移植linux系统到ZCU102

0.环境

装有Linux系统的上位机（或虚拟机）一台，本实验使用Linux版本为Ubuntu 16.04.6 LTS；Xilinx Zynq系列ZCU102开发板一块。

上位机安装vivado，本实验所用版本为2018.3。

移植的Linux版本见下文“源码下载”。

0.1格式化SD卡并分区

本实验最终的linux系统以开发板SD卡启动作为启动方式，需要配置SD卡并建立分区。见UG1144第七章。使用gparted将SD卡格式化，并新建分区。

安装gparted工具：

sudo apt-get install gparted

运行：

sudo gparted

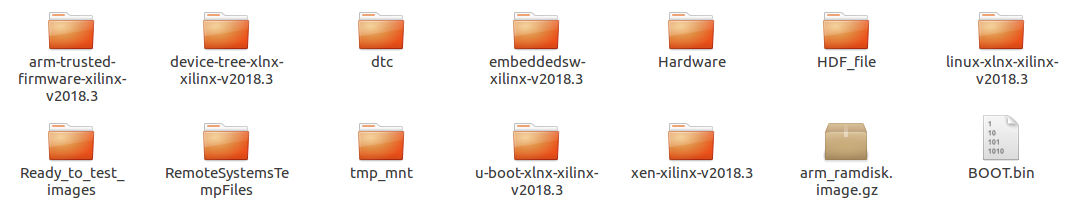
之后在gparted对SD卡进行分区。分2个主分区，第一个分区取名（设置卷标）为BOOT，文件系统选择FAT32，容量为1GB（大于60MB）；第二个分区取名为RootFS，占有剩余的全部容量，文件系统选择EXT4。

1.制作BOOT相关文件

1.0源码下载

从github上下载由xilinx提供的linux内核、u-boot源码以及其他相关的工程插件，下载时一定要注意统一源码以及工程的版本。相应的下载地址如下：

|  |  |
| --- | --- |
| [https://github.com/Xilinx/linux-xlnx](https://github.com/Xilinx/linux-xlnx.git) | 内核源码 |
| [https://github.com/Xilinx/u-boot-xlnx](https://github.com/Xilinx/u-boot-xlnx.git) | u-boot源码 |
| [https://github.com/Xilinx/device-tree-xlnx](https://github.com/Xilinx/device-tree-xlnx.git) | DTG插件 |
| [https://git.kernel.org/pub/scm/utils/dtc/dtc](https://git.kernel.org/pub/scm/utils/dtc/dtc.git) | DTC |
| [https://github.com/Xilinx/arm-trusted-firmware](https://github.com/Xilinx/arm-trusted-firmware.git) | ATF |
| <https://github.com/Xilinx/xen> | Xilinx Xen branch |
| <https://github.com/Xilinx/embeddedsw> | Xilinx embeddedsw repository |

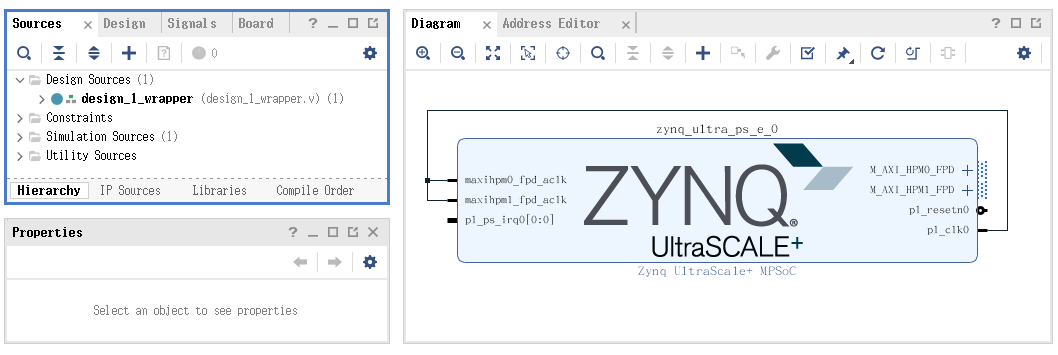


特别的，vivado工程生成的设备树源码文件有问题，可以用patelinux建立，此处直接给出，即lib中zcu102-rev1.0.dts。

1.1制作BOOT.BIN文件

1.1.1创建vivado工程，导出hdf文件，生成bit流

打开vivado，新建工程，板卡选择zcu102，打开Blockdesign，对MPSOC进行automatic，连线clock如下图。



保存、编译，然后使用vivado的generate output products功能，生成相关 IP 的网表、设计、仿真文件。选择 create HDL Wrapper，自动生成 HDL顶层代码（vivado的主要优势，无需关心HDL级设计，通过高层综合生成HDL级IP核）。

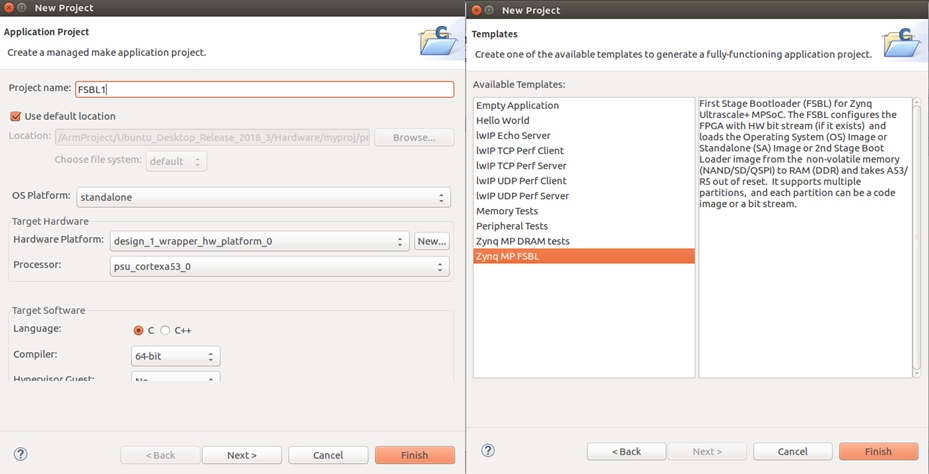
生成bitstream文件design\_1\_wrapper.bit，选择 Export Hardware导出硬件配置文件，硬件设计部分到此完成，然后Launch SDK。

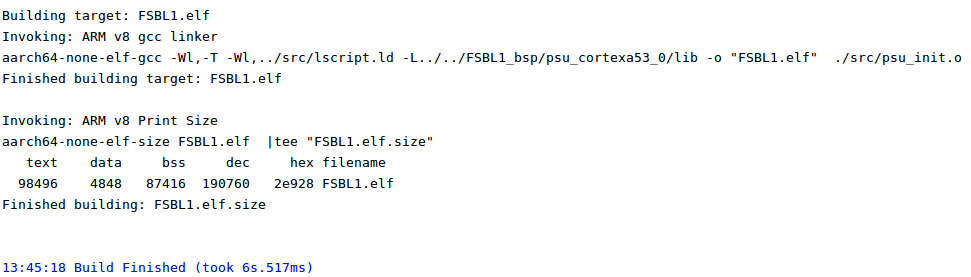
1.1.2制作FSBL

FSBL的全称为first stage boot loader，是zynq启动第一阶段的加载程序，经过了FSBL这一阶段，后面系统才能够运行裸机程序或者是引导操作系统的u-boot。可以使用SDK软件来创建FSBL:

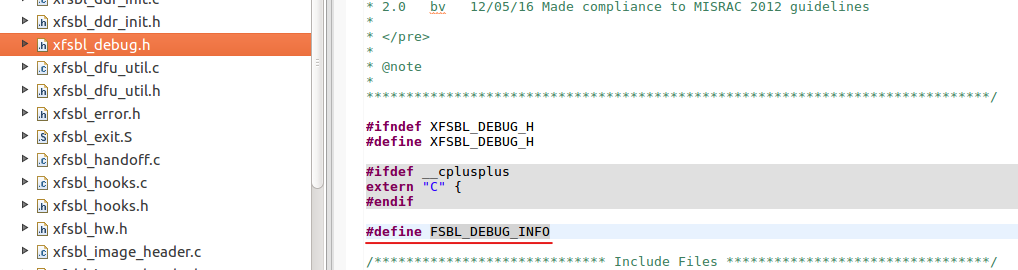
硬件设计已经整合在了design\_1\_wrapper\_hw\_platform\_0中。

