Linux内核移植

班级：数字媒体技术17-1 学号：20170106109 姓名：李磊

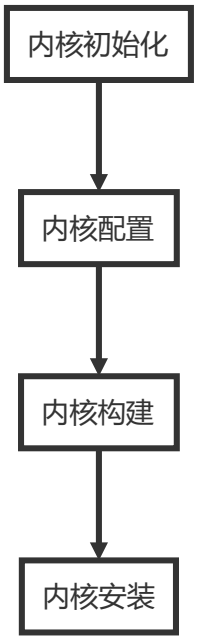
前言

使用的开发板：正点原子开发板

使用的linux内核源码：linux-imx-rel_imx_4.1.15_2.1.0_ga.tar.bz2

简述

linux内核移植流程图：



内核简介

- Linux内核是由许多逻辑构成要素组成的综合软件
- 由必要的调度程序、文件系统、内存管理器、网络系统等许多子系统组成。
- 通过构建过程生成二进制映像文件zImage--烧写到存储芯片中--通过uboot启动zImage

内核初始化

- 为了构建内核，用户必须先完成内核初始化过程
- 通过kernel.org等网站获得tar.gz格式的内核初始源代码
- 内核源代码解压后会内核初始状态的内核源代码树
- make mrproper命令回退，make distclean命令初始化
- 不能立即对内核进行编译，否则会引起内核严重错误。

内核配置

- 内核配置过程也是适当选择与自身系统相吻合的各种内核要素的过程。如我们使用的正点原子开发板与NXP官方开发板存在差别，需要进行一些配置。

- 使用menuconfig对象进行配置。make menuconfig提供一种基于文本的图形界面，如果使用该对象，需要安装两个软件包build-essential和libncurses5-dev。
- 配置后会生成.config文件，文件内形式为CONFIG_XXX（y，n或m），也称为kconfig

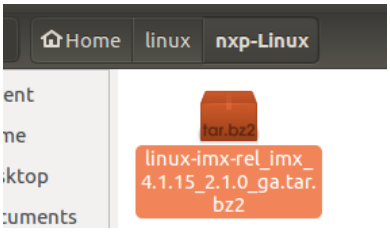
内核构建与编译

- 利用kconfig完成内核配置，并准备好具有自身内核配置的.config文件后，即可构建内核（编译内核并链接二进制文件，由此生成一个二进制文件zImage的一系列过程）。
- 通过顶层的Makefile文件进行编译
- 在Ubuntu环境下，利用make -kpkg命令完成内核的编译

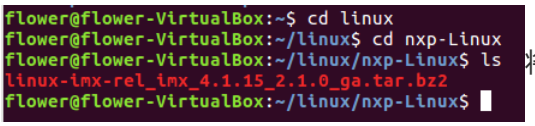
步骤

1.内核初始化

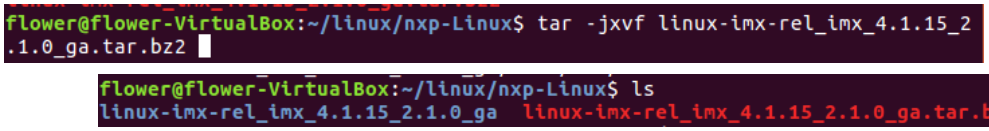
1. 在windows中准备好NXP（恩智浦）的linux内核源码。使用FileZilla传输到 linux 系统中。这里，我们将新建了一个名为nxp-Linux的文件夹，将内核放到了这个文件夹下。



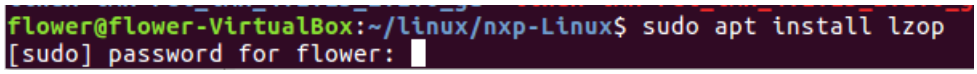
2. 打开终端，进入/linux/nxp-Linux目录。



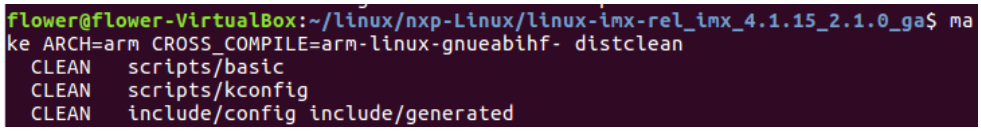
将源码进行解压操作。输入`tar -jxvf linux-imx-rel_imx_4.1.15_2.1.0_ga.tar.bz2`命令进行解压。



