QT5.6文件系统简介

本套资料建议使用Ubuntu14.04

QT\_arm官方参考文档：<http://doc.qt.io/qt-5/embedded-linux.html>

[一：用buildroot制作根文件系统 3](#_Toc22433)

[1.1 Ubuntu 14.04 编译arm 环境配置 3](#_Toc24840)

[1.2 安装buildroot，配置buildroot运行环境 4](#_Toc4905)

[1.3 根据开发板架构配置buildroot 4](#_Toc16374)

[1.3.1 配置开发板架构，如下图所示 4](#_Toc9586)

[1.3.2 配置外部交叉编译工具链 5](#_Toc20738)

[1.3.3 配置 根文件系统选项 6](#_Toc6968)

[1.3.4根文件系统压缩包的压缩方式 7](#_Toc29462)

[二：制作ubi文件系统 9](#_Toc12396)

[2.1 必备工具安装 9](#_Toc12358)

[2.2 ：制作200M大小的IMG文件系统 9](#_Toc13593)

[2.3对img文件系统进行读写操作 11](#_Toc21933)

[2.4制作ubi和ubifs根文件系统 13](#_Toc29245)

[2.4.1: 生成ubifs文件系统 13](#_Toc16218)

[2.42：将ubifs文件系统转化为ubi.img文件系统 13](#_Toc22155)

[2.4.4：rootfs目录来源（举列说明） 14](#_Toc8331)

[三：编译tslib 15](#_Toc12862)

[3.1手动编译tslib 15](#_Toc4801)

[3.1.1下载tslib(1.5)软件源码 15](#_Toc16980)

[3.1.2在Ubuntu上安装一些工具包 15](#_Toc19060)

[3.1.3解压源码，进入解压的目录，执行如下命令 15](#_Toc3832)

[3.1.4 安装 15](#_Toc1650)

[3.1.5 配置 16](#_Toc32225)

[3.1.6执行测试命令 16](#_Toc14126)

[四：手动编译QT5.6 18](#_Toc16616)

[4.1 安装pc机 qt5.6开发软件 18](#_Toc1417)

[4.1用buildroot编译QT支持3d编译效果： 18](#_Toc18989)

[4.2手动编译移植ARM板qt 19](#_Toc23769)

[安装必备软件：sudo apt-get install g++ 19](#_Toc31359)

[五：搭建QT5.6的编译环境 26](#_Toc31293)

[5.1在Ubuntu上配置qt编译arm格式的qt程序 26](#_Toc9188)

[5.1.1配置qmake 26](#_Toc21375)

[5.1.2配置交叉编译工具链 26](#_Toc2413)

[5.1.3 配置Kits工具 27](#_Toc31250)

[5.2 在PC机上编译一个QT小实例 28](#_Toc3783)

[5.2.1 QT-UI编程 28](#_Toc748)

[六：opensshd服务的配置 34](#_Toc7455)

[6.1 openssh 的移植 34](#_Toc591)

[6.2 配置openssh的配置文件 34](#_Toc12370)

[七：vsftpd服务的配置与使用 35](#_Toc25280)

[7.1 vsftpd 的移植 35](#_Toc22150)

[7.2 vsftpd使用前的准备 35](#_Toc22707)

[7.3配置Vsftpd的配置文件 35](#_Toc32156)

[7.4 使用vsftpd 36](#_Toc194)

[7.6 修改源码 36](#_Toc1307)

[八httpd服务的配置与使用 38](#_Toc25700)

[8.1 httpd的移植 38](#_Toc128)

[8.2 配置相关配置文件 38](#_Toc16759)

[8.2.1 创建配置文件 39](#_Toc13600)

[8.2.2 修改/etc/php.ini 39](#_Toc4904)

[8.2.3 创建启动脚本 39](#_Toc30714)

[8.2.4设置静态IP 40](#_Toc3828)

[8.2.5 PHP测试 40](#_Toc29480)

[配置VIM编辑器 41](#_Toc7895)

[Xorg与fluxbox移植 41](#_Toc22483)

[virtualbox 安装 USB 扩展功能 44](#_Toc27120)

[十二：定制ubuntu 47](#_Toc26841)

[十三：SGX应用开发入门 48](#_Toc18193)

[十四：AM335x+Linux移植SGX 49](#_Toc27521)

[十五：openVPN使用 55](#_Toc4381)

[十六：设备树的使用 55](#_Toc14065)

[参考链接：http://blog.csdn.net/dzw19911024/article/details/51741405 55](#_Toc21292)

[1. 设备树的编译与重新编译 56](#_Toc8265)

[2. 设备树在内核的表示方法： 56](#_Toc12985)

[3. 设备树的解读与配置 56](#_Toc12423)

[十七：gpio的配置 59](#_Toc28784)

[十八：TI芯片手册使用简介 60](#_Toc15014)

[十九：modprobe使用介绍 61](#_Toc23647)

[二十：开发板模块测试 61](#_Toc23444)

**一：用buildroot制作根文件系统**

* 1. **Ubuntu 14.04 编译arm 环境配置**

下载32交叉编译工具链所需要的库

**sudo apt-get install libc6:i386 libstdc++6:i386 libncurses5:i386 zlib1g:i386**

* 1. **安装buildroot，配置buildroot运行环境**

**sudo apt-get update –y**

**wget <https://buildroot.org/downloads/buildroot-2017.02.3.tar.gz>**

**tar xvf buildroot-2017.02.3.tar.gz**

**sudo apt-get install texinfo -y**

**sudo apt-get install libncurses5-dev -y**

**sudo add-apt-repository ppa:webupd8team/java -y**

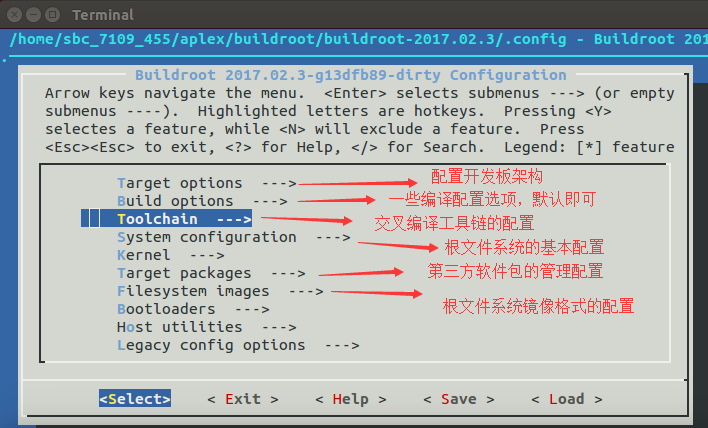
**sudo apt-get update**

**sudo apt-get install oracle-java8-installer -y**

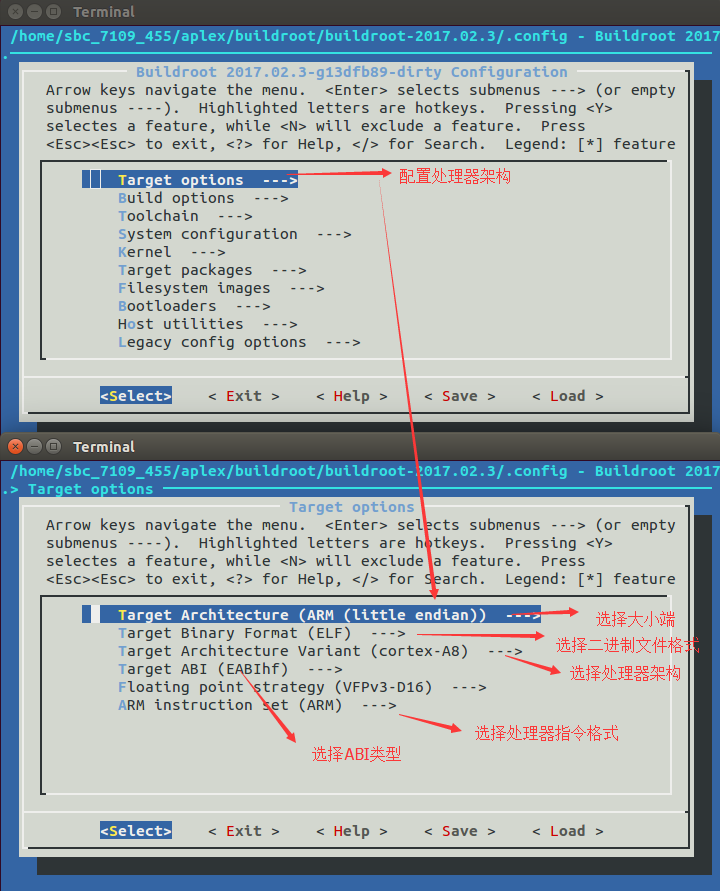
**sudo apt-get install oracle-java8-set-default -y**

* 1. **根据开发板架构配置buildroot**

Buildroot 简介如下图

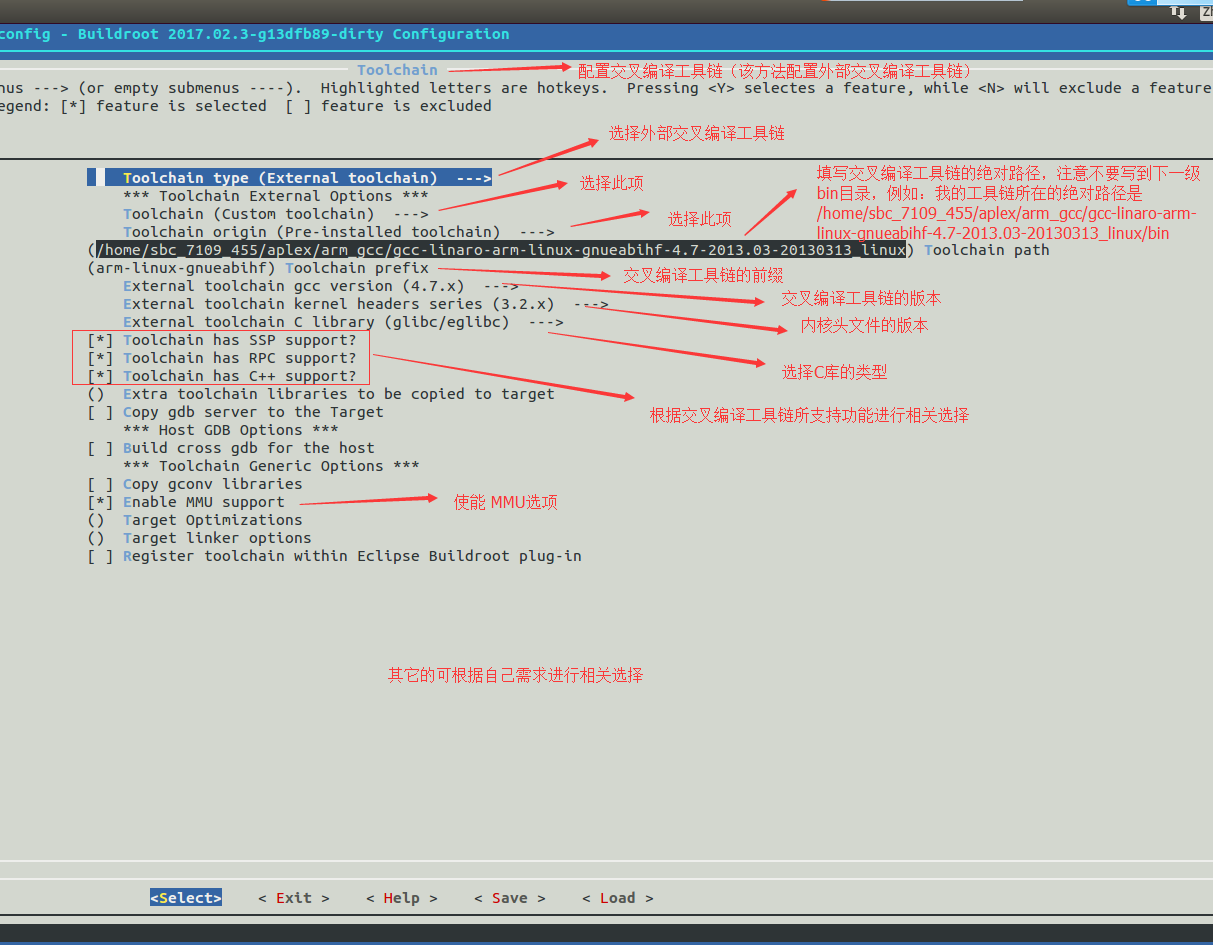


**1.3.1 配置开发板架构，如下图所示**

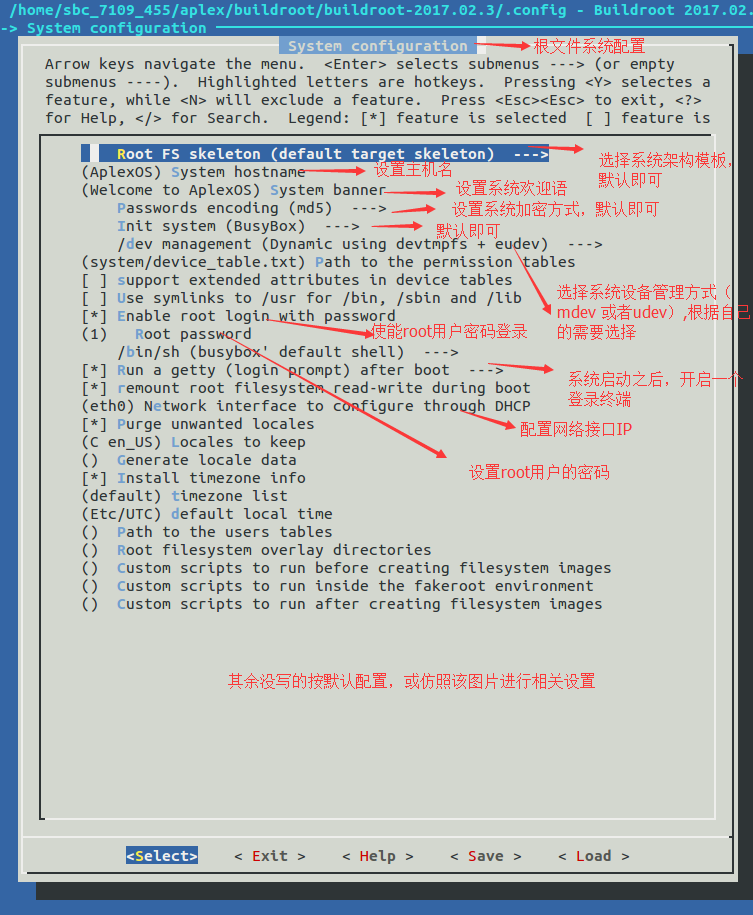


处理器架构配置示意图

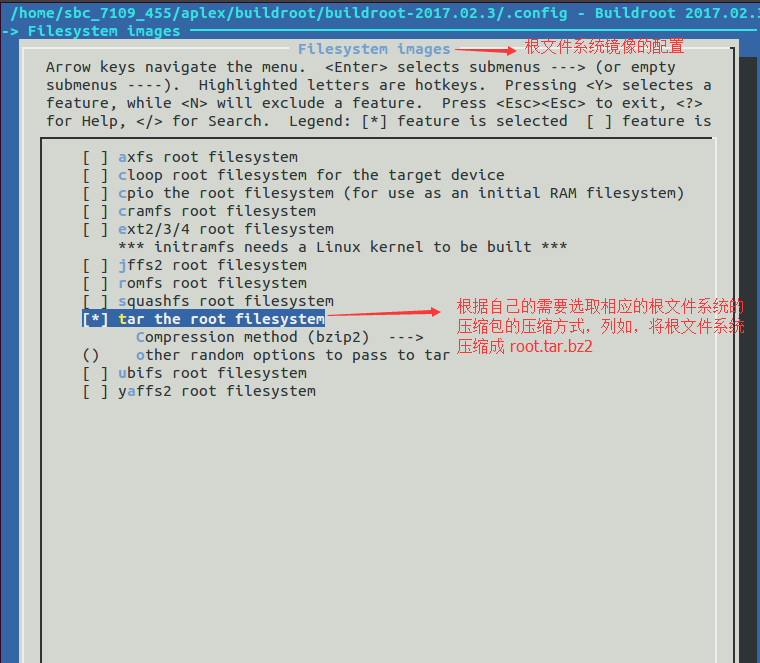
**1.3.2 配置外部交叉编译工具链**



**1.3.3 配置 根文件系统选项**



**1.3.4根文件系统压缩包的压缩方式**



**二：制作ubi文件系统**

**2.1 必备工具安装**

**sudo apt-get install kpartx**

**sudo apt-get install mtd-utils**

**2.2 ：制作200M大小的IMG文件系统**

**2.2.1**

使用dd命令来来创建一个具有确定大小的img文件my.img

**sudo dd if=/dev/zero of=my.img bs=1M count=200**

**2.2.2**

使用losetup命令将此文件转化为loop device(虚拟块设备)

**sudo losetup -f --show my.img** （它显示你挂载到那个设备节点下，一般为 /dev/loop0 ,1 ,2, 3...7）

**2.2.3**

通过 fdisk 命令对磁盘文件进行分区（以下假设块设备是/dev/loop0）（第一分区用来存放 ubi.img, uimage,MLO,u-boot.img这四个文件，所以第一分区可以根据实际需要去选择合适的大小）

**sudo fdisk /dev/loop0**

Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel

Building a new DOS disklabel with disk identifier 0x2e7df78e.

