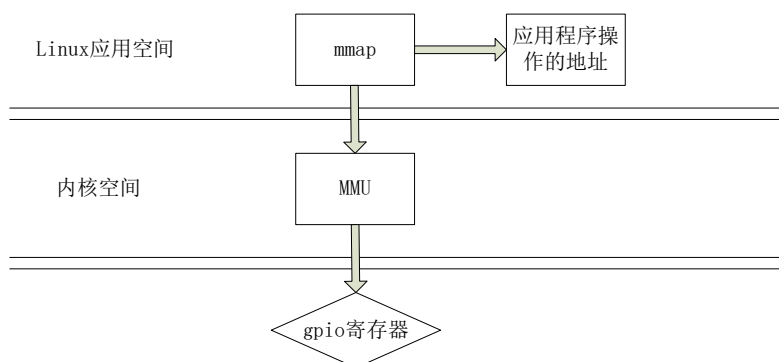
内存映射，就是将用户空间的一段内存区域映射到内核空间，映射成功后，用户对这段内存区域的修改可以直接反映到内核空间，同样，内核空间对这段区域的修改也直接反映用户空间。这对于内核空间与用户空间传输大量数据的话，效率是非常高的。

例如，从硬盘上将文件读入内存，都要经过文件系统进行数据拷贝，并且数据拷贝操作是由文件系统和硬件驱动实现的，理论上来说，拷贝数据的效率是一样的。但是通过内存映射的方法访问硬盘上的文件，效率要比 read 和 write 系统调用高，这是为什么呢？原因是 read() 是系统调用，其中进行了数据拷贝，它首先将文件内容从硬盘拷贝到内核空间的一个缓冲区，然后再将这些数据拷贝到用户空间，在这个过程中，实际上完成了两次数据拷贝；而 mmap() 也是系统调用，mmap() 中没有进行数据拷贝，真正的数据拷贝是在缺页中断处理时进行的，由于 mmap() 将文件直接映射到用户空间，所以中断处理函数根据这个映射关系，直接将文件从硬盘拷贝到用户空间，只进行了一次数据拷贝。因此，内存映射的效率要比 read/write 效率高。

7.2 实验简介

有时候，某个外设的功能已经在裸机下验证通过，如果该外设没有用到中断，这时，我们可以通过 linux 下映射寄存器的方法，将这个裸机下的程序直接移植到 linux 下使用，这样，可以省去开发驱动的事。

本例程移植了 vitis 下 gpio 的例程，驱动 ps_led，作每秒闪烁一次的动作。其它的操作，过程与之类似。



7.3 运行程序

运行应用程序 register_opt 后，可以看到，PS 的 led 灯每秒闪烁一次

7.4 代码分析

- (1) 关于 gpio 的寄存器地址, 这里都是从 vitisi 的 gpio 例程中抄出来的, 后面我们的开发思路也应该是这样, 先用 vitisi 的裸机程序验证。很多外设和 fpga 端的 ip, vitisi 会帮我们生成好操作的方法和操作的地址, 这样就不需要我们再去找对应关系。
- (2) 打开/dev/mem 的时候, 使用选项 O_SYNC
向外部写入数据后, 通常数据是写入 cache 缓冲。O_SYNC 将确保数据写入至外设才返回。需要注意, 这里的 O_SYNC, 仅影响写操作。
- (3) msync 的调用
如果需要向外设一次写入比较多的数据, 此时如果调用 O_SYNC, 将会严重影响系统的性能, 此时如果我们不使用 O_SYNC, 而是在写完数据后, 调用 msync, 这样将会提升写的性能。
- (4) 读操作一致性问题
如果需要读外设的数据, 因为 cache 的存在, 应用中取到的数据是 cache 中的数据, 而不是外设的最新状态, 此时读到的可能是一个错误的值。