1）打开SDK软件，New->Application Project->Next->finish





2）在FSBL->src->xfsbl\_debug.h中添加 #define FSBL\_DEBUG\_INFO ->保存。这样做的目的是为了在上电后输出FSBL的调试信息，以便出现问题时定位问题所在。

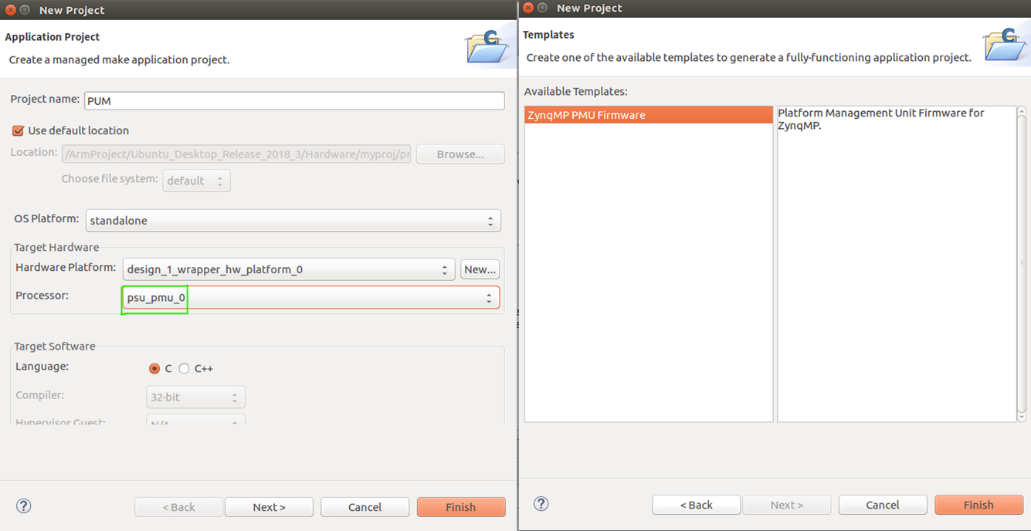


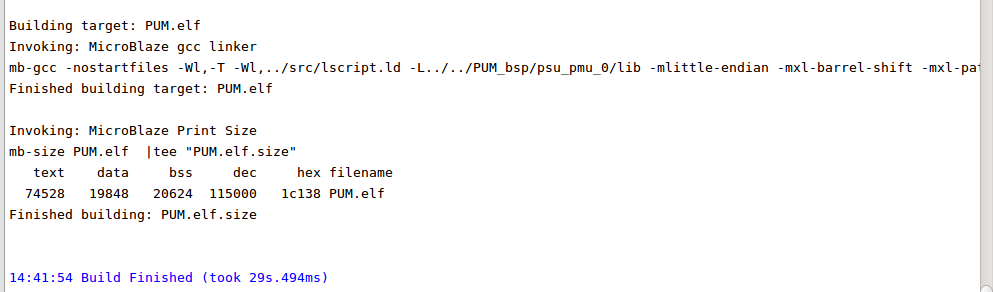
3）保存后，自动编译，在工程文件夹.sdk目录下FSBL目录（名称为自定义，可能不相同）下Debug内生成FSBL.elf文件。

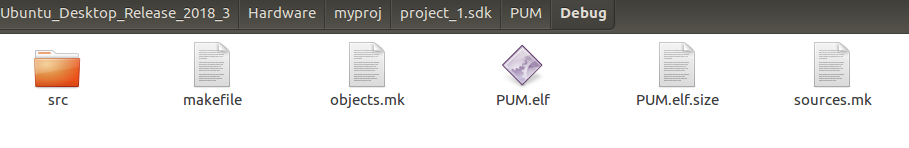
1.1.3制作PMU固件

PMU: platform management unit, 平台管理单元负责在加载FSBL之前进行调试，系统和软件的复位。PMU固件可以通过SDK软件创建。

New->Application Project->Processor选择psu\_pmu\_0，点击Next，选择Zynq PMU Firmware->finish，最后在工程目录下.sdk目录下/PUM/Debug/下生成可执行文件PUM.elf。







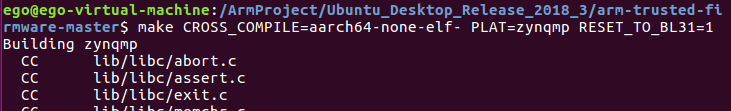
1.1.4制作ATF(ARM Trusted Firmware)

ATF的主要功能是在安全环境与非安全环境切换。创建ATF是创建BOOT.bin文件必不可少的一环。

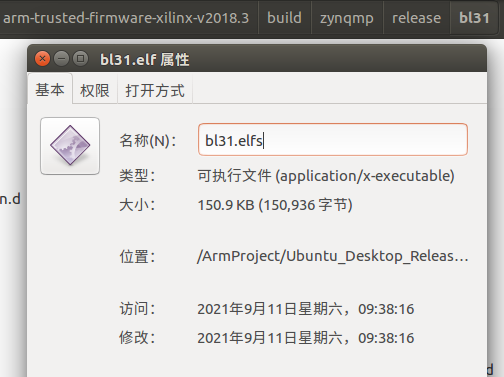
从<https://github.com/Xilinx/arm-trusted-firmware> 中下载arm-trusted-firmware-xilinx-v2018.3.zip，解压后进入文件夹（本实验为arm-trusted-firmware-master）。

执行make命令：

make CROSS\_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET\_TO\_BL31=1



编译完成后可以在/arm-trusted-firmware-master/build/zynqmp/release/bl31/中发现可执行文件bl31.elf。



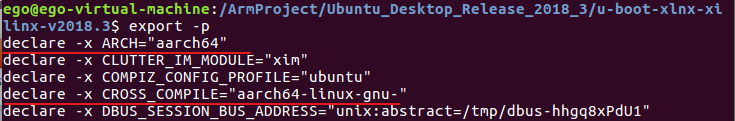
1.1.5制作u-boot

从<https://github.com/Xilinx/u-boot-xlnx>下载相应版本的u-boot源码，并解压到合适路径下，下文以./u-boot-xlnx-xilinx-v2018.3代指。

1）添加正确的路径变量，此处作为临时环境变量添加，对于ZCU102：

export CROSS\_COMPILE=aarch64-linux-gnu-

export ARCH=aarch64



2）编译u-boot源码(编译方法与源码版本有关，这里使用的是2018.3版本)：

make distclean

make xilinx\_zynqmp\_zcu102\_rev1\_0\_defconfig

make

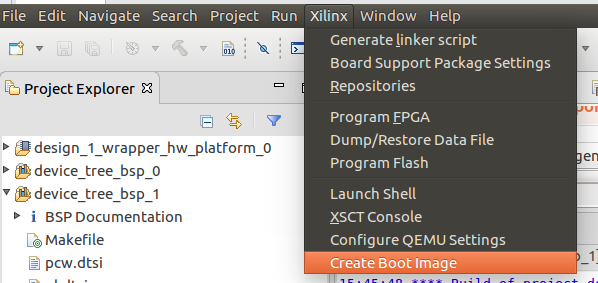
编译完成后可以在./u-boot-xlnx-xilinx-v2018.3/下发现可执行文件u-boot.elf。