Changes will remain in memory only, until you decide to write them.

After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): **n**  # 新建一个分区

Partition type:

p primary (0 primary, 0 extended, 4 free)

e extended

Select (default p): p # 主分区

Partition number (1-4, default 1): **1**  # 分区号

First sector (2048-409599, default 2048): # 默认2048

Using default value 2048

Last sector, +sectors or +size{K,M,G} (2048-409599, default 409599): **+30M**

# 给第一个分区30M大小

Command (m for help): **n**  # 新建一个分区

Partition type:

p primary (1 primary, 0 extended, 3 free)

e extended

Select (default p): **p**  # 主分区

Partition number (1-4, default 2): # 使用默认的2

Using default value 2

First sector (63488-409599, default 63488): # 使用默认值

Using default value 63488

Last sector, +sectors or +size{K,M,G} (63488-409599, default 409599): # 直接到结束

Using default value 409599

Command (m for help): **t**  # 改变分区

Partition number (1-4): **1** # 改变第几个分区

Hex code (type L to list codes): **e** # 改为FAT16类型的文件系统

Changed system type of partition 1 to e (W95 FAT16 (LBA))

Command (m for help): **a**  # 增加boot 属性

Partition number (1-4): **1**  # 指定第一个分区增加boot 属性

ommand (m for help): **w**  # 保存相关信息

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 22: Invalid argument.

The kernel still uses the old table. The new table will be used at

the next reboot or after you run partprobe(8) or kpartx(8)

WARNING: If you have created or modified any DOS 6.x

partitions, please see the fdisk manual page for additional

information.

Syncing disks.

**2.2.4**

使用kpartx命令来绑定磁盘分区

**sudo kpartx -av /dev/loop0**

**2.2.5**

对分区进行格式化（绑定文件系统）（img文件系统已经制作完毕）

**sudo mkfs.vfat -n "boot" -F 16 /dev/mapper/loop0p1**

**sudo mkfs.ext3 -L "rootfs" /dev/mapper/loop0p2**

**2.3对img文件系统进行读写操作**

（MLO/ubi.img/u-boot.img/uImage）

**2.3.1在Linux平台下：**

a：分别挂载两个分区进行操作（以分区loop0p1为例）

**sudo mount /dev/mapper/loop0p1 /mnt**

**sudo cp myfile /mnt**

(myfile 指的是 MLO/ubi.img/u-boot.img/uImage 四个文件)

**sudo umount /mnt**

b：将img文件系统解除映射

**sudo kpartx -d /dev/loop0**

**sudo losetup -d /dev/loop0**

**2.3.2：在 windows平台下：**

a :将img文件系统从Linux平台上解除映射

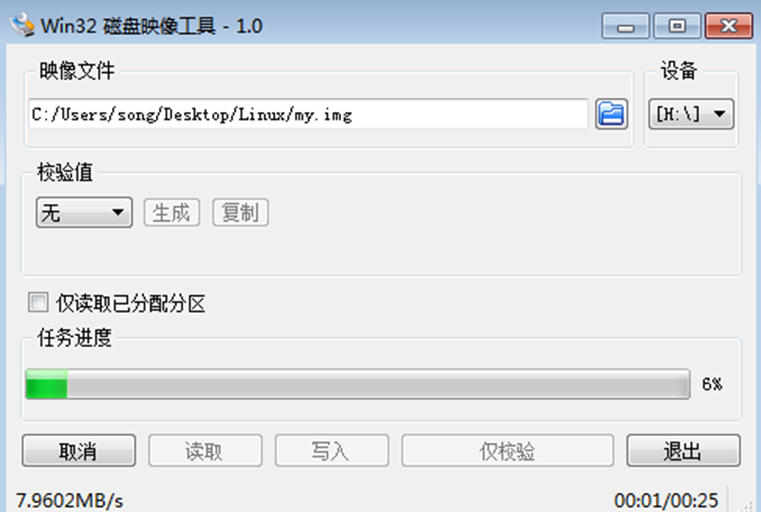
**sudo kpartx -d /dev/loop0**

**sudo losetup -d /dev/loop0**

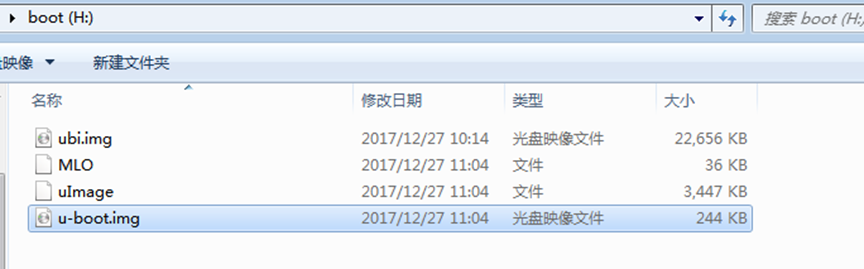
b:将img文件系统（my.img文件）拷到windows平台上，通过工具Win32DiskImager将img文件系统写到sd卡上（用此文件系统格式化sd卡），然后直接对此sd卡进行读写操作（）如下图1和图2。



图1 烧写工具界面，用my.img文件系统格式化sd卡



图二 烧录过程



图三 ：被写进去的文件

**2.4制作ubi和ubifs根文件系统**

**2.4.1: 生成ubifs文件系统**

**sudo mkfs.ubifs -q -r rootfs\_dir -m 2048 -e 126976 -c 8080 -o ubifs.img -F**

使用mkfs.ubifs工具将rootfs\_dir（路径可使用绝对路径和相对路径）目录文件格式化为ubifs.img镜像文件（是ubifs类型的镜像文件，ubifs文件系统可在u-boot下使用ubi write命令烧写到NAND FLASH上）

**2.42：将ubifs文件系统转化为ubi.img文件系统**

**sudo ubinize -o ubi.img -m 2048 -p 128KiB ubinize.cfg**

使用ubinize 工具通过ubinize.cfg 配置文件将ubifs.img镜像文件格式化为ubi文件系统（可直接在FLASH上烧写）

**2.4.3：ubinize.cfg配置文件如下**（在需要的地方可由程序员创建）：

**[ubifs]**  //要被格式化的原镜像文件的类型（如ubifs.img的文件系统类型是ubifs）

**mode=ubi** //固定格式

**image=ubifs.img** //mkfs.ubi工具使用的源文件

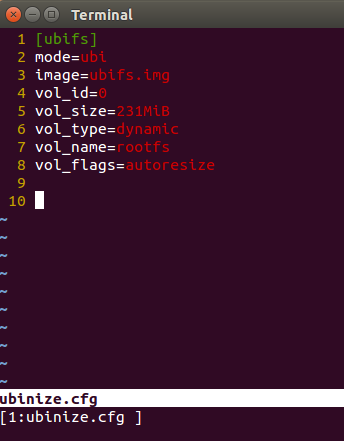
**vol\_id=0** //卷序号

**vol\_size=231MiB** //卷大小

**vol\_type=dynamic** //动态卷

**vol\_name=rootfs** //卷名

**vol\_flags=autoresize**



**2.4.4：rootfs目录来源（举列说明）**

1：在/home/sbc\_7109\_455/aplex/filesystem 创建rootfs目录文件

**mkdir rootfs**

2: 将/home/sbc\_7109\_455/aplex/buildroot/buildroot-2017.02.3/output/images 目录下的 rootfs.tar 文件解压到rootfs目录下

sudo tar –xvf /home/sbc\_7109\_455/aplex/buildroot/buildroot-2017.02.3/output/images/rootfs.tar -C /home/sbc\_7109\_455/aplex/filesystem/rootfs

（注意，不要直接使用output/target 目录制作ubifs镜像文件系统，因为这样做权限不够，平板启动不了，output/image文件夹下的两个文件都可以，原因是buildroot将image目录下的文件做了相关权限处理，但是若果非要以target目录来制作ubifs镜像文件系统，可以在buildroot编译根文件系统时添加上sudo ，也就是sudo make ）

**三：编译tslib**

Tslib 是QT触摸程序所使用到的库，手动编译tslib方法如下。

**3.1手动编译tslib**

**3.1.1下载tslib(1.5)软件源码**

**git clone https://github.com/scw-92/tslib.git**

**3.1.2在Ubuntu上安装一些工具包**

**sudo apt-get install autoconf automake libtool**

**3.1.3解压源码，进入解压的目录，执行如下命令**

**cd tslib**

创建auto\_configure.sh 脚本文件，内容如下：

**Vi auto\_configure.sh**

**echo "ac\_cv\_func\_malloc\_0\_nonnull=yes" >daiq\_tslib.cache**

**./autogen.sh**

**./configure \**

**--prefix=/home/sbc\_7109\_455/aplex/qt\_arm/tslib/tslib\_install \**

**--host=arm-linux-gnueabihf**

**make && make install**

**3.1.4 安装**

编译生成的库，头文件等都被拷贝到 prefix 指定的路径中。如果可以看到该指定的路径下有 4 个文件夹：/bin、/etc、/lib、/include，则表示安装完成。

将安装目录（prefix指定的目录）下所有文件打包移植到开发板上。

**cd /home/sbc\_7109\_455/aplex/qt\_arm/tslib/tslib\_install**

**tar cvf tslib.tar bin/ etc/ include/ lib/ share/**

在开发板上创建如下目录 ：

**mkdir /usr/local** //如果已存在无需重建

将tslib.tar 拷贝到该目录下做如下操作

**cd /usr/local/**

**mkdir tslib**

**tar xvf tslib.tar -C tslib/**

**root@OS:/usr/local/tslib # ls**

**bin etc include lib share**

**3.1.5 配置**

定制输入模块

**vi /usr/local/tslib/etc/ts.conf**

找到这一行module\_raw input，在其下面添加如下内容：

**module pthres pmin=1**

**module variance delta=30**

**module dejitter delta=100**

**module linear**

设置开发板环境，在/etc/profile文件的末尾添加上如下条目

**vi /etc/profile**

**export TSLIB\_ROOT=/usr/local/tslib**

**export TSLIB\_TSDEVICE=/dev/input/event0**

**export TSLIB\_CALIBFILE=/etc/pointercal**

**export TSLIB\_CONFFILE=$TSLIB\_ROOT/etc/ts.conf**

**export TSLIB\_PLUGINDIR=$TSLIB\_ROOT/lib/ts**

**export TSLIB\_FBDEVICE=/dev/fb0**

**export TSLIB\_CALIBFILE=/etc/pointercal**

**export POINTERCAL\_FILE=/etc/pointercal**

**export TSLIB\_CONSOLEDEVICE=none**

**export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/tslib/lib**

**export PATH=$TSLIB\_ROOT/bin:$PATH**

其中 TSLIB\_ROOT 更改为自己实际存放的 tslib 的绝对路径。

TSLIB\_TSDEVICE 和 QWS\_MOUSE\_PROTO 这两项需要查看自己的开发板触摸屏设备对应/dev/input/下的文件。

**3.1.6执行测试命令**

重新启动开发板，使系统重新读取/etc/profile 的环境变量，进入 tslib/bin 目录，执行如下命令：

**cd /usr/local/tslib/bin**

**./ts\_calibrate**

校正触摸屏执行ts\_test测试程序，看看校正效果

**./ts\_test**

**四：手动编译QT5.6**

**4.1 安装pc机 qt5.6开发软件**

安装一些必要软件

**sudo apt-get install mesa-common-dev**

**sudo apt-get install libglu1-mesa-dev –y**

**sudo apt-get install libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev**

ubuntu下载：

**wget** <http://download.qt.io/archive/qt/5.6/5.6.0/qt-opensource-linux-x64-5.6.0.run>

Ubuntu 安装：

**chmod +x qt-opensource-linux-x64-5.6.0.run**

**./qt-opensource-linux-x64-5.6.0.run**

Ubuntu 配置：

**vi /usr/lib/x86\_64-linux-gnu/qt-default/qtchooser/default.conf**

添加如下内容（可删除原有内容，没有该文件，则新建该文件）

**/home/sbc\_7109\_455/qt\_ubuntu/5.6/gcc\_64/bin** //指定qmake 路径

**/usr/lib/x84\_64-linux-gnu** //指定pc机编译器路径

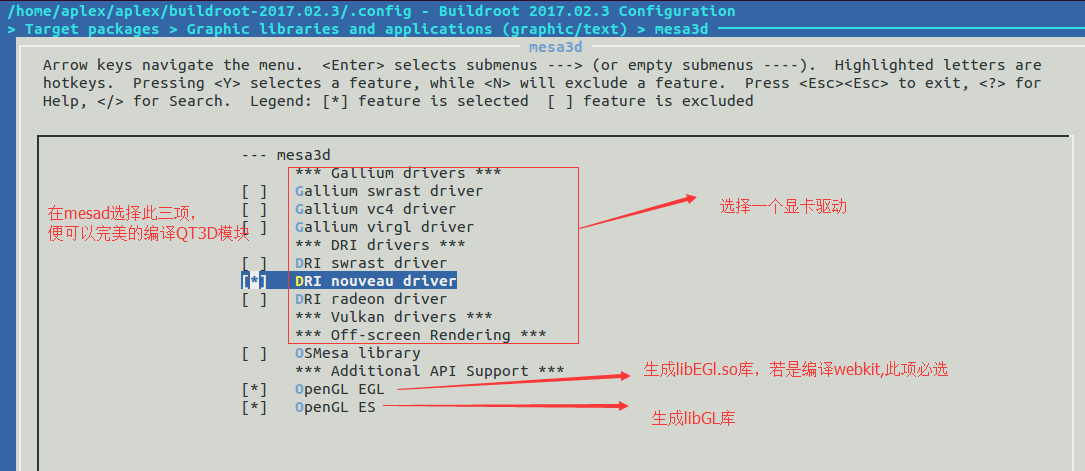
Ubuntu 配置效果

**qmake -v**

QMake version 3.0

Using Qt version 5.6.0 in /home/sbc\_7109\_455/qt\_ubuntu/5.6/gcc\_64/lib

**4.1用buildroot编译QT支持3d编译效果：**

****

如上图所示，只要选择了上述3个条件。便是满足了编译QT3d模块的条件

必选项：BR2\_PACKAGE\_MESA3D\_DRI\_DRIVER\_NOUVEAU

根据qt5canvas3d--->BR2\_PACKAGE\_QT5\_GL\_AVAILABLE--->BR2\_PACKAGE\_HAS\_LIBGLES （EABIHF架构），找到BR2\_PACKAGE\_HAS\_LIBGLES 之后，根据BR2\_PACKAGE\_HAS\_LIBGLES的满足条件，可以发现需要勾选下面四项内容：BR2\_PACKAGE\_MESA3D\_OPENGL\_ES 、