3. 安装相关包。输入`sudo apt install lzop`，安装这个包后才能成功编译linux内核。



4. 输入`make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean`命令进行初始化操作



2.内核配置

1. 我们使用的是imx芯片，所以要针对这个芯片进行一些配置。

输入 `make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- imx_v7_defconfig` 命令进行配置

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga$ ma
ke ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- imx_v7_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

可以在这个配置的基础上进行修改，实现在linux添加我们的开发板的目的。

2. NPX Linux内核源码的结构如下图：

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga$ ls
arch      Documentation  init      lib          README           sound
block     drivers        ipc       MAINTAINERS  REPORTING-BUGS   tools
COPYING   firmware       Kbuild    Makefile     samples          usr
CREDITS   fs             Kconfig   mm           scripts          virt
crypto    include        kernel    net          security
```

各文件夹及其功能内容如下表：

文件夹名称	功能				
arch	CPU相关信息	**drivers**	内置驱动	**include**	头文件
block	块设备相关	**firmware**	硬件管理	**init**	内核启动处理
Documentation	文本文档	**fs**	文件系统	**ipc**	进程间通信相关
mm	内存管理	**net**	网络相关	**security**	内核安全
sound	声音相关	**tools**	内核交互工具	**samples**	模块代码

3. 使用的开发板为**正点原子**开发板，需要修改一些NXP官方linux源码配置

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga$ cd
 arch/arm/configs
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga/arc
h/arm/configs$
```

输入 `cp imx_v7_mfg_defconfig imx_alientek_emmc_defconfig` 命令，拷贝 `imx_v7` 的配置文件并命名为 `imx_alientek_emmc_defconfig`，以后imx_alientek_emmc_defconfig就是正点原子的EMMC版开发板默认配置文件了。

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga/arc
h/arm/configs$ cp imx_v7_mfg_defconfig imx_alientek_emmc_defconfig
```

4. **添加开发板对应的设备树。**跳转到boot/dts，输入 `cp imx6ull-14x14-evk.dts imx6ull-alientek-emmc.dts` 命令，拷贝 `imx6ull-14x14` 的设备树并命名为 `imx6ull-alientek-emmc.dts`

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga/arc
h/arm/boot/dts$ cp imx6ull-14x14-evk.dts imx6ull-alientek-emmc.dts
```

5. 使用VS Code打开解压后的 `linux-imx-rel_imx_4.1.15_2.1.0_ga` 文件夹。由于之前新添加了 `imx_alientek_emmc_defconfig` 配置文件和对应的设备树需要修改设备树有关的Makefile。.dts是设备树源码文件，编译Linux的时候会将其编译为.dtb文件。在arch/arm/boot/dtc中找到Makefile文件并在VS Code中打开。添加以下代码 `imx6ull-alientek-emmc.dtb \`。添加完毕后，linux内核在编译的时候才能从imx6ull-alientek-

emmc.dts编译出imx6ull-alientek-emmc.dtb文件。

```
M Makefile ●
arch > arm > boot > dts > M Makefile
415     imx6ull-14x14-ddr3-arm2-usb.dtb \
416     imx6ull-14x14-ddr3-arm2-wm8958.dtb \
417     imx6ull-14x14-evk.dtb \
418     imx6ull-14x14-evk-btwifi.dtb \
419     imx6ull-14x14-evk-emmc.dtb \
420     imx6ull-14x14-evk-gpmi-weim.dtb \
421     imx6ull-14x14-evk-usb-certi.dtb \
422     imx6ull-alientek-emmc.dtb \
423     imx6ull-9x9-evk.dtb \
424     imx6ull-9x9-evk-btwifi.dtb \
425     imx6ull-9x9-evk-ldo.dtb
```