1.1.6制作BOOT.BIN文件

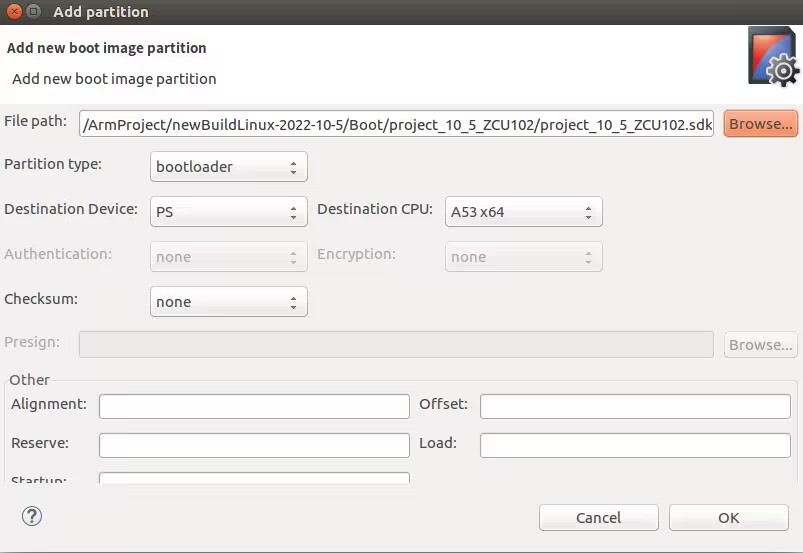
可以使用SDK软件来制作BOOT.bin文件

1）SDK->Xilinx->Create Boot Image

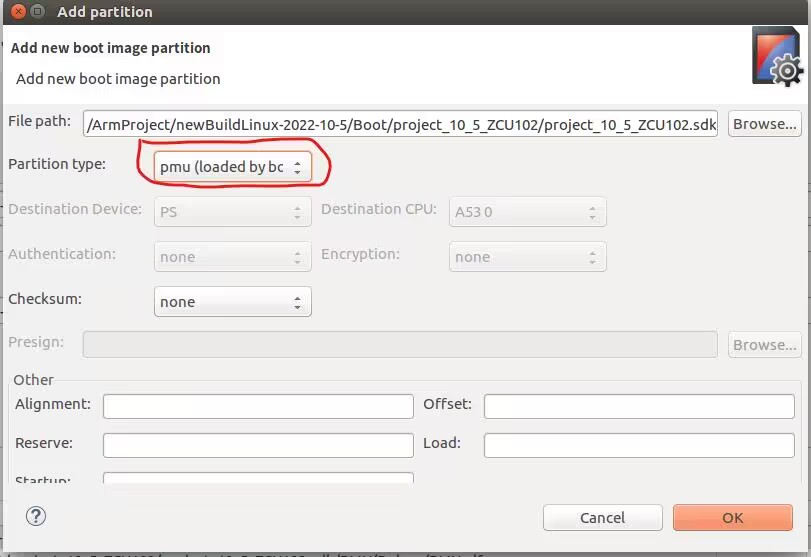


2）选择create new BIF file

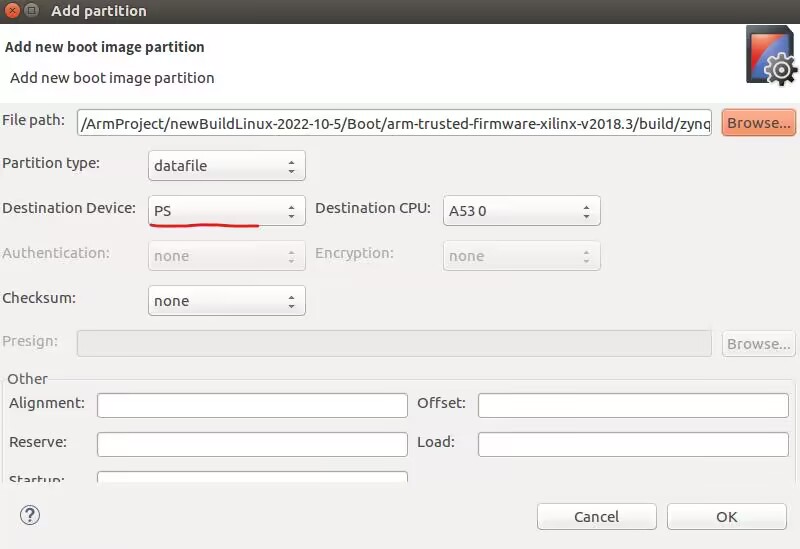
Boot image的五个文件：FSBL 文件FSBL.elf，PMU 文件PMU.elf，bit流文件design\_1\_wrapper.bit，ATF文件bl31.elf，u-boot文件u-boot.elf。按照之前记录的路径导入。选择合适的目标路径Output Path。