BR2\_PACKAGE\_MESA3D 、

BR2\_PACKAGE\_MESA3D\_DRIVER 、

BR2\_TOOLCHAIN\_USES\_GLIBC 、

BR2\_ARM\_EABIHF、

然后就可以在buildroot上面编译QT的3d库了

用buildroot 编译QT选择eglfs support ，需要做以下条件：

选择 [\*] mesa3d --->

[\*] DRI nouveau driver

[\*] OpenGL EGL //此选项会编译处libELG.so库文件。此库是底层显卡驱动与上层窗口系统之间的接口。

[\*] Qt5 --->

OpenGL support

QT选择编译directfb support (NEW) 需要先选择

Graphic libraries and applications (graphic/text) --->

[\*] directfb

QT选择编译qt5connectivity 模块，需要先选择下面3向的任意一项

BR2\_PACKAGE\_NEARD || BR2\_PACKAGE\_BLUEZ\_UTILS || BR2\_PACKAGE\_BLUEZ5\_UTILS

QT选择编译X.org XCB support (NEW) 需要先选择

[\*] xorg-server

X Window System server version (1.19.1) --->

X Window System server type (Modular X.org) --->

**4.2手动编译移植ARM板qt**

安装必备软件：sudo apt-get install g++

下载qt5.6源码

wget <http://download.qt.io/archive/qt/5.6/5.6.0/single/qt-everywhere-opensource-src-5.6.0.tar.gz>

解压、配置、交叉编译

（参考文件：https://github.com/scw-92/arm\_-\_tools/tree/master/qt\_config）

tar xvf qt-everywhere-opensource-src-5.6.0.tar.gz //解压源码

cd qt-everywhere-opensource-src-5.6.0/ //进入源码顶层目录

vi qtbase/mkspecs/linux-arm-gnueabi-g++/qmake.conf //修改qt编译的默认配置文件

#

# qmake configuration for building with arm-linux-gnueabihf-g++

#

**MAKEFILE\_GENERATOR = UNIX**

**CONFIG += incremental**

**QMAKE\_INCREMENTAL\_STYLE = sublib**

**QT\_QPA\_DEFAULT\_PLATFORM = linuxfb**

**QMAKE\_CFLAGS\_RELEASE += -O2 -march=armv7-a**

**QMAKE\_CXXFLAGS\_RELEASE += -O2 -march=armv7-a**

//指定开发板架构

**include(../common/linux.conf)**

**include(../common/gcc-base-unix.conf)**

**include(../common/g++-unix.conf)**

**QMAKE\_INCDIR +=/home/sbc\_7109\_455/aplex/qt\_arm/tslib/tslib\_install/include**

**QMAKE\_LIBDIR +=/home/sbc\_7109\_455/aplex/qt\_arm/tslib/tslib\_install/lib**

//上面两项指定tslib的头文件和库文件，如果不需要tslib，则不用加上面两项

# modifications to g++.conf

**QMAKE\_CC = arm-linux-gnueabihf-gcc**

**QMAKE\_CXX = arm-linux-gnueabihf-g++**

**QMAKE\_LINK = arm-linux-gnueabihf-g++**

**QMAKE\_LINK\_SHLIB = arm-linux-gnueabihf-g++**

//换成自己的交叉编译工具链

# modifications to linux.conf

**QMAKE\_AR = arm-linux-gnueabihf-ar cqs**

**QMAKE\_OBJCOPY = arm-linux-gnueabihf-objcopy**

**QMAKE\_NM = arm-linux-gnueabihf-nm -P**

**QMAKE\_STRIP = arm-linux-gnueabihf-strip**

**load(qt\_config)**

//换成自己的交叉编译工具链

**vi my\_auto.sh** //在qt5.6源码顶层目录下创建该自动配置脚本，在该文件中添加如下内容

**#!/bin/sh**

**./configure \**

**-prefix /home/sbc\_7109\_455/aplex/qt\_arm/qt\_arm\_insatll \** //指定安装目录

**-release \**

**-opensource \**

**-confirm-license \**

**-make libs \**

**-xplatform linux-arm-gnueabi-g++ \**

**-optimized-qmake \**

**-linuxfb \**

**-pch \**

**-qt-sql-sqlite \**

**-qt-zlib \**

**-qt-libpng \**

**-tslib \**

**-no-opengl \**

**-no-sse2 \**

**-no-openssl \**

**-no-nis \**

**-no-cups \**

**-no-glib \**

**-no-dbus \**

**-no-xcb \**

**-no-xcursor -no-xfixes -no-xrandr -no-xrender \**

**-no-separate-debug-info \**

**-make examples -nomake tools -nomake tests -no-iconv**

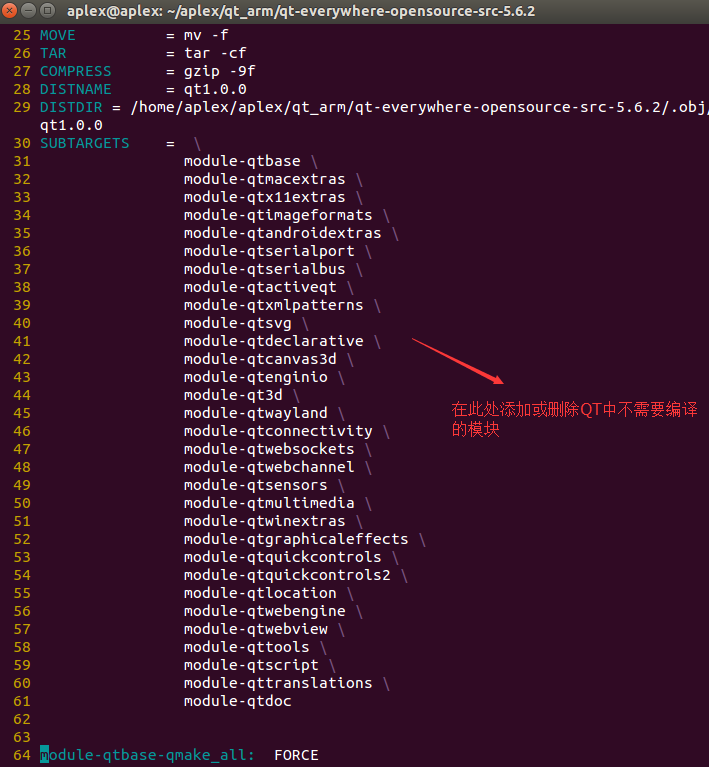
**---------------------------------------------------------------------------------------------------------------------------------**

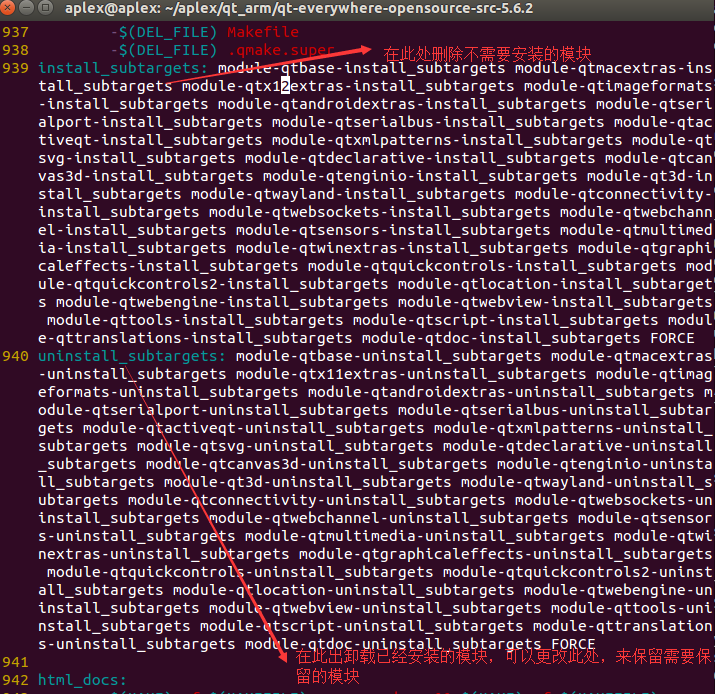
**chmod 777 my\_auto.sh** //修改my\_auto.sh脚本文件的执行权限

**./my\_auto.sh** //执行该脚本

**make -j4 && make install**  //不要使用make，否则可能会报错

Make install 中途会出错，那些东西没有编译，修改办法，在QT顶层目录里面的Makefile中的修改如下几项，具体如下图所示：





将编译好的相关库和插件（examples/ lib/ plugins/三个文件夹）打包移植到开发板上

**cd /home/sbc\_7109\_455/aplex/qt\_arm/qt\_arm\_insatll** //进入到交叉编译指定的安装目录

**tar cvf qt5.tar examples/ lib/ plugins/** 打包成qt5.tar

在开发板上创建如下目录 ：

**mkdir /usr/local** //如果已存在无需重建

将qt5.tar拷贝到该目录下做如下操作

**cd /usr/local/**

**mkdir qt5**

**tar xvf qt5.tar -C qt5/**

**root@AplexOS:/usr/local/qt5 # ls**

**examples lib plugins**

在开发板上配置 qt5.6

**vi /etc/profile** //在文件末尾添加如下内容

**export QTDIR=/usr/local/qt5**

**export QT\_QPA\_PLATFORM\_PLUGIN\_PATH=/usr/local/qt5/plugins**

**export QT\_QPA\_PLATFORM=linuxfb:fb=/dev/fb0**

**export QT\_QPA\_FONTDIR=/usr/local/qt5/lib/fonts**

**export QT\_QPA\_GENERIC\_PLUGINS=tslib:$TSLIB\_TSDEVICE**

**export LD\_LIBRARY\_PATH=/usr/local/qt5/lib:$LD\_LIBRARY\_PATH**

**export QT\_DEBUG\_PLUGINS=1**

**export QWS\_KEYBOARD=USB:/dev/input/event0**

**export QWS\_MOUSE\_PROTO=Tslib:/dev/input/event0**

**export QWS\_DISPLAY=LinuxFb:mmWidth91:mmHeight53:0**

**export QWS\_SIZE=800x480**

配置开发板开机自启动QT运行程序

**vi /etc/init.d/S90aplex\_ts** //创建开机自启动文件（内容如下）

**#! /bin/sh**

**export TSLIB\_ROOT=/usr/local/tslib**

**export TSLIB\_TSDEVICE=/dev/input/event0**

**export TSLIB\_CALIBFILE=/etc/pointercal**

**export TSLIB\_CONFFILE=$TSLIB\_ROOT/etc/ts.conf**

**export TSLIB\_PLUGINDIR=$TSLIB\_ROOT/lib/ts**

**export TSLIB\_FBDEVICE=/dev/fb0**

**export TSLIB\_CALIBFILE=/etc/pointercal**

**export POINTERCAL\_FILE=/etc/pointercal**

**export TSLIB\_CONSOLEDEVICE=none**

**export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/tslib/lib**

**export PATH=$TSLIB\_ROOT/bin:$PATH**

**export QTDIR=/usr/local/qt5**

**export QT\_QPA\_PLATFORM\_PLUGIN\_PATH=/usr/local/qt5/plugins**

**export QT\_QPA\_PLATFORM=linuxfb:fb=/dev/fb0**

**export QT\_QPA\_FONTDIR=/usr/local/qt5/lib/fonts**

**export QT\_QPA\_GENERIC\_PLUGINS=tslib:$TSLIB\_TSDEVICE**

**export LD\_LIBRARY\_PATH=/usr/local/qt5/lib:$LD\_LIBRARY\_PATH**

**export QT\_DEBUG\_PLUGINS=1**

**export QWS\_KEYBOARD=USB:/dev/input/event0**

**export QWS\_MOUSE\_PROTO=Tslib:/dev/input/event0**

**export QWS\_DISPLAY=LinuxFb:mmWidth91:mmHeight53:0**

**export QWS\_SIZE=800x480**

**if [-e /etc/pointercal ]**

**then**

**rm /etc/pointercal**

**/usr/local/tslib/bin/ts\_calibrate &**

**else**

**/usr/local/tslib/bin/ts\_calibrate &**

**fi**

**/test/qt\_tools &**//qt\_tools 是我的一个QT程序，&表示后台运行，使其不占用控制台窗口

**chmod 777 /etc/init.d/S90aplex\_ts** //修改该文件的执行权限

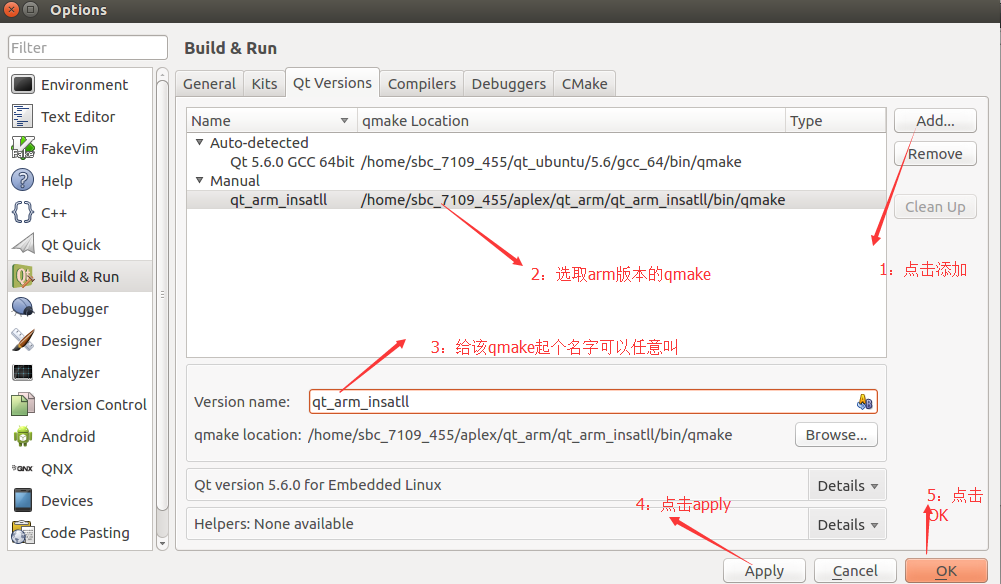
**五：搭建QT5.6的编译环境**

**5.1在Ubuntu上配置qt编译arm格式的qt程序**

配置Ubuntu 上 qt 的qmake（该qmake是交叉编译qt时产生的可以编译arm 版本的Makefile的qmake）

**5.1.1配置qmake**

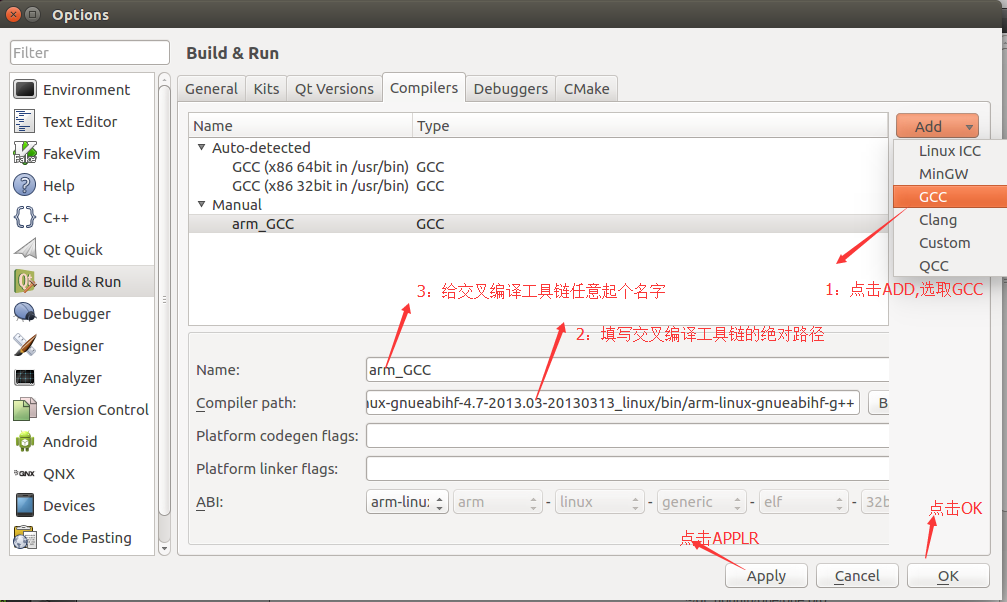
首先打开QtCreator->Tools->Options->Build &Run -> Qt Version如下图所示