至此，设备树相关工作配置完成。

6. 我们的开发板eMMC作为flash闪存，而nxp官方开发板使用4根线连接CPU与闪存芯片，正点原子采用8根线，传送速度更快，所以需要更改闪存相关驱动配置。

   打开我们刚才创建的设备树文件`imx6ull-alientek-emmc.dts`，添加如下代码

```
734 &usdhc2 {
735     pinctrl-names = "default", "state_100mhz", "state_200mhz";
736     pinctrl-0 = <&pinctrl_usdhc2_8bit>;
737     pinctrl-1 = <&pinctrl_usdhc2_8bit_100mhz>;
738     pinctrl-2 = <&pinctrl_usdhc2_8bit_200mhz>;
739     bus-width = <8>;
740     non-removable;
741     status = "okay";
742 };
```

   总线宽度设置为8。

7. **修改网络驱动。**正点原子开发板的网络和NXP官方的网络硬件上有所不同，网络PHY芯片由KSZ8081换为了LAN8720A，两个网络PHY芯片的复位IO也不同，所以要做一些修改。

   - **修改LAN8720的复位引脚驱动。**ENET1复位引脚ENET1_RST连接在MX6ULL的SNVS_TAMPER7引脚上。ENET2的复位引脚ENET2_RST连接在SNVS_TAMPER8引脚上。打开`imx6ull-alientek-emmc.dts`，找到如下代码：

```
584         pinctrl_spi4: spi4grp {
585                     fsl,pins = <
586                             MX6ULL_PAD_BOOT_MODE0__GPIO5_IO10       0x70a1
587                             MX6ULL_PAD_BOOT_MODE1__GPIO5_IO11       0x70a1
588                             /*MX6ULL_PAD_SNVS_TAMPER7__GPIO5_IO07     0x70a1
589                             MX6ULL_PAD_SNVS_TAMPER8__GPIO5_IO08     0x80000000*/
590                     >;
591                 };
```

     发现spi4grp使用了这两个引脚，所以要把588行和589行删除或注释掉。

   - 找到`spi4`中关于两个引脚的代码，删除或注释

```
spi4 {
    compatible = "spi-gpio";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_spi4>;
    pinctrl-assert-gpios = <&gpio5 8 GPIO_ACTIVE_LOW>;
    status = "okay";
    gpio-sck = <&gpio5 11 0>;
    gpio-mosi = <&gpio5 10 0>;
    cs-gpios = <&gpio5 7 0>;
    num-chipselects = <1>;
    #address-cells = <1>;
    #size-cells = <0>;
```

- 现在需要将GPIO5_I07和GPIO5_I08分别作为ENET1和ENET2的复位引脚。添加如下代码：

```
308
309     pinctrl_enet1: enet1grp {
310         fsl,pins = <
311             MX6UL_PAD_ENET1_RX_EN__ENET1_RX_EN  0x1b0b0
312             MX6UL_PAD_ENET1_RX_ER__ENET1_RX_ER  0x1b0b0
313             MX6UL_PAD_ENET1_RX_DATA0__ENET1_RDATA00 0x1b0b0
314             MX6UL_PAD_ENET1_RX_DATA1__ENET1_RDATA01 0x1b0b0
315             MX6UL_PAD_ENET1_TX_EN__ENET1_TX_EN  0x1b0b0
316             MX6UL_PAD_ENET1_TX_DATA0__ENET1_TDATA00 0x1b0b0
317             MX6UL_PAD_ENET1_TX_DATA1__ENET1_TDATA01 0x1b0b0
318             MX6UL_PAD_ENET1_TX_CLK__ENET1_REF_CLK1  0x4001b031
319             MX6UL_PAD_SNVS_TAMPER7__GPIO5_IO07 0x10b0
320         >;
321     };
323     pinctrl_enet2: enet2grp {
324         fsl,pins = <
325             MX6UL_PAD_GPIO1_IO07__ENET2_MDC     0x1b0b0
326             MX6UL_PAD_GPIO1_IO06__ENET2_MDIO    0x1b0b0
327             MX6UL_PAD_ENET2_RX_EN__ENET2_RX_EN  0x1b0b0
328             MX6UL_PAD_ENET2_RX_ER__ENET2_RX_ER  0x1b0b0
329             MX6UL_PAD_ENET2_RX_DATA0__ENET2_RDATA00 0x1b0b0
330             MX6UL_PAD_ENET2_RX_DATA1__ENET2_RDATA01 0x1b0b0
331             MX6UL_PAD_ENET2_TX_EN__ENET2_TX_EN  0x1b0b0
332             MX6UL_PAD_ENET2_TX_DATA0__ENET2_TDATA00 0x1b0b0
333             MX6UL_PAD_ENET2_TX_DATA1__ENET2_TDATA01 0x1b0b0
334             MX6UL_PAD_ENET2_TX_CLK__ENET2_REF_CLK2  0x4001b031
335             MX6UL_PAD_SNVS_TAMPER8__GPIO5_IO08 0x10b0
336         >;
337     };
```