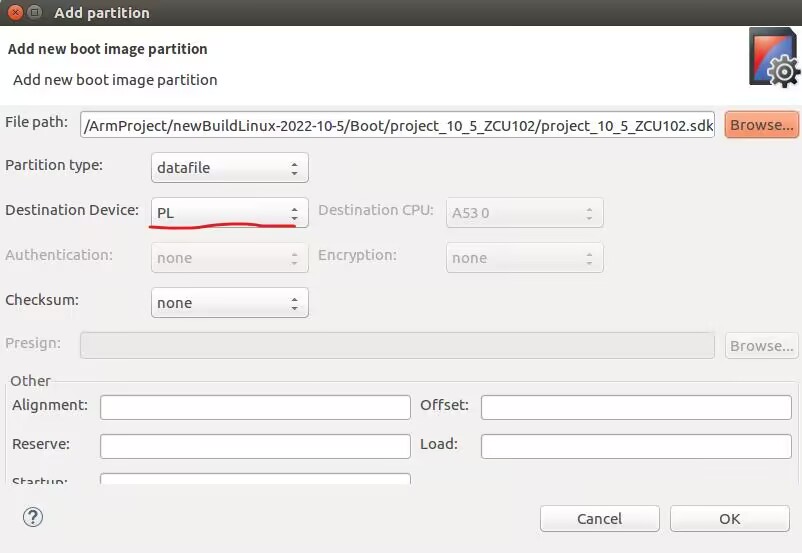
FSBL



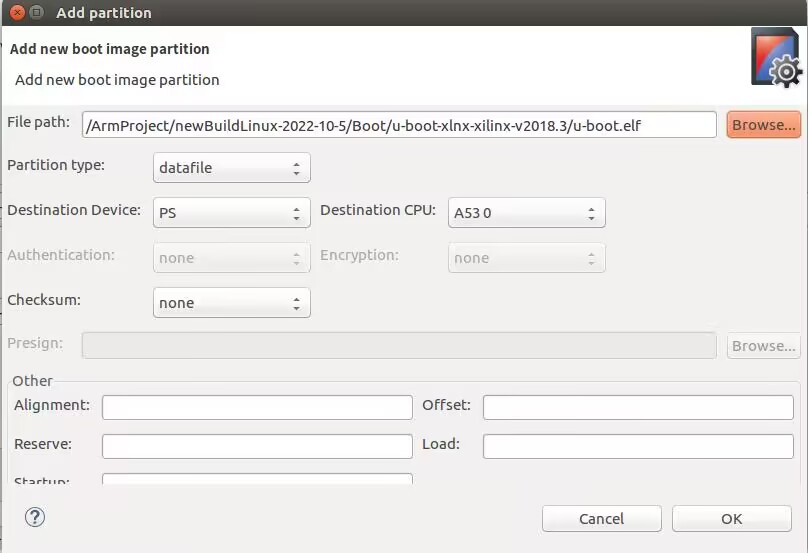
PMU



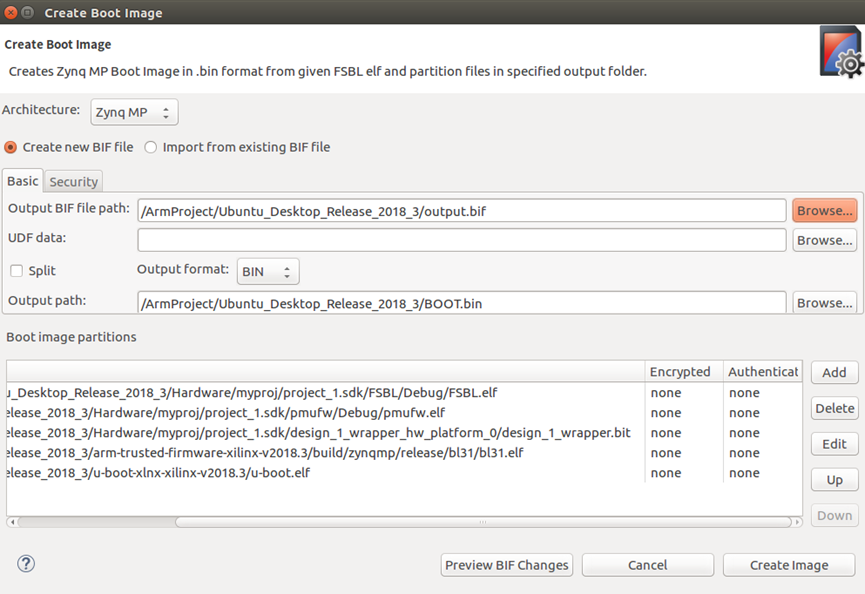
ATF



Bitstream

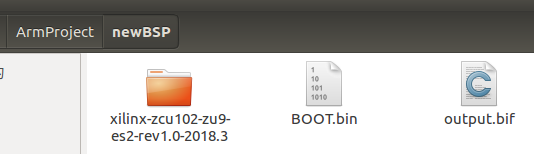


u-boot

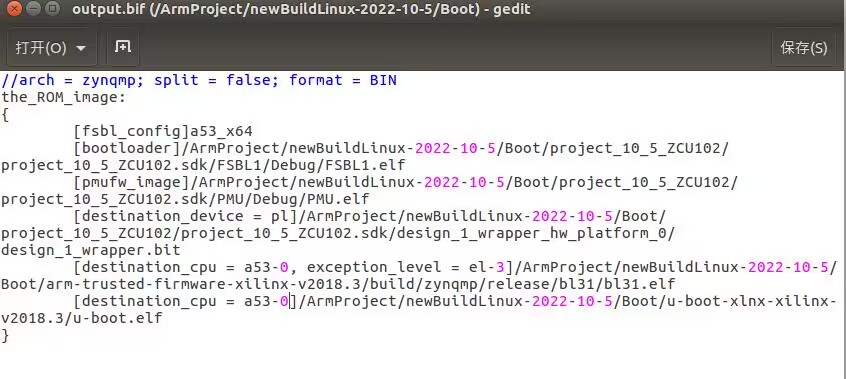


准备完成

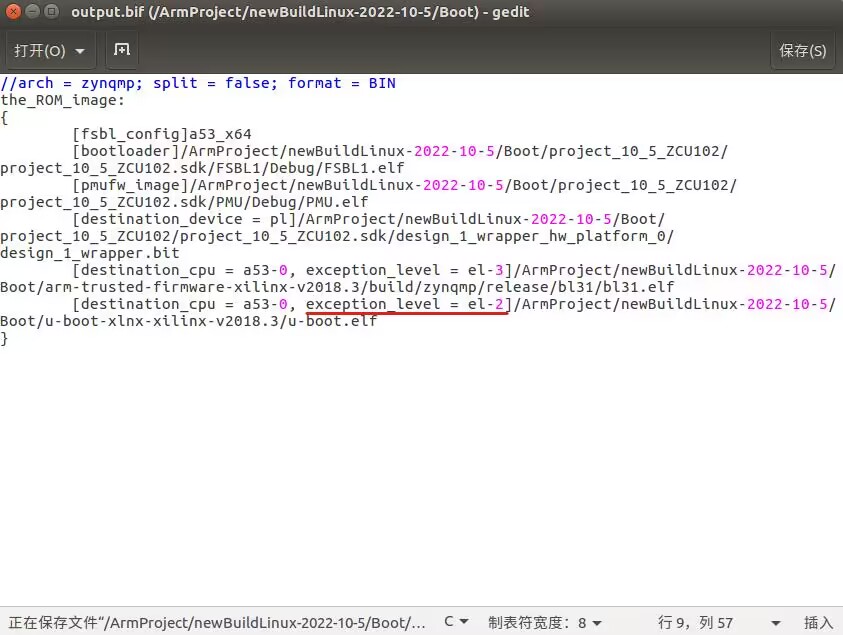
Create Image，可以发现目标路径Output Path中生成了output.bif与BOOT.bin文件，两个文件尚存在问题，需要先修改output.bif并用正确的output.bif生成正确的BOOT.bin。



3）找到output.bif打开，按照这样修改：

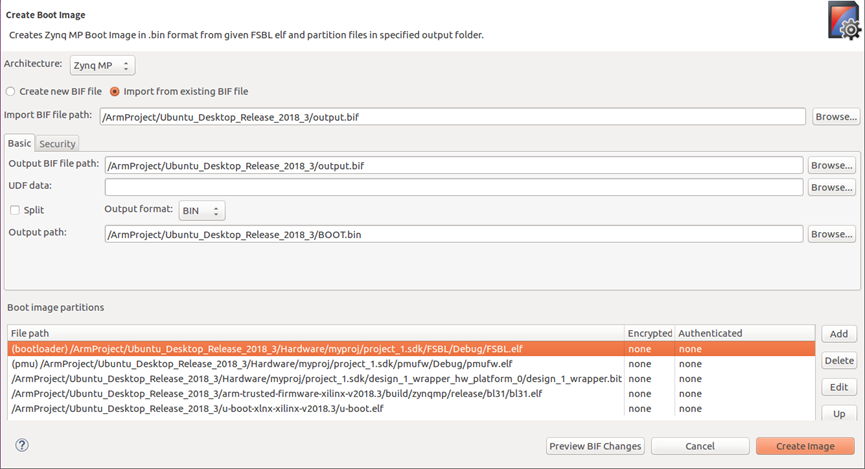


更改前



更改后

4）再次打开Xilinx->Create Boot Image，选择Import from existing BIF file，导入之前的output.bif，再一次Create Image，在目标路径Output Path得到BOOT.bin。



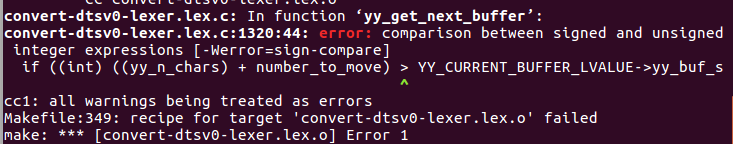
1.2制作DTB文件

1.2.1制作设备树编译器DTC（device tree compiler）

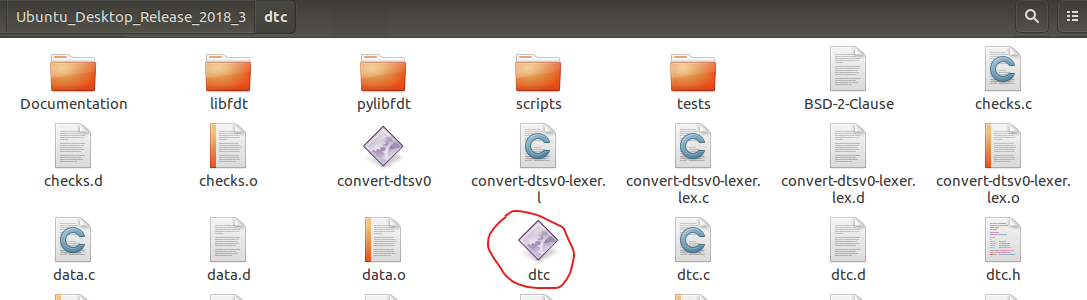
DTC用于将SDK软件生成的.dts、.dtsi文件编译为.dtb文件。

在<https://git.kernel.org/pub/scm/utils/dtc/dtc>网站下载dtc，放置在一个合适的路径下，本实验路径为/ArmProject/Ubuntu\_Desktop\_Release\_2018\_3/，下文以/dirDTC代替。

1）进入/dirDTC目录，执行make命令。在执行过程中可能会遇到报错：



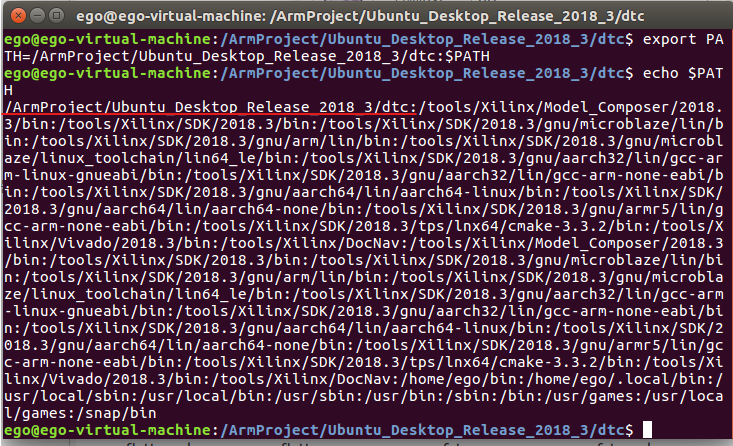
暂时的解决办法是根据错误信息将if ((int) ((yy\_n\_chars) + number\_to\_move) > YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size)修改为if ((unsigned int) ((yy\_n\_chars) + number\_to\_move) > YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size)。编译完成后，可以在/dirDTC路径下发现可执行文件dtc.elf。