点击add在version name输入名字(随便写一个)，在 qmake location 中填入ARM版本qmake的绝对路径。

**5.1.2配置交叉编译工具链**

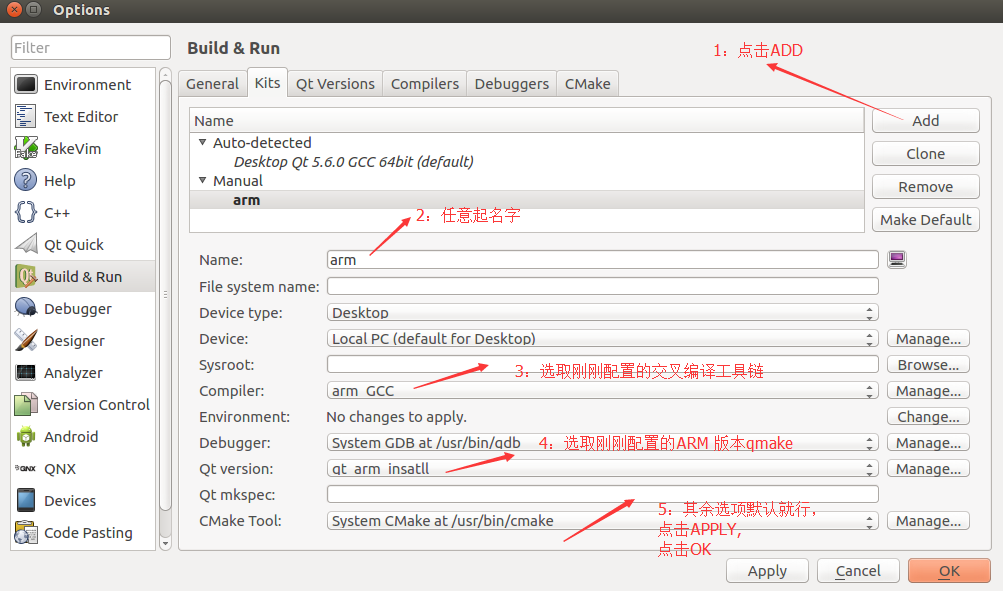
首先打开QtCreator->Tools->Options->Build &Run -> Compieirs如下图所示



点击add->gcc，添加交叉编译工具链，在Name中输入名字，Complier path中输入你的交叉编译工具链在linux上的绝对路径

**5.1.3 配置Kits工具**

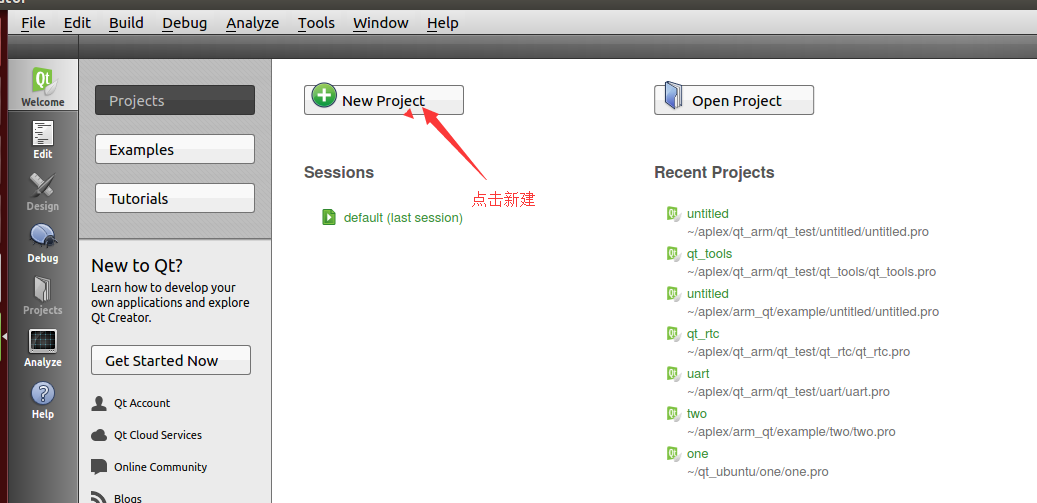
首先打开QtCreator->Tools->Options->Build &Run -> Kits如下图所示