- **修改LAN8720A的PHY地址。**ENET1的地址为0x0，ENET2的地址为0x1。

```
194         ethphy0: ethernet-phy@0 {
195             compatible = "ethernet-phy-ieee802.3-c22";
196             reg = <0>;
197         };
198
199         ethphy1: ethernet-phy@1 {
200             compatible = "ethernet-phy-ieee802.3-c22";
201             reg = <1>;
202         };
```

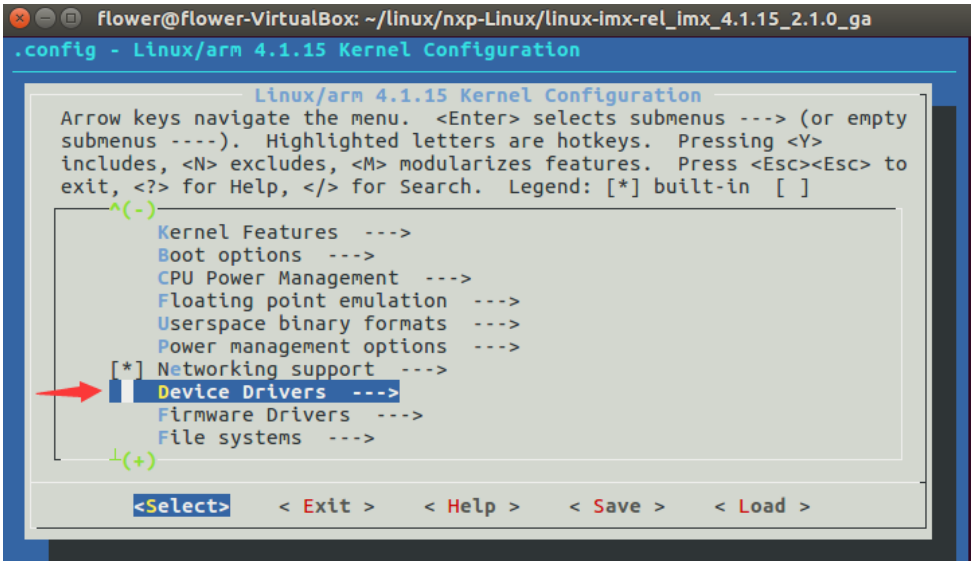
```
&fec1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet1>;
    phy-mode = "rmii";
    phy-handle = <&ethphy0>;
    phy-reset-gpios = <&gpio5 7 GPIO_ACTIVE_LOW>;
    phy-reset-duration = <200>
    status = "okay";
};

&fec2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_enet2>;
    phy-mode = "rmii";
    phy-handle = <&ethphy1>;
    status = "okay";
    phy-reset-gpios = <&gpio5 8 GPIO_ACTIVE_LOW>;
    phy-reset-duration = <200>
```