2）将当前路径添加到$PATH变量中。这里不推荐使用export命令因为在后续的步骤中才会使用到DTC，而使用export命令添加$PATH变量的效果是暂时的，关掉shell后就会失效。建议编辑.bashrc文件 vim ~/.bashrc，在文件末尾添加(路径根据自己的情况填写)：

export PATH="/dirDTC /dtc:$PATH"

保存退出后输入命令source ~/.bashrc使添加立即生效。然后在使用export -p 或echo $PATH查看是否添加成功。

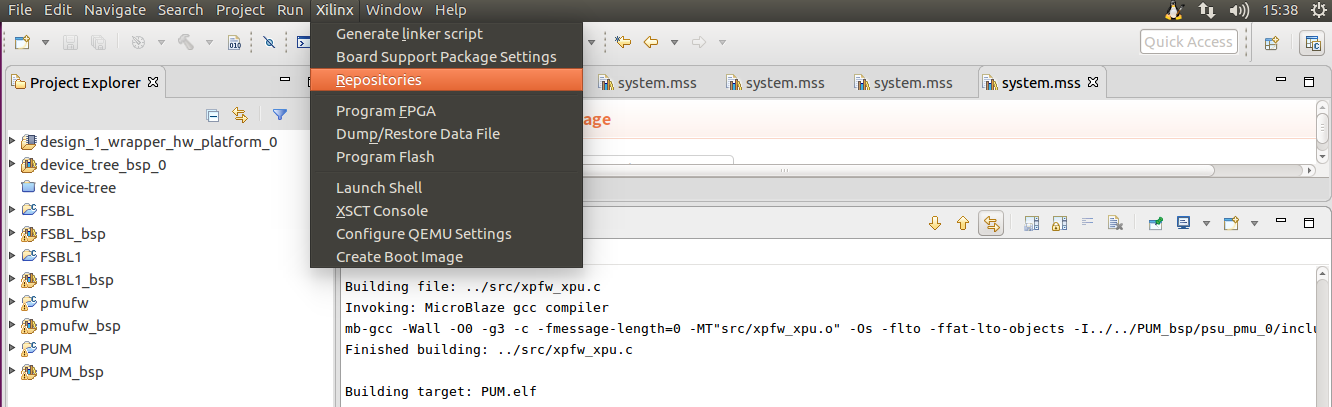


1.2.2制作DTB文件

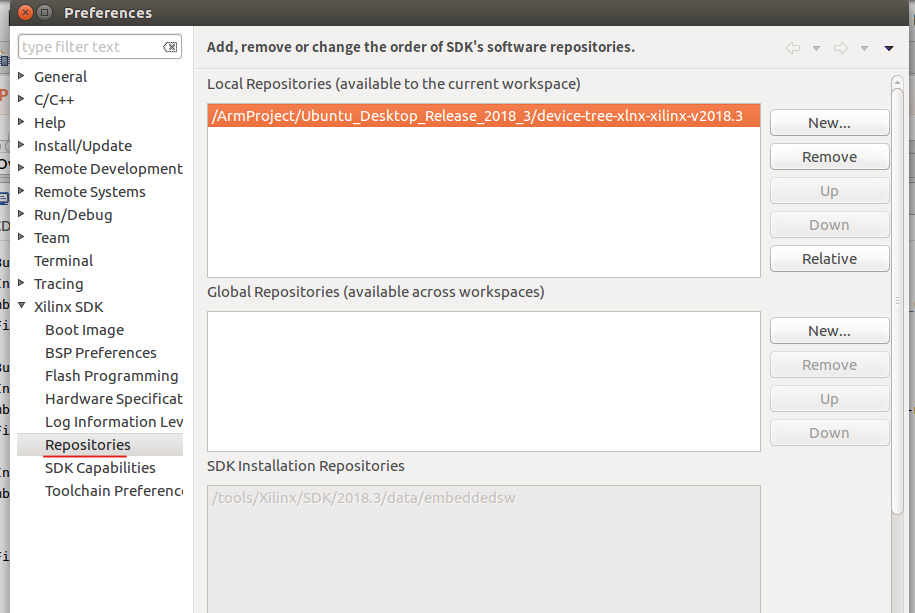
使用SDK软件制作DTS/DTSI文件。

从<https://github.com/Xilinx/device-tree-xlnx>下载DTG，解压。

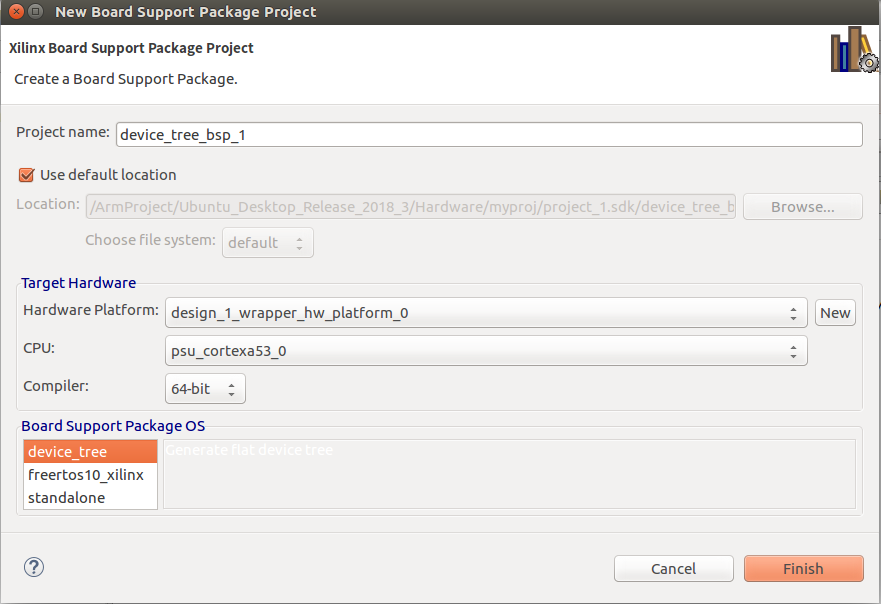
1）打开SDK-> Xilinx Tools > Repositories



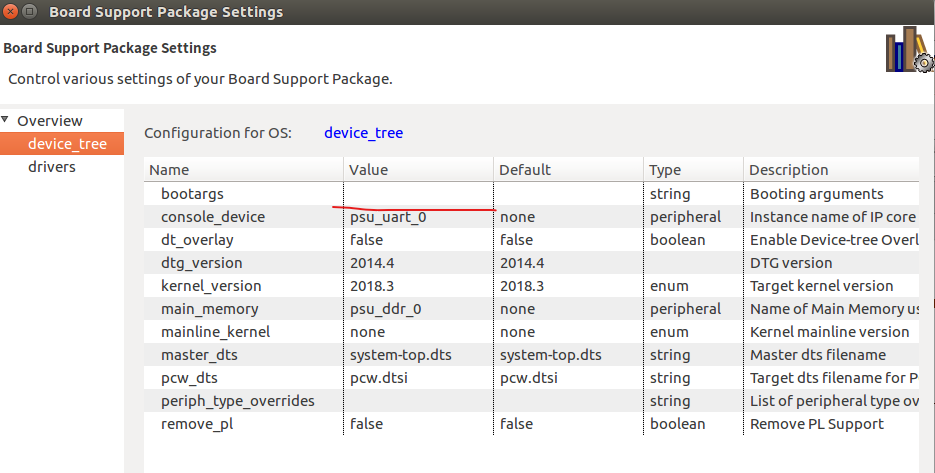
2）点击New->选择DTG所在的目录->点击OK。



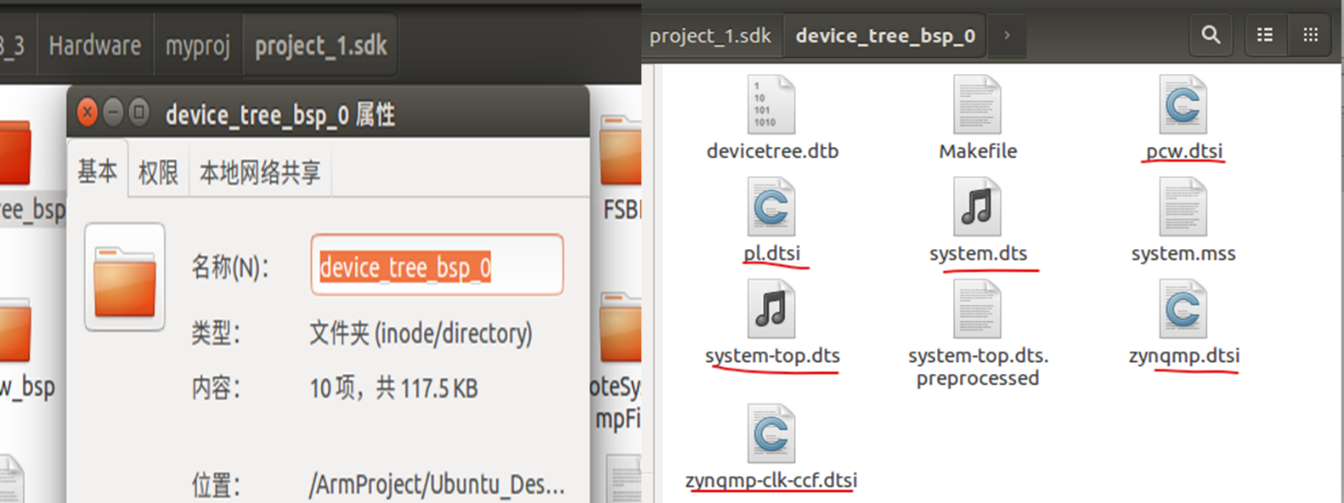
3）选项File->New->Board Support Package->Board Support Package OS: device-tree->Finish。



4）在弹出的窗口中，在bootargs一栏输入： console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=0 ，其他可保持不变，点击OK即可



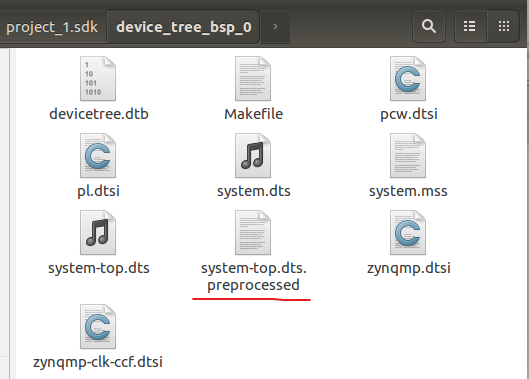
然后可以发现在当前目录下生成一个设备树文件夹，其中有数个.dts、.dtsi后缀文件。



5）预处理.dts/.dtsi文件，执行命令：

cpp -nostdinc -I include -I arch -undef -x assembler-with-cpp system-top.dts system-top.dts.preprocessed