点击Add、在Complier处选择刚才配置好的交叉编译工具链的名字，在Qt Version处选择刚才配置好的qmake的名字，点击apply，OK。

**5.2 在PC机上编译一个QT小实例**

**5.2.1 QT-UI编程**



图一

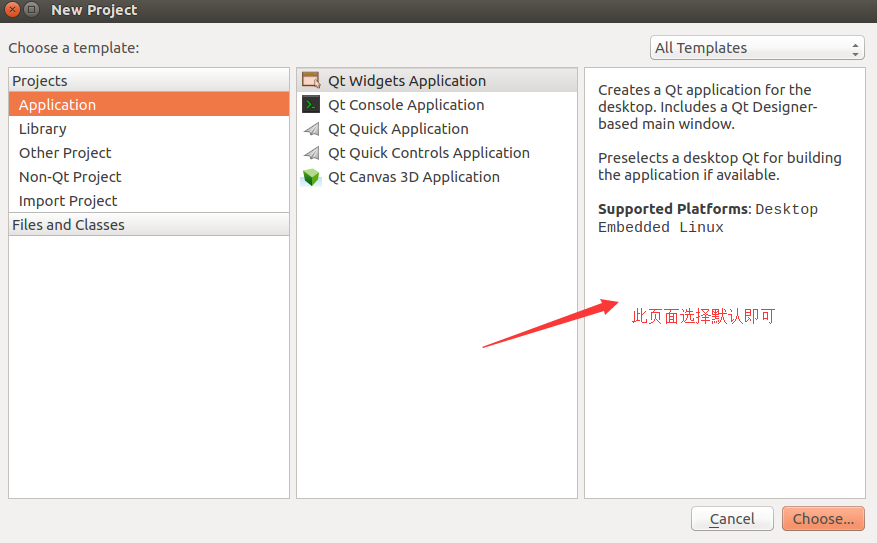


图2

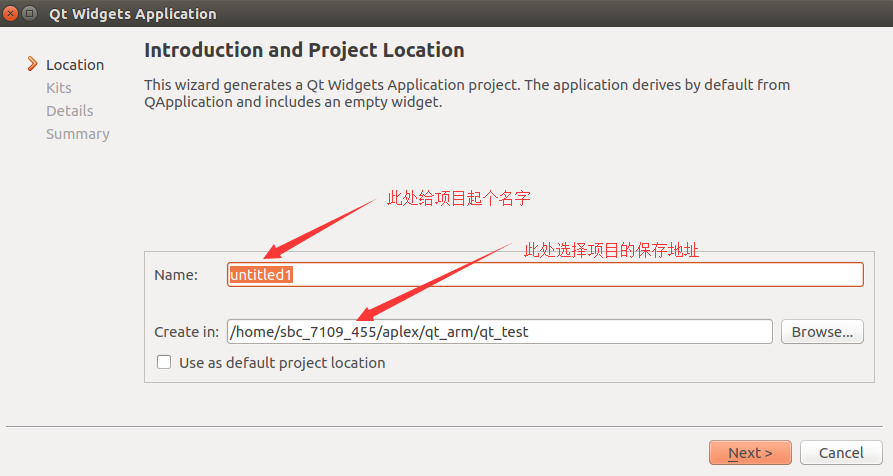


图 3

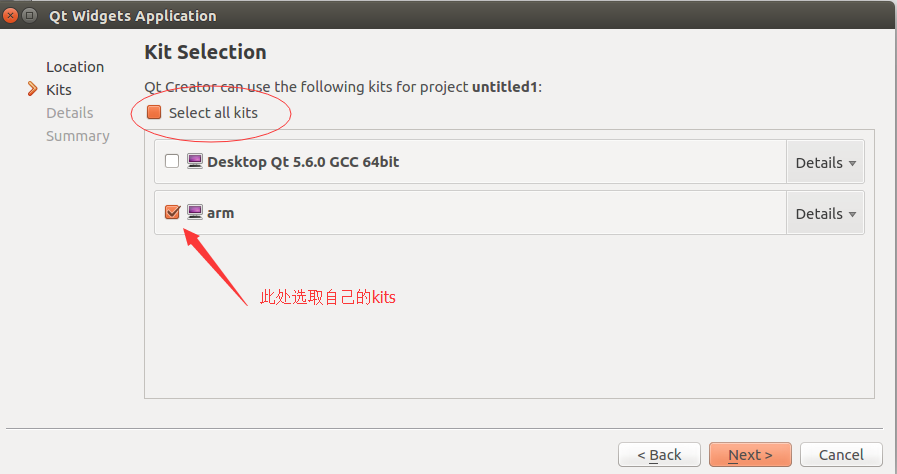


图4

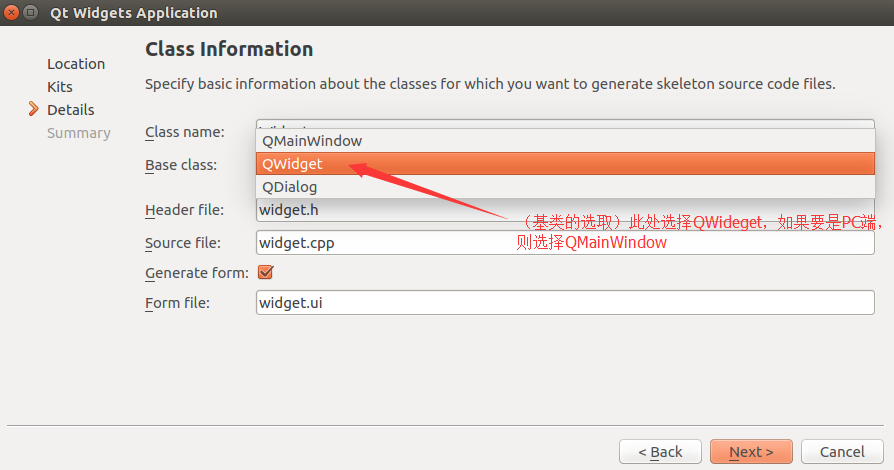


图5

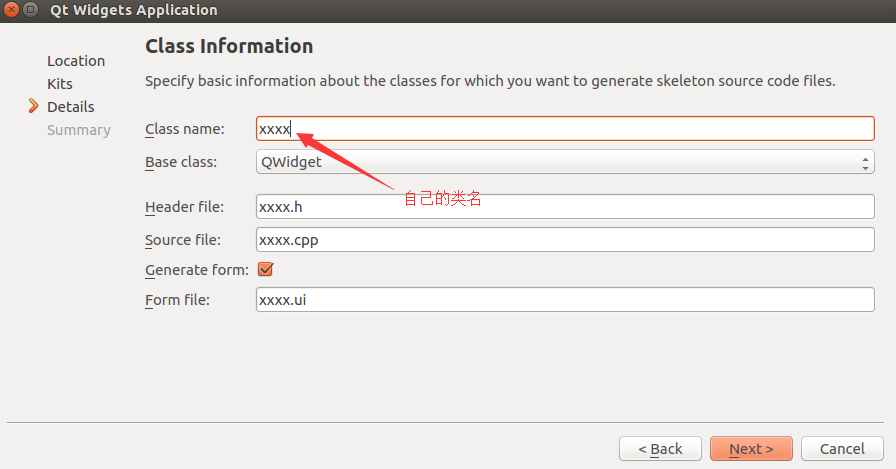


图 6

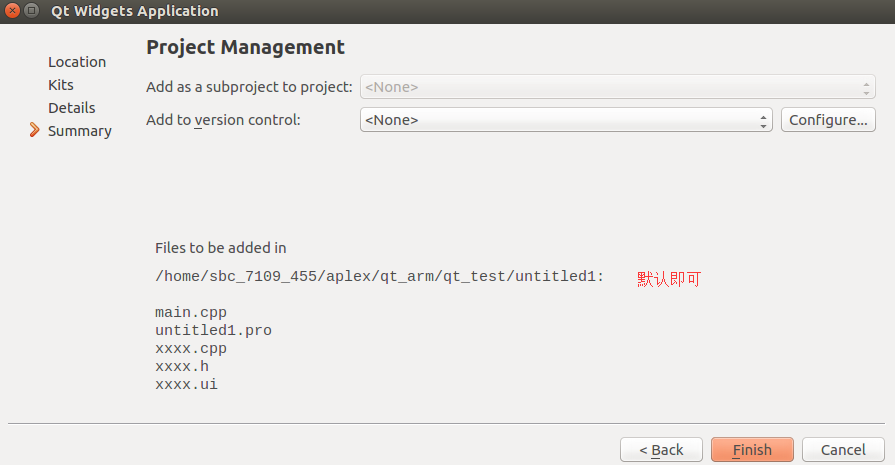


图 7

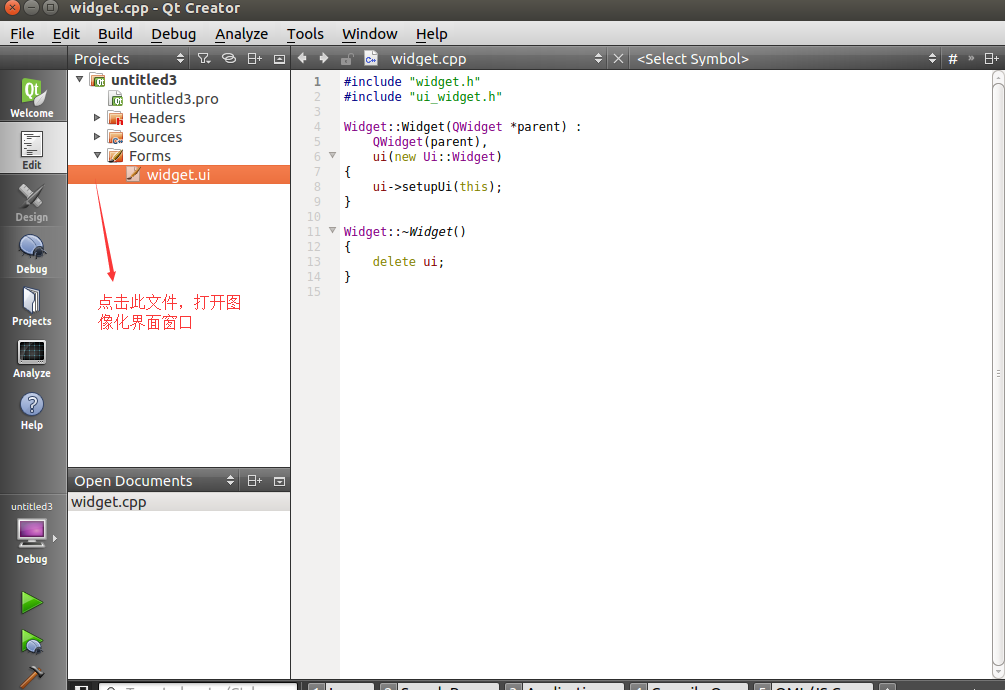


图 8

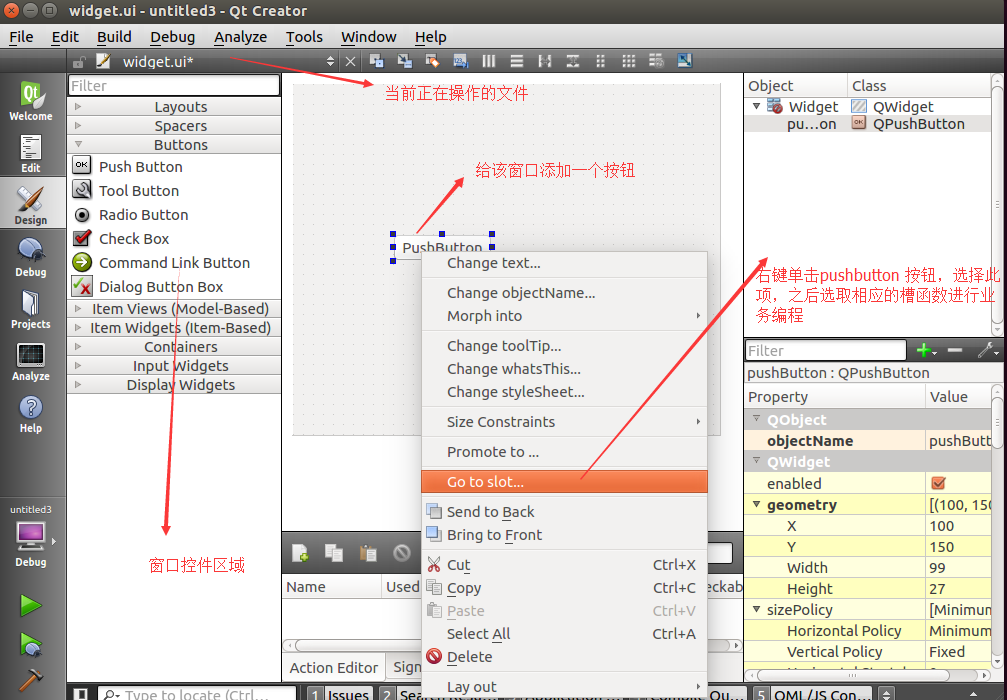


图 9

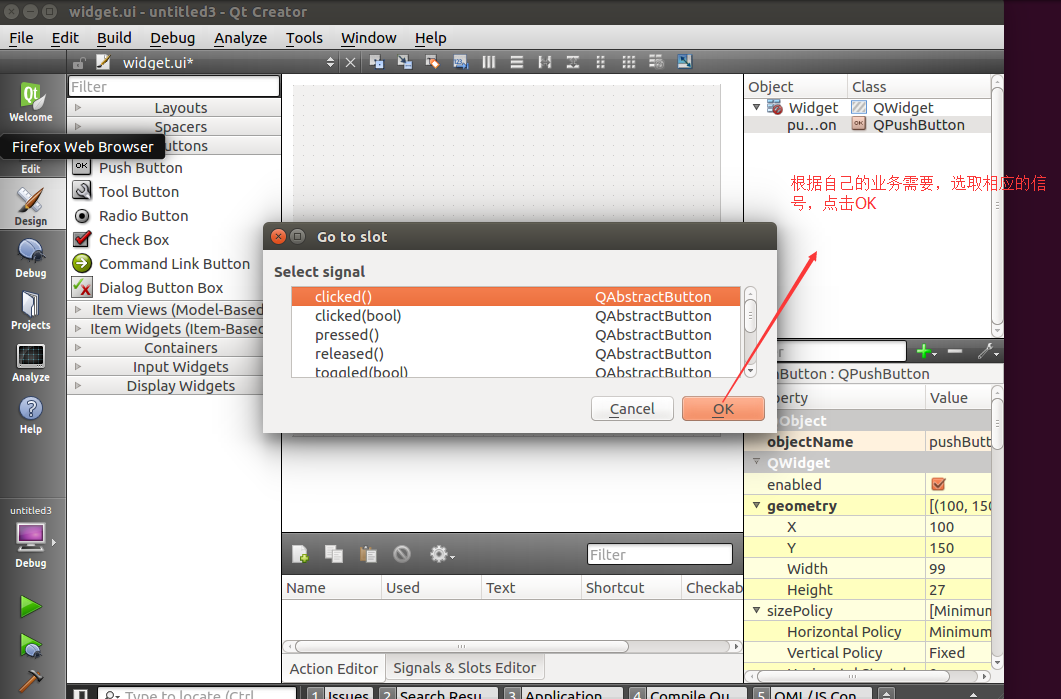
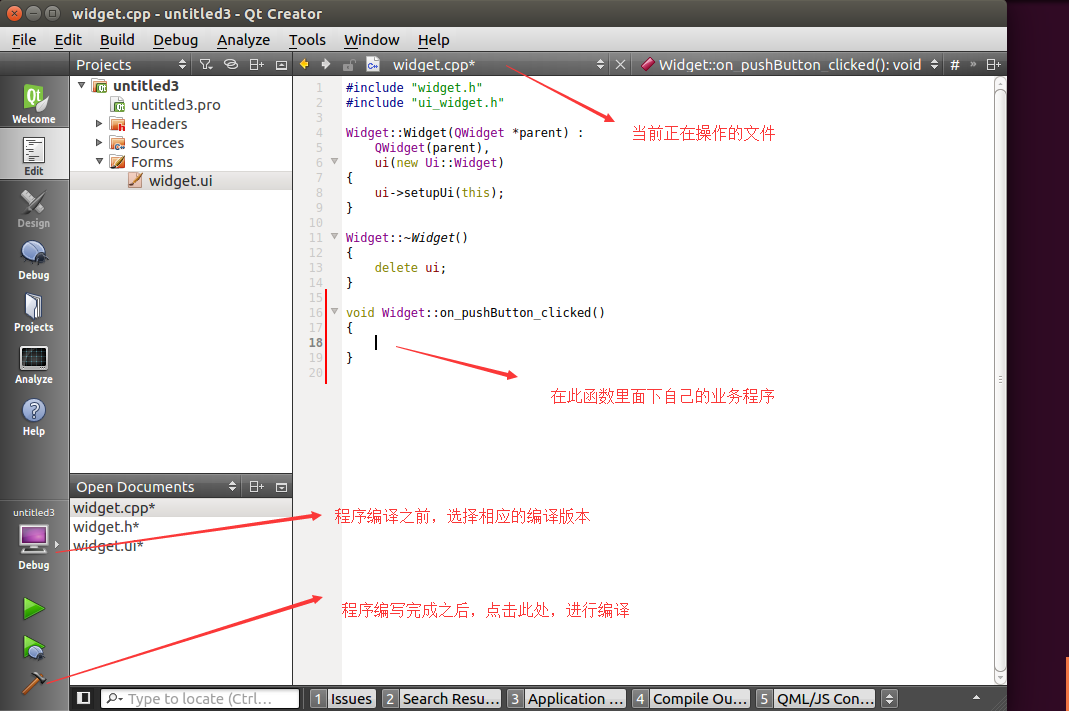


图 10



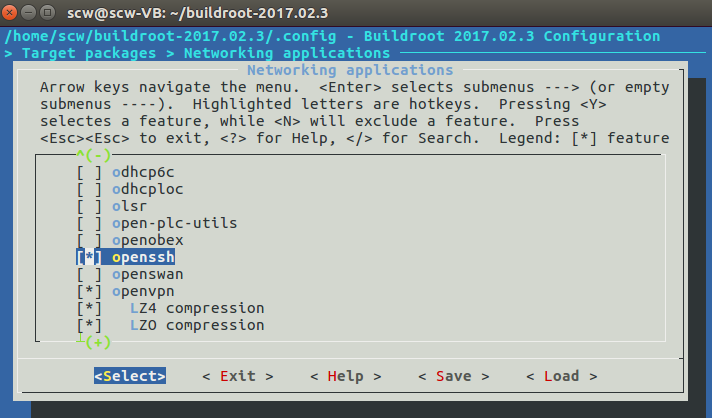
程序编译完成之后，将其拷贝到开发板上，试运行即可（编译后的程序在该项目所在的同级目录下，图三中项目保存的地址 如：/home/sbc\_7109\_455/aplex/qt\_arm/qt\_test/build-xxx-arm-Debug）

**六：opensshd服务的配置**

**6.1 openssh 的移植**

使用buildroot生成需求

**make menuconfig** 打开 openssh 选项如下图



**6.2 配置openssh的配置文件**

Openssh作为服务器的配置文件是 /etc/ssh/ sshd\_config (ssh\_config

poenssh作为客户端的配置文件)，打开/etc/ssh/ sshd\_config做如下修改，即可满足opensshd的最低要求配置。

**Vi /etc/ssh/sshd\_config**

**Port 22**

**AddressFamily any**

**ListenAddress 0.0.0.0**

**ListenAddress ::**

**PermitRootLogin yes**

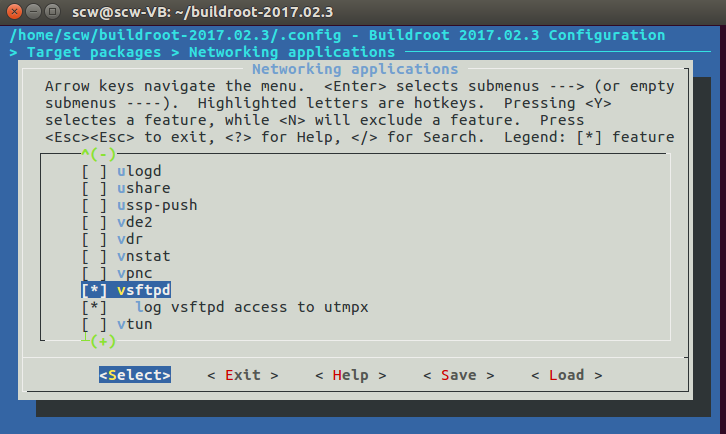
**PasswordAuthentication yes**

**七：vsftpd服务的配置与使用**

**7.1 vsftpd 的移植**

使用buildroot生成需求

**make menuconfig** 打开 Vsftpd 选项如下图



**7.2 vsftpd使用前的准备**

a) **useradd nobody**  添加nobody用户

b) **mkdir /usr/share/empty/** 创建一个empty目录

c) **mkdir /var/ftp/**  创建一个ftp目录

d) **useradd -d /var/ftp ftp**（创建用户FTP，匿名用户会映射到服务器的ftp用户）

e) **chown root.root /var/ftp**

f) **chmod og-w /var/ftp** （去掉ftp目录的写权限）

**7.3配置Vsftpd的配置文件**

Vsftpd作为服务器的配置文件是 /etc/vsftpd.config，打开/etc/Vsftpd做如下修改，即可满足vsftpd的最低要求配置。

**vi /etc/vsftpd.conf**

**anonymous\_enable=YES**

**local\_enable=YES**

**write\_enable=YES**

**local\_umask=022**

**anon\_root=/var/ftp**  匿名用户的根目录

**anon\_upload\_enable=YES**

**anon\_mkdir\_write\_enable=YES**

**listen=YES**  采用ipv4

**7.4 使用vsftpd**

客户端输入：**ftp IP** ；

客户端输入账号密码登陆成功；

Root用户登录：root

匿名用户登录：anonymous

**7.5更改vsftpd的数据传输模式为被动模式**（passive）

客户端输入账号密码登陆成功之后，客户端命令行输入passive 将主动模式切换为被动模式，之后进行vsftpd功能的使用。

**7.6 修改源码**

若vsftpd 源码编译的软件无法在arm板上进行文件传输，则将vsftpd源码中关于权限验证的代码注释掉。

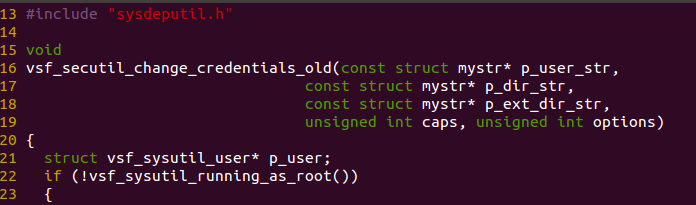
为了能够在arm板使用vsftpd的功能，需要更改源码，为了避免麻烦，直接将vsftpd中的关于验证的部分去掉

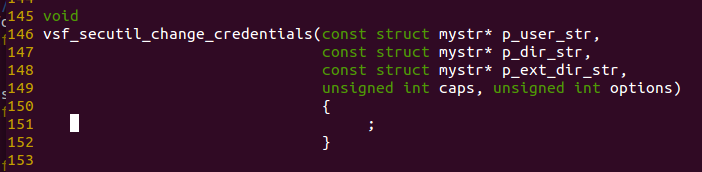
**cd buildroot-2017.02.3/output/build/vsftpd-3.0.3/**

**rm .stamp\_built .stamp\_target\_installed**

**vi secutil.c** 做如下修改

将原先的vsf\_secutil\_change\_credentials函数名改为vsf\_secutil\_change\_credentials\_old，然后从新构建该函数（新函数的函数体为空即可），如下图，重新编译即可



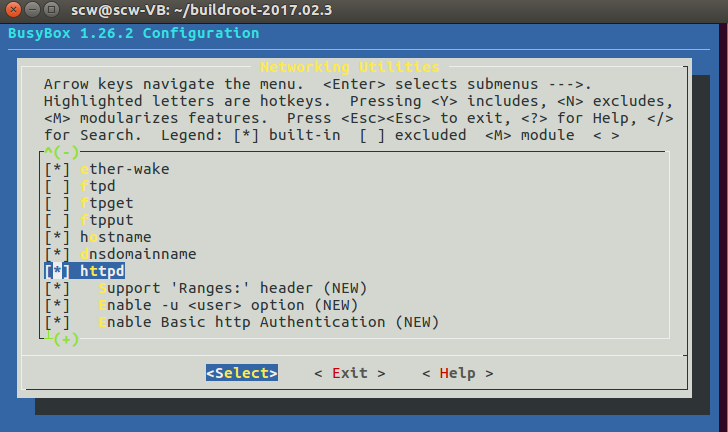


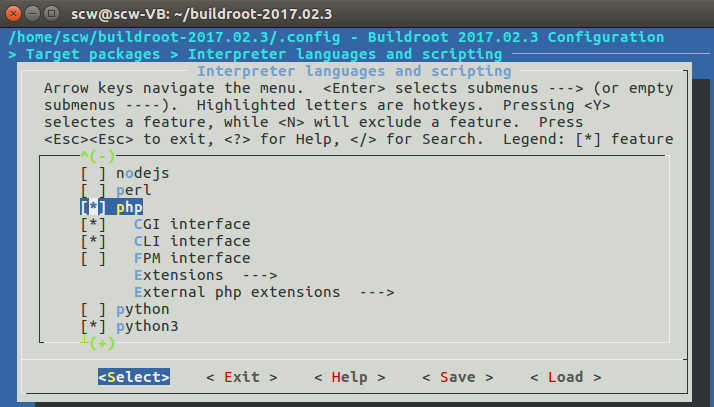
**八httpd服务的配置与使用**

**8.1 httpd的移植**

使用buildroot生成需求

**make busybox-menuconfig** 打开 httpd 选项如下图

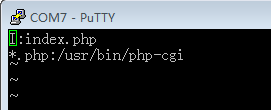
**make menuconfig** 打开PHP选项



**8.2 配置相关配置文件**

**8.2.1 创建配置文件**

在/etc 目录下（开发板）创建http的配置文件 httpd.conf，在httpd.conf文件中添加下面代码



**I:index.php** ：指定httpd的默认网页主页文件

**\*.php:/usr/bin/php-cgi** ：指定PHP文件的解释器

**8.2.2 修改/etc/php.ini**

**cgi.force\_redirect = 0**

**cgi.redirect\_status\_env ="yes";**

并且将这两句前面注释号去掉

**8.2.3 创建启动脚本**

在 /etc/init.d/ 下面创建httpd的启动脚本S71httpd



启动脚本内容如下：

**#! /bin/sh**

**set -e**

**DESC="httpd"**

**NAME=httpd**

**DAEMON=/usr/sbin/$NAME**

**case "$1" in**

**start)**

**printf "Starting $DESC: "**

**start-stop-daemon -S -b -x $NAME -- -h /var/www/SettingsPage-master/src**

**echo "OK"**

**;;**

**stop)**

**printf "Stopping $DESC: "**

**start-stop-daemon -K -x $NAME**

**echo "OK"**

**;;**

**restart|force-reload)**

**echo "Restarting $DESC: "**

**$0 stop**

**sleep 1**

**$0 start**

**echo ""**

**;;**

**\*)**

**echo "Usage: $0 {start|stop|restart|force-reload}" >&2**

**exit 1**

**;;**

**esac**

**exit 0**

-h /var/www/SettingsPage-master/src : -h 指定服务器的家目录

**8.2.4设置静态IP**

**vi /etc/network/interfaces**

修改为如下内容：

**auto eth0**

**iface eth0 inet static**

**address 192.168.8.100**

**netmask 255.255.255.0**

**gateway 192.168.8.2**

**dns-nameserver 119.29.29.29**

**8.2.5 PHP测试**

**vi /var/www/SettingsPage-master/src/index.php**

**<?php**

**phpinfo();**

**?>**

**配置VIM编辑器**

**打开家目录下 .vimrc文件，在里面添加如下内容**

**set showcmd**

**set showmatch**

**set ignorecase**

**set smartcase**

**set incsearch**

**set autowrite**

**set hidden**

**set mouse=a**

**set nu**

**syntax on**

**set ruler**

**set showcmd**

**set cmdheight=2**

**set scrolloff=3**

**set laststatus=2**

**map <F3> :tabnew .<CR>**

**set autoread**

**set cursorline**

**set noeb**

**set confirm**

**set autoindent**

**set cindent**

**set tabstop=4**

**set report=0**

**autocmd FileType c,cpp map <buffer> <leader><space> :w<cr>:make<cr>**

PS1=‘${debian\_chroot:+($debian\_chroot)}\[\033[01;32m\]\u\[\033[00m\]@\h:\[\033[01;33m\]\w\[\033[00m\]\$ ‘

**Xorg与fluxbox移植**

fluxbox-generate\_menu (fluxbox的菜单更新工具)

X11移植的原则是：只要xorg能够运行就说明移植成功了，与x的运行程序（客户端无关，与X11R7 Applications中的手动选项无关，默认的有关,）

10.1 用buildroot编译X11编译选项，如图片所示

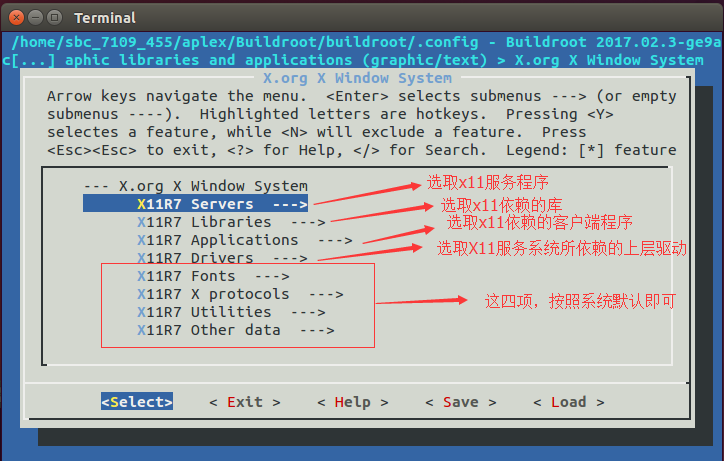


图10.1 ，X11界面介绍

功能选项（本文中没有明文列出的可以，表示可以选择默认选项）

Toolchain —>  
·[X] Enable WCHAR support //经测试此项可以不选  
·[X] Enable C++ support

X.org X Window System

X11R7 Servers --->

[\*] xorg-server

X Window System server version (1.19.1) --->

X Window System server type (Modular X.org) --->

X11R7 Libraries --->

[\*] xcb-util-cursor //经测试，此项可不选

X11R7 Applications --->（经测试，Xorg的启动可以不选则此条目下的东西，但为了可以配置xorg，所以可以根据自己的需求去选择相关运用软件）

[\*] setxkbmap

[\*] twm

-\*- xauth

[\*] xclock

[\*] xinit

[\*] xinput

-\*- xkbcomp

-\*- xmodmap

X11R7 Drivers --->（此条目下的驱动可不选，经测试，Xorg源码中自带相关驱动的源码，所以，此条目下的驱动可以不选）

[\*] xf86-input-evdev （当选择udev时可以选择此项，否则如果没有需要，不用选择此项）

[\*] xf86-input-keyboard

[\*] xf86-input-mouse

[\*] xf86-video-cirrus

[\*] xf86-video-fbdev

[\*] xf86-video-vesa

其他的杂项选择

[\*] xterm

[\*] fluxbox

(经测试，x11 的运行只需要MDEV就足够了，所以无需多此一举选择udev.)

Buildroot安装eudev(udev)的情况,首先编译mdev，然后根据需要将mdev替换成udev再重新编译一次。

10.2 配置x11，让其开机自启动

1. 在/etc/profile/文件末尾添加下面一句话：

export DISPLAY=:0

1. 创建X11的启动脚本/etc/init.d/S72X11\_fluxbox.sh，内容如下：

#!/bin/sh

set -e

cd /usr/bin/

QTAPP="startfluxbox"

QTAPPSTART="./${QTAPP} > /var/log/Xsession.log 2>&1"

case "$1" in

start)

echo "Starting ${QTAPP}"

Xorg &

export DISPLAY=:0

eval $QTAPPSTART &

;;

stop)

echo "Stopping ${QTAPP}"

killall $QTAPP

killall Xorg

;;

restart)

$0 stop

$0 start

;;

\*)

echo "usage: $0 { start | stop | restart }" >&2

exit 1

;;

esac

exit 0

10.3 跟新fluxbox的菜单，运行命令：fluxbox-generate\_menu

**virtualbox 安装 USB 扩展功能**

virtualbox 安装 USB 扩展功能，可以让虚拟机自动挂载sd卡。

11.1 参考链接：<http://www.cnblogs.com/chenfulin5/p/8425175.html>

https://github.com/AplexOS/zh-cmn-Hans/blob/CMI\_AT151\_Linux-BSP/User's\_Guide.md

11.2 安装虚拟机USB驱动，让虚拟机可以检测到SD卡，聚体步骤如图所示

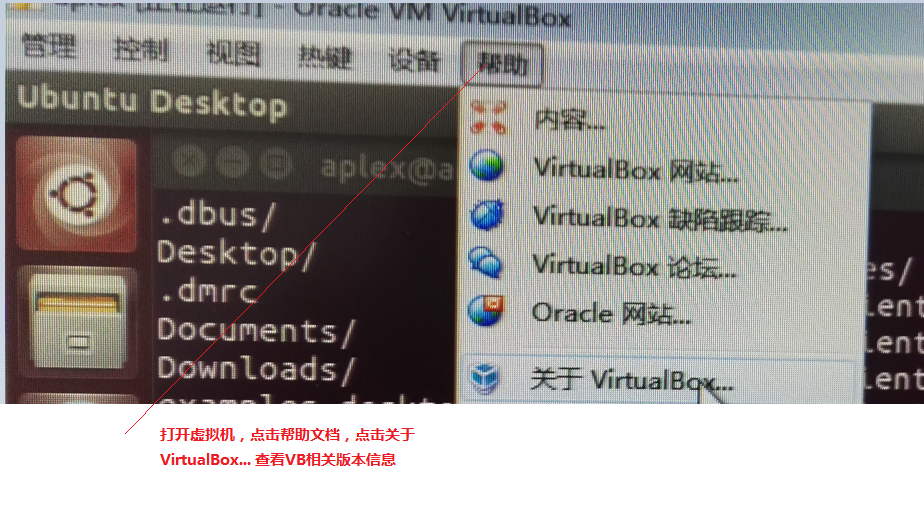




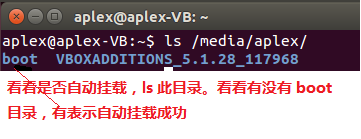
11.3 安装virtualbox 安装 USB 扩展功能，可以让虚拟机自动挂载sd卡，具体步骤如下图所示。（•扩展包下载地址：

<http://download.virtualbox.org/virtualbox/>

）







（此 图片中，/media/aplex 中的aplex是我的用户名）

11.4 直接在虚拟机上制作SD卡

将SD卡插入 Ubuntu 虚拟机中，并等待被检测生成设备节点。

首先要将挂载的设备节点解除挂载并格式化：

sudo umount /media/am335x/\*

# 这里的 am335x 是我用户名，当前环境是 ubuntu14.04

sudo dd if=/dev/zero of=/dev/sdb bs=1024 count=10

# 在执行 格式化 SD 卡的命令

# /dev/sdb 是SD卡产生的设备节点， 有可能也是 /dev/sdc /dev/sdd，但是一定不是 /dev/sda

对 SD 卡设置分区：

# 带 # 的都是注释

am335x@am335x:~$ sudo fdisk /dev/sdb # /dev/sdb 是SD卡设备节点

Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel

Building a new DOS disklabel with disk identifier 0x5c5103a7.

Changes will remain in memory only, until you decide to write them.

After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n # 添加新的分区

Partition type:

p primary (0 primary, 0 extended, 4 free)

e extended

Select (default p): p # 添加主分区

Partition number (1-4, default 1): # 回车确认

Using default value 1

First sector (2048-15126527, default 2048): # 回车确认

Using default value 2048

Last sector, +sectors or +size{K,M,G} (2048-15126527, default 15126527): +100M # 100 M 大小

Command (m for help): t # 改变分区格式

Selected partition 1

Hex code (type L to list codes): e # 改变为 FAT16 分区格式

Changed system type of partition 1 to e (W95 FAT16 (LBA))

Command (m for help): a # 添加 boot 属性

Partition number (1-4): 1 # 指定 第一个分区添加 boot 属性

Command (m for help): p # 打印当前分区信息

Disk /dev/sdb: 7744 MB, 7744782336 bytes

239 heads, 62 sectors/track, 1020 cylinders, total 15126528 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x5c5103a7

Device Boot Start End Blocks Id System

/dev/sdb1 \* 2048 206847 102400 e W95 FAT16 (LBA)

Command (m for help): w # 保存退出

The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x

partitions, please see the fdisk manual page for additional

information.

Syncing disks.

am335x@am335x:~$

对 SD 卡进行 FAT 16格式化

sudo mkfs.vfat -F 16 -n "boot" /dev/sdb1 # sdb1 表示你刚才配置的第一个分区

拔出 SD 卡并重新插入（为了让虚拟机自动挂载SD卡）， 将 ~/image 下面的所有文件拷贝到 SD 卡第一个分区内：

//可以自动挂载，无需此步骤，非自动挂载，可以运行此步骤sudo mkdir /media/am335x/sdb1 && sudo mount -t vfat /dev/sdb1 /media/am335x/sdb1

sudo cp ~/image/\* /media/am335x/sdb1 -rf //拷贝镜像文件

sync

拔出 SD 卡。

**十二：定制ubuntu**

使用工具：remastersys

12.1 参考文章简介

国外论坛参考<https://askubuntu.com/questions/133272/how-do-i-install-remastersys>

详细定制操作步骤<http://blog.51cto.com/solin/1865855>

详细定制之后安装的步骤<http://www.360doc.com/content/14/1128/09/14187254_428673726.shtml>

界面好看的定制步骤：<https://www.cnblogs.com/chenkun1/p/5532977.html>

12.2 安装必要的软件

转到此网站：http : //www.filewatcher.com并搜索，然后下载以下文件：

remastersys\_3.0.4-2\_all.deb //必须安装

remastersys-gui\_3.0.4-1\_i386.deb（32-bit）//32系统下载这个

remastersys-gui\_3.0.4-1\_amd64.deb（64-bit）//64 系统下载这个

也可以自动下载（Remastersys包下载）：

mkdir tools

cd tools/

wget ftp://ftp.gwdg.de/pub/linux/easyvdr/mirror/remastersys/ubuntu/remastersys/remastersys\_3.0.4-2\_all.deb

wget <ftp://ftp.gwdg.de/pub/linux/easyvdr/mirror/remastersys/ubuntu/remastersys-gui/remastersys-gui_3.0.4-1_amd64.deb>

安装依赖

（1）安装remastersys依赖

sudo apt-get install syslinux-utils isolinux squashfs-tools casper libdebian-installer4 ubiquity-frontend-debconf user-setup discover xresprobe systemd

（2）安装remastersys-gui依赖

sudo apt-get install libvte-common libvte9 plymouth-x11

开始安装Remastersys

cd tools/

sudo dpkg -i remastersys\_3.0.4-2\_all.deb

sudo dpkg -i remastersys-gui\_3.0.4-1\_amd64.deb

若是安装不成功（还有依赖没有安装），则执行这条命令

sudo apt-get -f install

重新安装Remastersys

sudo dpkg -i remastersys\_3.0.4-2\_all.deb

sudo dpkg -i remastersys-gui\_3.0.4-1\_amd64.deb

使用Remastersys备份操作系统

sudo remastersys-gui （根据需要进行备份）

**十三：SGX应用开发入门**

Sgx简介及开发指南参考链接：<https://software.intel.com/zh-cn/SGX-Development-Guide-Part-1>

链接的使用：将上述链接中的part-1改为part-2即可进入到第二页，一次类推进入第三页，第四页等等

Sgx个人理解，sgx是一门数据保护技术，是数据与操作系统之间的一层数据保护层，用户将需要保护的数据或函数（比如函数A）在sgx中定义并实现，然后在通过sgx对外的接口文件xxx.edl将函数A抛出，使外部App能够调用该函数。