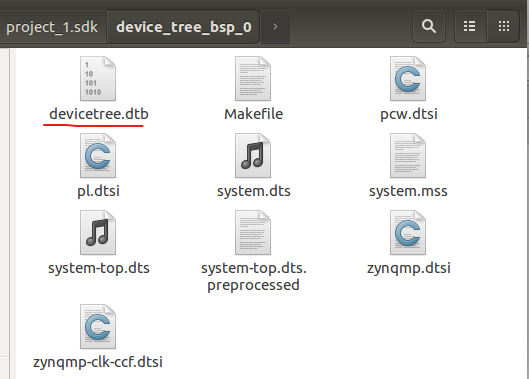
将归集dts文件到一个文件中，生成system-top.dts.preprocessed文件。



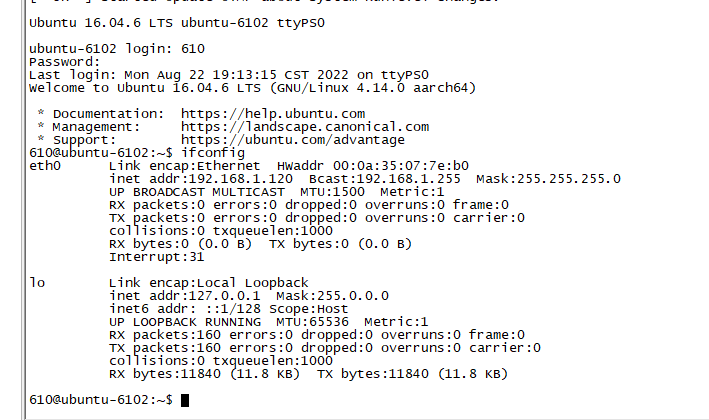
6）编译设备树，执行命令：

dtc -I dts -O dtb system-top.dts.preprocessed -o system.dtb

生成DTB文件devicetree.dtb。



移植全部完成后的验证->若设备树文件正确，ifconfig后应观察到以太网卡eth0正常。



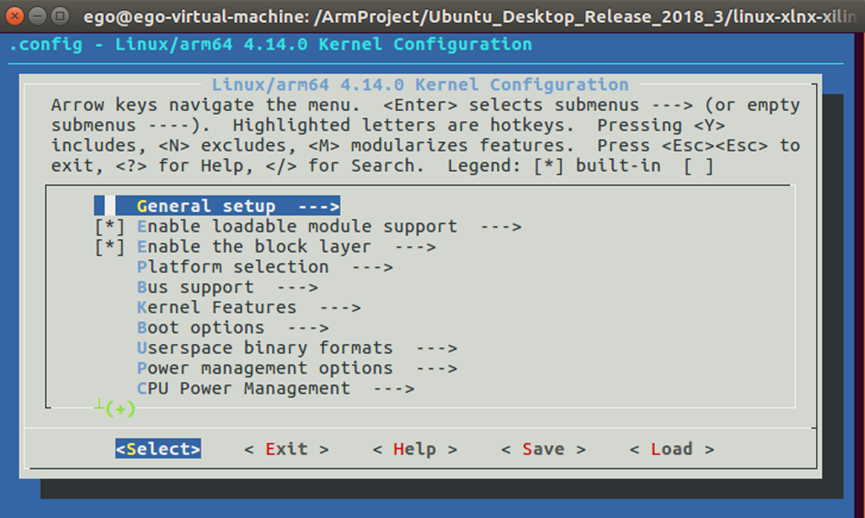
1.3编译内核

1）配置内核，执行命令make ARCH=arm64 xilinx\_zynqmp\_defconfig。



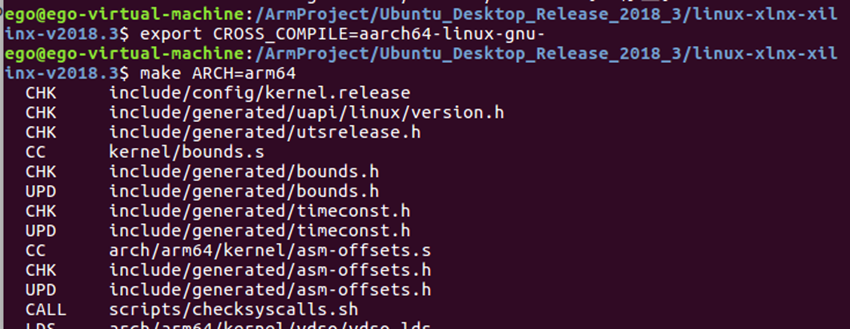
2）输入make ARCH=arm64 menuconfig。

执行该命令后会进入图形化的配置界面，暂时保持默认设置，直接exit。

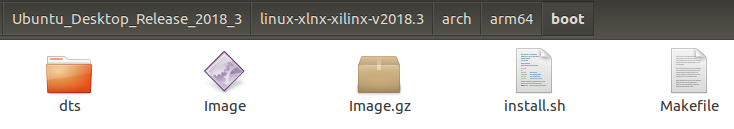


3）配置交叉编译路径export CROSS\_COMPILE=aarch64-linux-gnu-。

4）编译内核make ARCH=arm64。



5）编译完成后，可以在./linux-xilinx-v2018.3/arch/arm64/boot发现可执行文件Image.elf。



1.4拷贝到SD卡

将以上文件BOOT.bin（1.1.6生成），devicetree.dtb（1.2.2生成），Image.elf（1.3生成）拷贝到SD卡BOOT分区。至此，BOOT相关文件准备完毕。

2.制作根文件系统RootFS

2.0资料准备

从[官网](http://cdimage.ubuntu.com/ubuntu-base/releases/16.04/release/)下载ubuntu-base-arm64压缩文件。注意一定要下载arm64版本的。