由于sgx对象的创建需要安全密钥，而函数A又是sgx的内容，所以要想使用函数A，需要首先创建sgx对象，而要想创建可以使用函数A的sgx对象，则必须要有该sgx对象所需要的安全密钥（从而做到了对数据或函数的一层保护）。

**十四：AM335x+Linux移植SGX**

缩写全称：

DVSDK ：Digital Video Software Development Kit

PSP ：Linux Platform Support Package

安装XXX.bin文件运行必要的库：

**sudo apt-get install libc6:i386 libstdc++6:i386 libncurses5:i386 zlib1g:i386**

为AM335x+Linux移植SGX+OpenGL+Qt5之完全开发笔记：<http://blog.csdn.net/jzjhome/article/details/54923765>

AM335X开发日记:<https://wenku.baidu.com/view/d0b7ab07172ded630a1cb659.html>

SGXDbg（错误调试）：<http://processors.wiki.ti.com/index.php/SGXDbg>

通过 SGX 编译 Qt支持OpenGL ES ：<http://processors.wiki.ti.com/index.php/Building_Qt_with_OpenGL_ES_accelerated_by_SGX>

配置内核是内核支持对应的ti的SGX开发工具包（富林）：<http://www.cnblogs.com/chenfulin5/p/7245590.html>

Ti 基本工具链接

Ti 交叉编译工具链简介：<http://processors.wiki.ti.com/index.php/Processor_Linux_SDK_GCC_Toolchain>

Ti 的SGX简介：<https://baike.baidu.com/item/PowerVR%20SGX/4331963>

u\_boot源码库（点击Archived）：<http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_U-Boot_Release_Notes>

u\_boot 构建指南：<http://processors.wiki.ti.com/index.php/Linux_Core_U-Boot_User%27s_Guide>

内核（PSP）使用指南：<http://processors.wiki.ti.com/index.php/Linux_Kernel_Users_Guide>

内核源码库：[http://processors.wiki.ti.com/index.php/Processor\_SDK\_Linux\_Kernel\_Release\_Notes#Archived](http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Kernel_Release_Notes" \l "Archived)

处理器开发工具包库（process\_sdk）：<http://processors.wiki.ti.com/index.php/Processor_SDK_Building_The_SDK>

处理器开发工具包支持功能介绍（ti组件大全）：<http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Software_Developer%E2%80%99s_Guide>

数字视频处理器开发工具包（库）：<http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/dvsdk/>

数字视频处理器开发工具包使用指南：<http://processors.wiki.ti.com/index.php/AM35x-OMAP35x_Graphics_SDK_Getting_Started_Guide>

3d图形开发工具包库（Graphics SDK）：<http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/gfxsdk/>

3d图形开发工具包库使用说明：<http://processors.wiki.ti.com/index.php/Graphics_SDK_Quick_installation_and_user_guide>

编译QT：<http://processors.wiki.ti.com/index.php/Building_Qt>

Ti sgx 编译与安装

1：安装必要的组件（以内核模块的形式加载进内核xxx.ko）

将omaplfb, pvrsrvkm, bufferclass\_ti 三个模块加载进内核

2：通过上述组件提供合适的GLES11/2.0/VG 的头文件，用户可以根据此头文件和上述组件编译自己的3D软件。

http://processors.wiki.ti.com/index.php/Graphics\_SDK\_Quick\_installation\_and\_user\_guide

3：具体操作步骤

1：准备交叉编译工具链

2：编译U-boot 参考uboot使用指南

3：编译内核（PSP）参考内核使用指南

4: 根据编译的内核编译SGX

编译SGX（根据自己的需要编译属于自己的sgx内核模块）

1：编译sgx

1：准备工作

交叉编译工具链

1. boot kernel

2：开始编译sgx演示案列

cd Graphics\_SDK\_#\_##\_##\_##

export ARCH=arm

3: vi Graphics\_SDK\_#\_##\_##\_##/Rules.make 在该文件中设置几个变量：

GRAPHICS\_INSTALL\_DIR变量，使其指向SGX的安装目录，例如：

GRAPHICS\_INSTALL\_DIR=~/Graphics\_SDK\_#\_##\_##\_##

CSTOOL\_DIR ：交叉编译工具链的安装目录（不包含bin目录），例如：

CSTOOL\_DIR = /home/<user\_account>/toolchain

CSTOOL\_PREFIX:交叉编译工具链的前缀，例如

CSTOOL\_PREFIX = arm-none-linux-gnueabi-

KERNEL\_INSTALL\_DIR：内核源文件的根目录。例如

KERNEL\_INSTALL\_DIR = ~/linux-##.##.##.##

TARGETFS\_INSTALL\_DIR：根文件系统的根目录。例如

TARGETFS\_INSTALL\_DIR=~/aplex/rootfs

打补丁方法适用于没有设备树的内核版本：

4.1：给内核打补丁（当内核版本大于3.14时，假如我的内核版本是3.2.0，支持PVRSRVDriver功能）

1：补丁链接：<http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/gfxsdk/4_09_00_01/index_FDS.html> （点击下载[AM335x kernel patch for SGX device registration using HWMOD APIs](http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/gfxsdk/4_09_00_01/exports/0001_SGX_AM335x_SGX_graphics_device_registration_using_HW.patch)）

2：给内核打补丁 （补丁文件针对于AM335X，其他的无需打补丁）

将下载的补丁文件拷贝到内核的顶层目录下，并打补丁

cp 0001\_SGX\_AM335x\_SGX\_graphics\_device\_registration\_using\_HW.patch kernel3.2.0

cd kernel3.2.0

patch -p1 < 0001\_SGX\_AM335x\_SGX\_graphics\_device\_registration\_using\_HW.patch

patch -R -p1 < 0001\_SGX\_AM335x\_SGX\_graphics\_device\_registration\_using\_HW.patch （卸载补丁文件）

3：如果打补丁执行失败，则可以根据补丁文件内容手动将补丁文件的内容添加到内核源码中对应的文件上。

修改arch/arm/mach-omap2/board-am335xevm.c 文件，在static void clkout2\_enable(int evm\_id, int profile)函数后面添加如下内容：

static void sgx\_init(int evm\_id, int profile)

{

if (omap3\_has\_sgx()) {

am33xx\_gpu\_init();

}

}；

在static struct evm\_dev\_cfg gen\_purp\_evm\_dev\_cfg[]结构体中（末尾）添加如下内容：

{sgx\_init, DEV\_ON\_BASEBOARD, PROFILE\_ALL}, ；

根据补丁文件中的结构体在内核源码中查找相应的结构体，并做对应的修改，若内核里面没有对应的结构体，则无需修改。

修改arch/arm/mach-omap2/devices.c文件，在int \_\_init omap\_init\_gpmc(struct gpmc\_devices\_info \*p data, int pdata\_len)函数后面添加如下函数：

void \_\_init am33xx\_gpu\_init(void)

{

int id = -1;

struct platform\_device \*pdev;

struct omap\_hwmod \*oh;

char \*oh\_name = "gfx";

char \*dev\_name = "pvrsrvkm";

oh = omap\_hwmod\_lookup(oh\_name);

if (!oh) {

pr\_err("Could not find %s hwmod data\n", oh\_name);

return;

}

pdev = omap\_device\_build(dev\_name, id, oh, NULL, 0, NULL, 0, 0);

WARN(IS\_ERR(pdev), "could not build omap\_device for %s\n", oh\_name);

}

修改arch/arm/mach-omap2/devices.h文件，在extern int \_\_init omap\_init\_elm(void)后面添加如下内容：

extern void \_\_init am33xx\_gpu\_init(void);

4.：内核修改完成后，接着修改Graphics\_SDK\_5\_01\_01\_02/GFX\_Linux\_KM/services4/srvkm/env/linux/module.c文件，在 下面三句话前面添加一条宏定义：#define CONFIG\_RESET\_CONTROLLER

#ifndef CONFIG\_RESET\_CONTROLLER

#include <linux/platform\_data/sgx-omap.h>

#endif



4.2 手动修改内核（带有设备树的版本，支持PVRSRVDriver功能）

参考文件：<http://www.eefocus.com/marianna/blog/15-04/311437_e4f0a.html>

1. 修改memuconfig中有关reset的内容：

使用make menuconfig命令，使能RESET\_CONTROLLER：

CONFIG\_RESET\_CONTROLLER=y

这一步是必须的，因为PVR服务的pvrsrvkm模块用到了很多reset\_control\_\*函数。

2 在drivers/reset/core.c文件末尾添加如下内容：

int reset\_control\_is\_reset(struct reset\_control \*rstc)

{

if (rstc->rcdev->ops->is\_reset)

return rstc->rcdev->ops->is\_reset(rstc->rcdev, rstc->id);

return -ENOSYS;

}

EXPORT\_SYMBOL\_GPL(reset\_control\_is\_reset);

int reset\_control\_clear\_reset(struct reset\_control \*rstc)

{

if (rstc->rcdev->ops->clear\_reset)

return rstc->rcdev->ops->clear\_reset(rstc->rcdev, rstc->id);

return -ENOSYS;

}

EXPORT\_SYMBOL\_GPL(reset\_control\_clear\_reset);

3 修改include/linux/reset-controller.h文件，修改struct reset\_control\_ops的定义，在里面将上述两个函数类型添加进去；

struct reset\_control\_ops {

int (\*reset)(struct reset\_controller\_dev \*rcdev, unsigned long id);

int (\*assert)(struct reset\_controller\_dev \*rcdev, unsigned long id);

int (\*deassert)(struct reset\_controller\_dev \*rcdev, unsigned long id);

int (\*is\_reset)(struct reset\_controller\_dev \*rcdev, unsigned long id);

int (\*clear\_reset)(struct reset\_controller\_dev \*rcdev, unsigned long id);

};

4 修改include/linux/reset.h文件，将上述步骤2中两个函数的声明添加进去：

int reset\_control\_is\_reset(struct reset\_control \*rstc);

int reset\_control\_clear\_reset(struct reset\_control \*rstc);

5 在arch/arm/boot/dts/am33xx.dtsi文件添加如下内容（若已经存在，则无需重复添加）：

prcm: prcm@44e00000 {

compatible = "ti,am3-prcm";

reg = <0x44e00000 0x4000>;

#reset-cells = <1>;

prcm\_clocks: clocks {

#address-cells = <1>;

#size-cells = <0>;

};

prcm\_clockdomains: clockdomains {

};

};

sgx@0x56000000 {

compatible = "ti,sgx";

ti,hwmods = "gfx";

reg = <0x56000000 0x1000000>;

interrupts = <37>;

resets = <&prcm 0>;

};

6 为drivers/video/fbdev/da8xx-fb.c文件添加如下内容：

static vsync\_callback\_t vsync\_cb\_handler;

static void \*vsync\_cb\_arg;

int register\_vsync\_cb(vsync\_callback\_t handler, void \*arg, int idx)

{

if ((vsync\_cb\_handler == NULL) && (vsync\_cb\_arg == NULL)) {

vsync\_cb\_arg = arg;

vsync\_cb\_handler = handler;

} else {

return -EEXIST;

}

return 0;

}

EXPORT\_SYMBOL(register\_vsync\_cb);

int unregister\_vsync\_cb(vsync\_callback\_t handler, void \*arg, int idx)

{

if ((vsync\_cb\_handler == handler) && (vsync\_cb\_arg == arg)) {

vsync\_cb\_handler = NULL;

vsync\_cb\_arg = NULL;

} else {

return -ENXIO;

}

return 0;

}

EXPORT\_SYMBOL(unregister\_vsync\_cb);

7 在include/video/da8xx-fb.h文件添加如下内容：

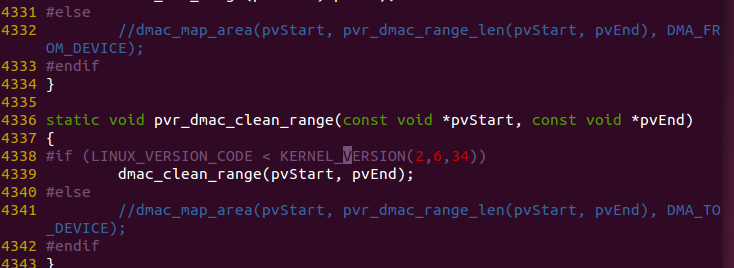
typedef void (\*vsync\_callback\_t)(void \*arg);

int register\_vsync\_cb(vsync\_callback\_t handler, void \*arg, int idx);

int unregister\_vsync\_cb(vsync\_callback\_t handler, void \*arg, int idx);

8 修改Graphics\_SDK\_5\_01\_01\_02/GFX\_Linux\_KM/services4/srvkm/env/linux/osfunc.c文件。

因为我们的内核版本大于2.65，所以用不到dmac\_map\_area这个函数（否则），因此将其注释掉即可（否则会报错）。



4.3 用TI提供的PSP开发包进行编译（不支持PVRSRVDriver功能）

参考链接：<http://www.cnblogs.com/chenfulin5/p/7245590.html>

5: 使用make命令进行编译，规则如下：

make BUILD={debug | release} OMAPES={3.x | 5.x | 6.x | 8.x} FBDEV={yes | no} SUPPORT\_XORG= {1 | 0 } all|all\_km

BUILD：编译版本

OMAPES：对应的平台

FBDEV：显示设备

SUPPORT\_XORG：1表示支持Xorg

all|all\_km：编译选项，all全部（包括实例）编译，all\_km只编译SGX的内核模块。

编译使用举列：

make BUILD=release OMAPES=8.x all\_km（不包含xorg编译）

make BUILD=release OMAPES=6.x SUPPORT\_XORG=1 all (支持xorg 编译)

make BUILD=release OMAPES=6.x SUPPORT\_XORG=1 install\_km （安装）

make BUILD=release OMAPES=6.x SUPPORT\_XORG=1 install （安装）

5 启动sgx，运行特定的启动脚本

Sgx SDK开发包SDK附带一个的启动脚本（跟soc相绑定），负责安装正确的sgx驱动程序库和内核模块。

OMAP35x：/etc/init.d/omap-demo

AM335x：/etc/init.d/335x-demo

驱动程序安装成功的标志是：在控制台上看到这条消息

Loaded PowerVR consumer services

1. 编译sgx版本的QT

1 从http://processors.wiki.ti.com/index.php/Building\_Qt\_with\_OpenGL\_ES\_accelerated\_by\_SGX#Building\_Qt\_with\_OpenGL\_ES 文档中下载QT的配置文件：点击Download this tarball 下载配置文件。

2 qmake.cong配置文件中一些变量讲解

QT\_INSTALL\_DIR ：qt的源码顶层目录

QT\_INSTALL\_DIR = /home/aplex/qt/qt-everywhere-opensource-src-5.6.2

SGX\_SDK\_ROOT ：sgx的Sdk的根目录

SGX\_SDK\_ROOT = /home/aplex/Graphics\_SDK\_5\_01\_01\_02

TSLIB\_INCDIR：tslib头文件目录

TSLIB\_LIBDIR ：tslib的库文件目录

**十五：openVPN使用**

软件链接地址（可直接使用）：<https://pan.baidu.com/s/1dna4d4> 密码jn6f

参考地址：<http://blog.csdn.net/joyous/article/details/78756865>

**十六：设备树的使用**

参考链接：<http://blog.csdn.net/dzw19911024/article/details/51741405>

1. **设备树的编译与重新编译**

首先配置内核 ，列如：make am335x\_cmi\_at151\_defconfig

(重新)编译设备树：make dtbs make dbts clean && make dbts

1. **设备树在内核的表示方法：**

操作设置树的函数一般都是以of打头。

设备树的注释文档：Documentation/devicetree/bindings

设备树节点的表示方法：struct device\_node 代表了设备树节点

struct device\_node \*of\_find\_node\_by\_path(const char \*path)

功能：通过路径找到设备树节点

参数：path 节点路径（列如 /cpus）

返回值：设备树节点指针

struct property \*of\_find\_property(const struct device\_node \*np,

const char \*name, int \*lenp)

功能：已知设备树节点的指针，获取设备节点的属性键值对的值

参数：np 节点指针

name 键的名称

lenp 值的长度

返回值：得到 属性信息结构体指针

struct property 设备节点属性信息结构体

属性键值对结构体

struct property {

char \*name; // 名字

int length;

void \*value; // 值

struct property \*next;

unsigned long \_flags;

unsigned int unique\_id;

};

int of\_property\_read\_string(struct device\_node \*np,

const char \*propname,

const char \*\*out\_string);

功能：将节点属性键值对的值当做字符串转化出来，是对上一个函数进行封装

参数：np 节点指针

proname 键的名称

out\_string 值的存储地址

1. **设备树的解读与配置**

下面以一个最简单的machine为例来看如何写一个.dts文件。假设此machine的配置如下：

1个双核ARM Cortex-A9 32位处理器；

ARM的local bus上的内存映射区域分布了2个串口（分别位于0x101F1000 和 0x101F2000）、GPIO控制器（位于0x101F3000）、SPI控制器（位于0x10170000）、中断控制器（位于0x10140000）和一个external bus桥；

External bus桥上又连接了SMC SMC91111 Ethernet（位于0x10100000）、I2C控制器（位于0x10160000）、64MB NOR Flash（位于0x30000000）；

External bus桥上连接的I2C控制器所对应的I2C总线上又连接了Maxim DS1338实时钟（I2C地址为0x58）。

其对应的.dts文件为：

/ {

compatible = "acme,coyotes-revenge";

#address-cells = <1>;

#size-cells = <1>;

interrupt-parent = <&intc>;

cpus {

#address-cells = <1>;

#size-cells = <0>;

cpu@0 {

compatible = "arm,cortex-a9";

reg = <0>;

};

cpu@1 {

compatible = "arm,cortex-a9";

reg = <1>;

};

};

serial@101f0000 {

compatible = "arm,pl011";

reg = <0x101f0000 0x1000 >;

interrupts = < 1 0 >;

};

serial@101f2000 {

compatible = "arm,pl011";

reg = <0x101f2000 0x1000 >;

interrupts = < 2 0 >;

};

gpio@101f3000 {

compatible = "arm,pl061";

reg = <0x101f3000 0x1000

0x101f4000 0x0010>;

interrupts = < 3 0 >;

};

intc: interrupt-controller@10140000 {

compatible = "arm,pl190";

reg = <0x10140000 0x1000 >;

interrupt-controller;

#interrupt-cells = <2>;

}; intc 是一个节点的标号，类似于汇编语句的标号

spi@10115000 {

compatible = "arm,pl022";

reg = <0x10115000 0x1000 >;

interrupts = < 4 0 >; 此处interrupts的含义需要看中断控制器在内核的解读方式，请查看Documentation/devicetree/bindings的相关文档进行解读

};

external-bus {

#address-cells = <2>

#size-cells = <1>;

ranges = <0 0 0x10100000 0x10000 // Chipselect 1, Ethernet

1 0 0x10160000 0x10000 // Chipselect 2, i2c controller

2 0 0x30000000 0x1000000>; // Chipselect 3, NOR Flash

其中reg的组织形式为reg = <address1 length1 [address2 length2] [address3 length3] ... >，其中的每一组address length表明了设备使用的一个地址范围。address为1个或多个32位的整型（即cell），而length则为cell的列表或者为空（若#size-cells = 0）。address 和 length 字段是可变长的，父结点的#address-cells和#size-cells分别决定了子结点的reg属性的address和length字段的长度。在本例中，root结点的#address-cells = <1>;和#size-cells = <1>;决定了serial、gpio、spi等结点的address和length字段的长度分别为1。cpus 结点的#address-cells = <1>;和#size-cells = <0>;决定了2个cpu子结点的address为1，而length为空，于是形成了2个cpu的reg = <0>;和reg = <1>;。external-bus结点的#address-cells = <2>和#size-cells = <1>;决定了其下的ethernet、i2c、flash的reg字段形如reg = <0 0 0x1000>;、reg = <1 0 0x1000>;和reg = <2 0 0x4000000>;。其中，address字段长度为0，开始的第一个cell（0、1、2）是对应的片选，第2个cell（0，0，0）是相对该片选的基地址，第3个cell（0x1000、0x1000、0x4000000）为length。特别要留意的是i2c结点中定义的 #address-cells = <1>;和#size-cells = <0>;又作用到了I2C总线上连接的RTC，它的address字段为0x58，是设备的I2C地址。

root结点的子结点描述的是CPU的视图，因此root子结点的address区域就直接位于CPU的memory区域。但是，经过总线桥后的address往往需要经过转换才能对应的CPU的memory映射。external-bus的ranges属性定义了经过external-bus桥后的地址范围如何映射到CPU的memory区域。

ranges是地址转换表，其中的每个项目是一个子地址、父地址以及在子地址空间的大小的映射。映射表中的子地址、父地址分别采用子地址空间的#address-cells和父地址空间的#address-cells大小。对于本例而言，子地址空间的#address-cells为2，父地址空间的#address-cells值为1，因此0 0 0x10100000 0x10000的前2个cell为external-bus后片选0上偏移0，第3个cell表示external-bus后片选0上偏移0的地址空间被映射到CPU的0x10100000位置，第4个cell表示映射的大小为0x10000。

ethernet@0,0 {

compatible = "smc,smc91c111";

reg = <0 0 0x1000>;

interrupts = < 5 2 >;

};

i2c@1,0 {

compatible = "acme,a1234-i2c-bus";

#address-cells = <1>;

#size-cells = <0>;

reg = <1 0 0x1000>;

interrupts = < 6 2 >;

rtc@58 {

compatible = "maxim,ds1338";

reg = <58>;

interrupts = < 7 3 >;

};

};

flash@2,0 {

compatible = "samsung,k8f1315ebm", "cfi-flash";

reg = <2 0 0x4000000>;

};

};

};

**十七：gpio的配置**

1. Ti的AM33XX系列芯片，gpio共有132个pin，分为四个gpio模块，分别为gpio0，g

pio,1，gpio2，gpio3，所以gpio1\_0的引脚对应的gpio编号是gpio33。

1. Gpio配置
2. 进入 /sys/class/gpio/ 目录
3. 假设你想控制的GPIO0\_29，步骤如下：
4. echo 29 > export
5. 此时会产生一个gpio29 的目录.
6. cd gpio29
7. 关注以下几个文件：

direction 这个文件是控制你是输出还是输入模式 如果想设置为输入：echo in > direction

如果想设置为输出：echo out > direction

value 这个文件是在输出模式时，控制高低电平 高电平：echo 1 > value 低电平：echo 0 > value

edge 这个控制中断触发模式 无： echo none > edge 上升沿触发：echo rising > edge 下降沿触发：echo falling > edge 轮询：echo both > edge active\_low: 这个按照官方的说法是这个翻转电平，输入非零的数即上升沿变为下降沿。 o: echo 0 > active\_low 非零的数： echo 3 > active\_low 5.

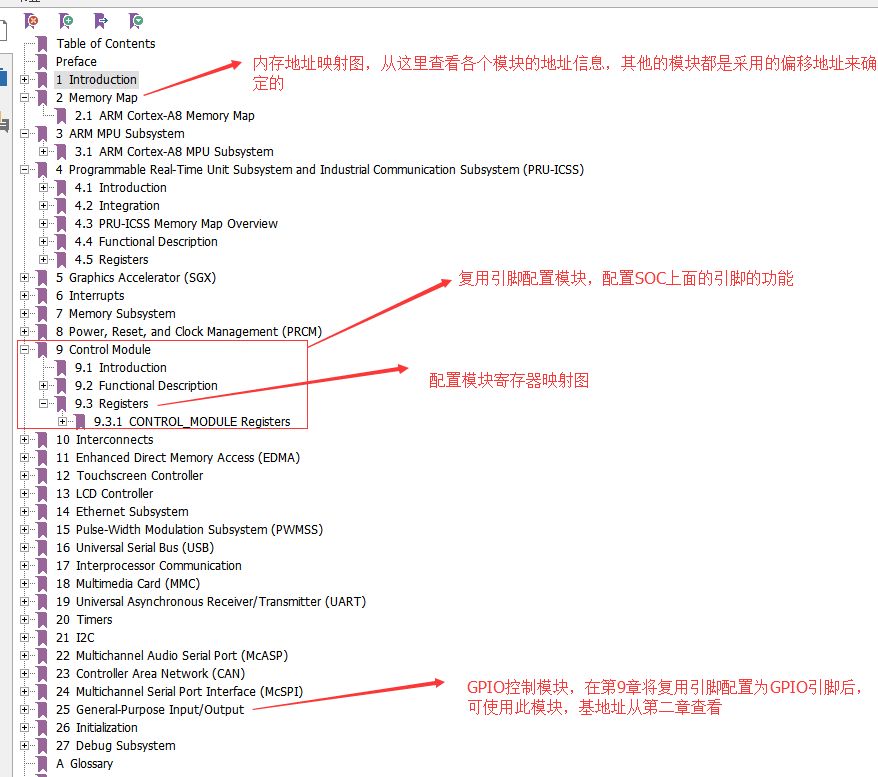
关闭这个GPIO控制。 cd /sys/class/gpio/ echo 29 > unexport

1. 参考：https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

<http://blog.csdn.net/mirkerson/article/details/8464290>

**十八：TI芯片手册使用简介**

1. Ti的芯片手册使用不同于三星的芯片手册，具体详看下图



**十九：modprobe使用介绍**

1. 假设我有三个内核模块如下：

mmc\_block.ko mmc\_core.ko sep\_mci.ko

1. 在开发板上创建如下目录：

mkdir /lib/modules

cd /lib/modules

uname -r （此命令会出现一条结果。列如：4.4.12-g2a384b7-dirty）

mkdir 4.4.12-g2a384b7-dirty

将上述三个模块拷贝到4.4.12-g2a384b7-dirty目录文件中

cp mmc\_block.ko /lib/modules/4.4.12-g2a384b7-dirty

cp mmc\_core.ko /lib/modules/4.4.12-g2a384b7-dirty

cp sep\_mci.ko /lib/modules/4.4.12-g2a384b7-dirty

1. 使用depmod 命令将模块之间生成依赖关系（modules.dep）

Depmod （-a）

1. 使用modprobe命令加载内核模块

modprobe sep\_mci

参考资料：<https://zhidao.baidu.com/question/1823330792079126708.html>

**二十：开发板模块测试**

串口测试命令：microcom -s 115200 /dev/ttyO1

串口类型接口：

RS232 接线方式 三线链接（主机r接从机T，地线接地线）

RS485 接线方式 半双工（两线 主机的A（高电压）线接从机的A，顺序要接对）

Uart 接线方式，同RS232

**二十一：websocket使用简介**

**websocket 初步使用经验（python）**

**参考链接：<http://www.cnblogs.com/chenfulin5/p/7794950.html>**

**•想实现网页前端和后端的数据同步交互，就有必要使用 websocket 的方式进行通信。**

**•python websocket github 地址：git@github.com:Aplexchenfl/python-websocket-server.git**

**•下载之后：**

**ECM\_5412@chenfl:/var/www/html$ ls**

**client.html README.md server.py setup.py tests websocket\_server•其中，服务端的代码如下：**

**vim server.pyfrom websocket\_server import WebsocketServer**

**# Called for every client connecting (after handshake)**

**// 定义一个函数，当有新的客户端连接时会调用此函数**

**def new\_client(client, server):**

**print("New client connected and was given id %d" % client['id'])**

**server.send\_message\_to\_all("Hey all, a new client has joined us")**

**//定义一个函数，当客户端断开连接是会调用此函数**

**# Called for every client disconnecting**

**def client\_left(client, server):**

**print("Client(%d) disconnected" % client['id'])**

**// 定义一个函数，接收到客户端的信息会回调此函数。**

**# Called when a client sends a message**

**def message\_received(client, server, message):**

**if len(message) > 200:**

**message = message[:200]+'..'**

**print("Client(%d) said: %s" % (client['id'], message))**

**PORT=9001 //定义服务端端口**

**server = WebsocketServer(PORT, "0.0.0.0") //建立websocket服务器对象**

**server.set\_fn\_new\_client(new\_client) //注册客户端链接的回到函数**

**server.set\_fn\_client\_left(client\_left) //注册客户端断开连接时的回调函数**

**server.set\_fn\_message\_received(message\_received) //注册消息处理函数**

**server.run\_forever()// 启动websocket服务器对象**

**python3 server.py （通过python3运行此文件）**

**•客户端代码如下（用JS所写）：**

**<html>**

**<head>**

**<title>Simple client</title>**

**<script type="text/javascript">**

**var ws;**

**function init() {**

**// 获取服务端ip**

**var ip\_addr = document.location.hostname;**

**window.WebSocket = window.WebSocket || window.MozWebSocket;**

**ws = new WebSocket('ws://' + ip\_addr +':9001'); // 申请新的客户端**

**// Connect to Web Socket**

**//ws = new WebSocket("ws://localhost:9001/");**

**// Set event handlers.**

**ws.onopen = function() {**

**output("onopen");**

**};**

**ws.onmessage = function(e) {**

**// e.data contains received string.**

**output("onmessage: " + e.data);**

**};**

**ws.onclose = function() {**

**output("onclose");**

**};**

**ws.onerror = function(e) {**

**output("onerror");**

**console.log(e)**

**};**

**}**

**function onSubmit() {**

**var input = document.getElementById("input");**

**// You can send message to the Web Socket using ws.send.**

**ws.send(input.value);**

**output("send: " + input.value);**

**input.value = "";**

**input.focus();**

**}**

**function onCloseClick() {**

**ws.close();**

**}**

**function output(str) {**

**var log = document.getElementById("log");**

**var escaped = str.replace(/&/, "&amp;").replace(/</, "&lt;").**

**replace(/>/, "&gt;").replace(/"/, "&quot;"); // "**

**log.innerHTML = escaped + "<br>" + log.innerHTML;**

**}**

**</script>**

**</head>**

**<body onload="init();">**

**<form onsubmit="onSubmit(); return false;">**

**<input type="text" id="input">**

**<input type="submit" value="Send">**

**<button onclick="onCloseClick(); return false;">close</button>**

**</form>**

**<div id="log"></div>**

**</body>**

**</html> // 通过网页启动客户端。**

**Python 系统调用参考：<https://www.cnblogs.com/yangykaifa/p/7127776.html>**

**Python websocket实现简介：<http://blog.csdn.net/toulen3171/article/details/48345027>**