新建目录：mkdir ubuntu-rootfs。模拟存放将要放置于SD卡文件系统分区的文件。

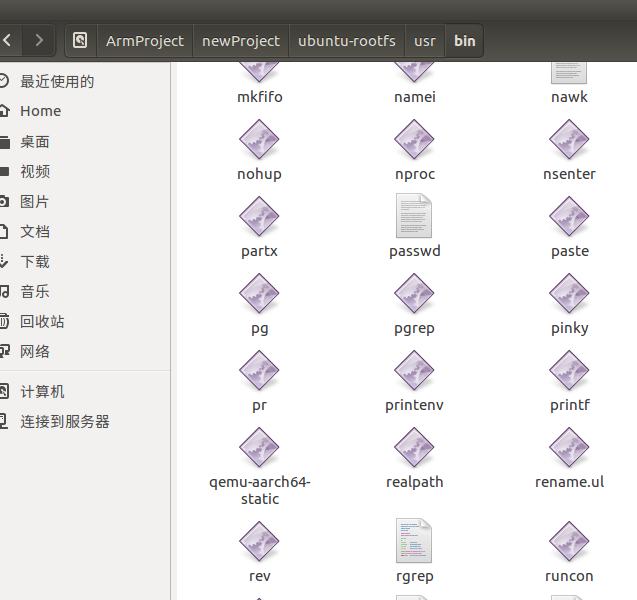
将下载来的压缩文件解压到新建的目录中：tar xvf ubuntu-base-16.04.6-base-arm64.tar.gz -C ./ubuntu-rootfs。

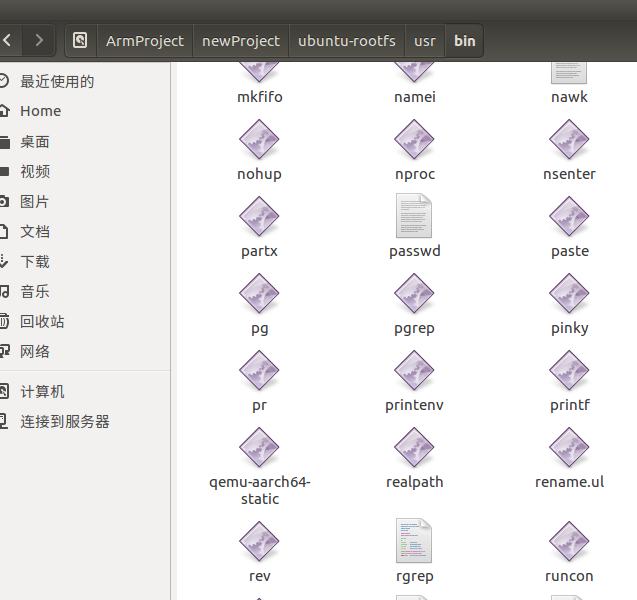
2.1制作步骤

1）安装qemu-user-static（此处安装到/usr/bin/目录下）：sudo apt-get install qemu-user-static。这是由于我们的开发机是x86架构的。如果想要制作适用于ARM的文件系统，就需要下载qemu-user-static来模拟arm环境。

进入之前创建的ubuntu-rootfs/路径下，将qemu-aarch64-static复制到ubuntu-rootfs/usr/bin：

cp /usr/bin/qemu-aarch64-static ./usr/bin





2）将本机的DNS配置文件复制到ubuntu-rootfs/etc/：

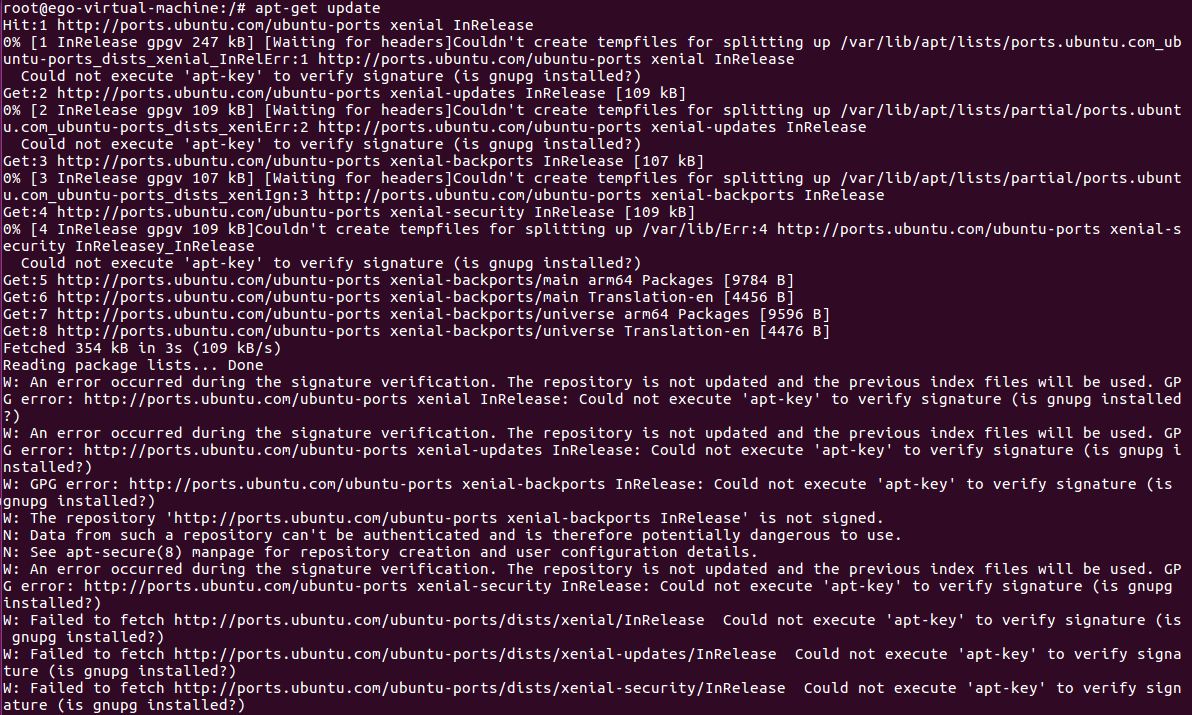
cp -b /etc/resolv.conf ./etc/

这样进入chroot环境后才能正常访问网络。

3）修改ubuntu-rootfs/tmp/文件权限：

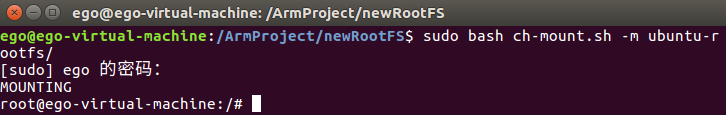
chmod 777 ./tmp

这是为了进入chroot环境后可以正常的更新软件源。如果跳过该步在更新软件源时会遇到signature verification error的问题，导致无法更新软件源。之后执行命令cd ..切换到ubuntu-rootfs/的上级目录。



4）挂载ubuntu-rootfs，进入chroot环境（此处需要一个脚本文件ch-mount.sh，放在ubuntu-rootfs的上级目录，其具体内容见文末）：

sudo bash ch-mount.sh -m ubuntu-rootfs/



接下来的步骤都是在此虚拟环境中进行，直到退出chroot环境。

5）挂载后更新软件源： apt-get update

6）安装必要的工具包：

apt-get install language-pack-en-base（可选）

apt-get install dialog

apt-get install language-pack-zh-hans-base

apt-get install vim

apt-get install sudo ssh ethtool iputils-ping net-tools ifupdown

在安装language-pack-en-base后需要配置local文件：

vim /etc/default/locale

LANG="en\_US.UTF-8"

LANGUAGE="en\_US:en"

LC\_ALL="en\_US.UTF-8"

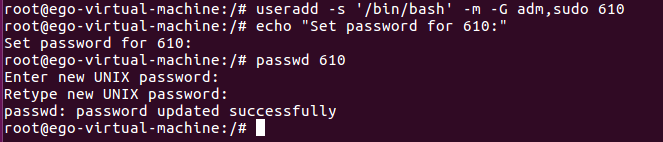
7）添加用户：

useradd -s '/bin/bash' -m -G adm,sudo 610

echo "Set password for 610:"

passwd 610

将用户“610”的密码设置为610



echo "Set password for root:"

passwd root

将root 的密码设置为 root。

8）设置主机名与入口ip

echo 'ubuntu-610' > /etc/hostname

echo "127.0.0.1 localhost" >> /etc/hosts

echo "127.0.1.1 ubuntu-610" >> /etc/hosts

9）配置登录的串口（关键）

cp /lib/systemd/system/serial-getty@.service [/lib/systemd/system/serial-getty@ttyPS0.service](mailto:/lib/systemd/system/serial-getty@ttyPS0.service)

vim [/lib/systemd/system/serial-getty@ttyPS0.service](mailto:/lib/systemd/system/serial-getty@ttyPS0.service)



ln -s /lib/systemd/system/serial-getty@ttyPS0.service /etc/systemd/system/getty.target.wants/serial-getty@ttyPS0.service

10）配置网络

echo auto eth0 > /etc/network/interfaces.d/eth0

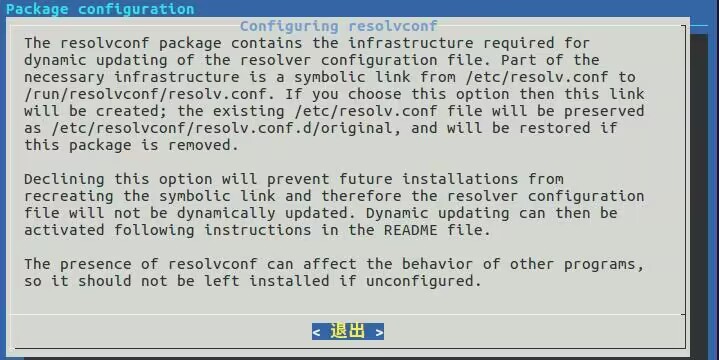
echo iface eth0 inet dhcp >> /etc/network/interfaces.d/eth0

11）设置自动更新DNS

apt-get install resolvconf

dpkg-reconfigure resolvconf

会弹出configuring resolvconf等窗口，都直接选退出。

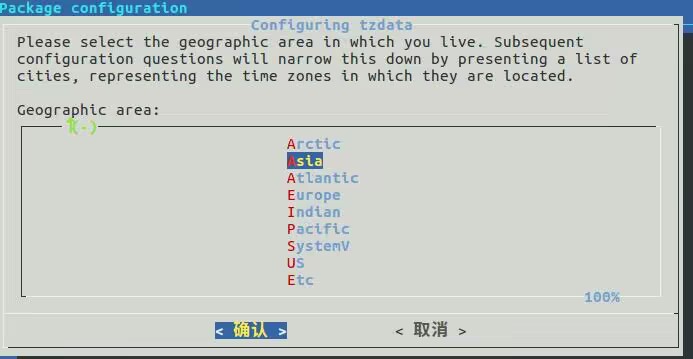


12）设置时区

apt-get install tzdata

dpkg-reconfigure tzdata

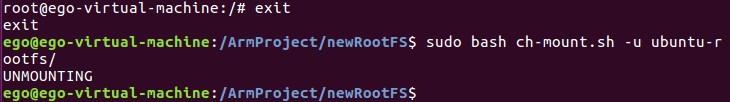
弹出窗口，按需选择即可。



13）退出chroot环境，取消挂载

exit

sudo bash ch-mount.sh -u ubuntu-rootfs/



2.2拷贝到SD卡

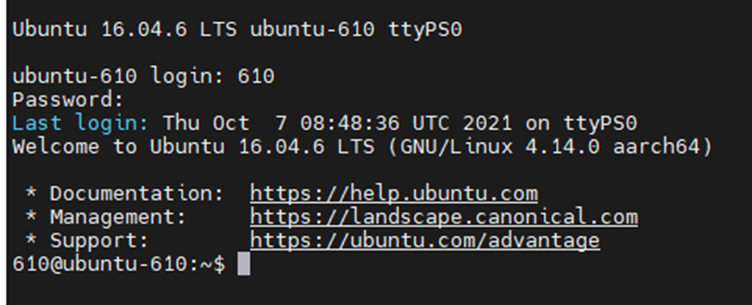
将ubuntu-rootfs路径下全部内容拷贝到SD卡（其中/media/ego/RootFS路径可以在主机/虚拟机文件系统目录左侧“计算机”位置附近“RootFs”处找到，右键->属性，“ego”为本实验用户名，请实验者替换成自己的用户名）

sudo rsync -avSH （自己的ubuntu-rootfs上级路径）/ubuntu-rootfs/\* /media/ego/RootFS

至此，RootFS相关文件也已准备完毕。包含整个Linux系统和文件系统的SD卡内容已齐全，可以上板验证。

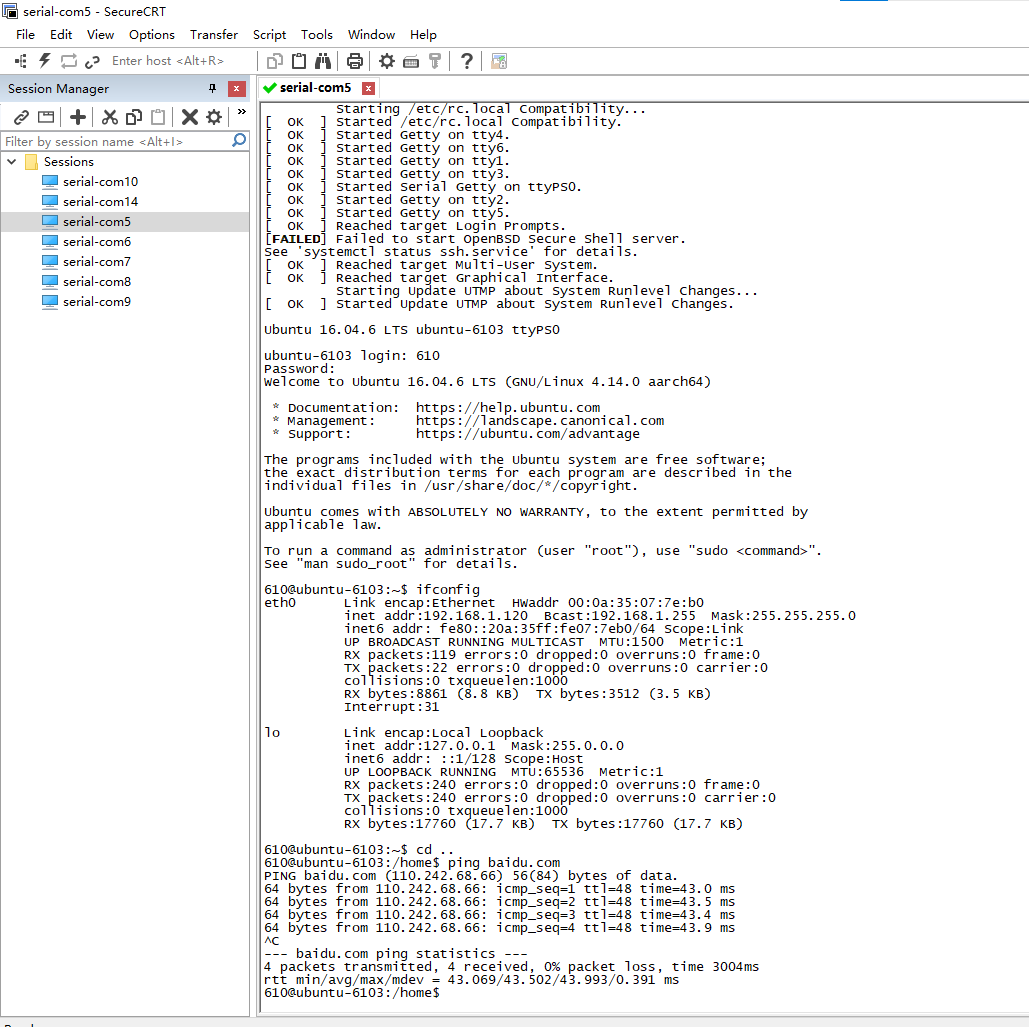
3.启动验证

插入SD卡上电，发现文件系统可以启动并登录：



上电时要插上网线，因为启动过程中会拉起网口，如果不插网线会等待5分钟再跳过该步骤，导致启动变慢（后续可以设置缩短等待时间）。

可以执行ifconfig、ping命令查看网卡设备是否正常工作。



复现实验测试，与其他配图用户名有所不同

4.后续完善

登录后进行更新软件、安装工具包network-manager、 rsyslog。这两个工具包具有网络设备管理、系统日志管理的功能，是十分必要的工具包。而不在chroot环境下安装，而在启动之后再这两个包的原因是在chroot下安装会导致启动过程卡死。原因暂时还没摸索清楚。

编写一个简单的C++程序，编译生成可执行文件并运行