添加复位引脚信息，设置低电平有效，持续时间200ms

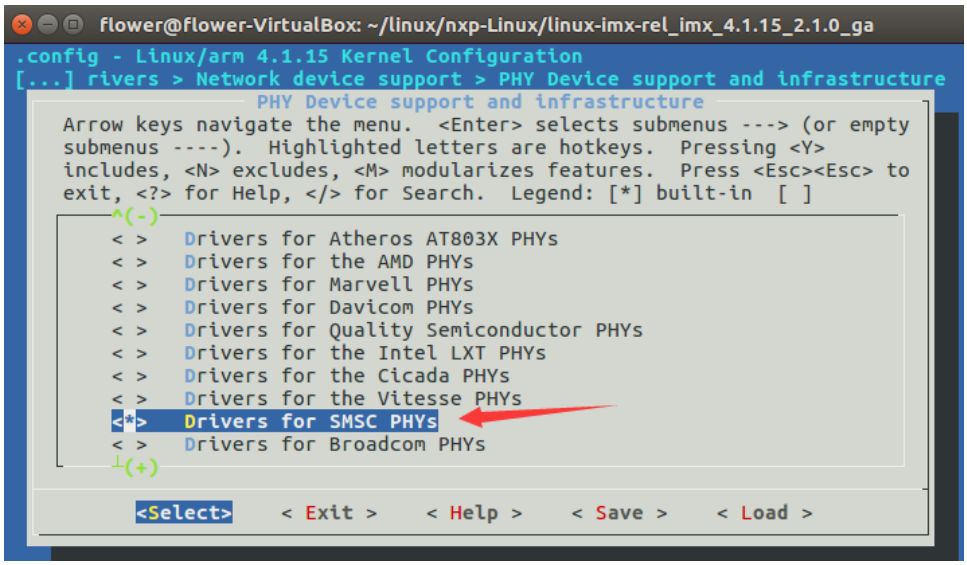
- **修改fec_main.c文件。**引脚信息修改完之后，修改驱动信息。打开`/drivers/net/ethernet/freescale/fec_main.c`，找到`fec_probe`函数（3439行），添加软复位代码。

```
void __iomem *IMX6UL_ENET1_TX_CLK;
void __iomem *IMX6UL_ENET2_TX_CLK;

IMX6UL_ENET1_TX_CLK = ioremap(0x020E00DC, 4);
writel(0x14, IMX6UL_ENET1_TX_CLK);

IMX6UL_ENET2_TX_CLK = ioremap(0x020E00FC, 4);
writel(0x14, IMX6UL_ENET2_TX_CLK);
```

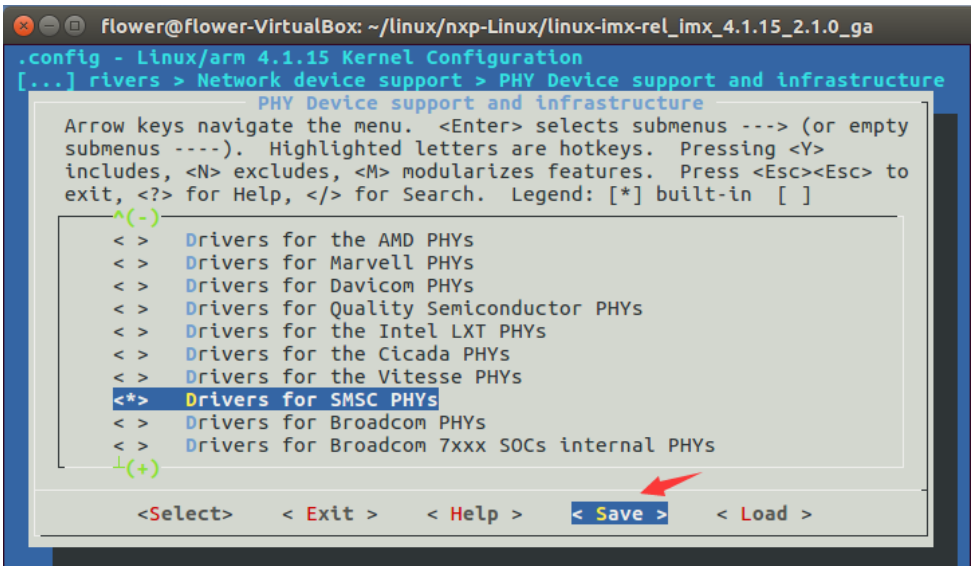
- 代码部分配置好后，需要在config中配置驱动。输入`make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig`命令，打开界面配置。通过上下箭头键找到`Device Drivers`，按回车进入。

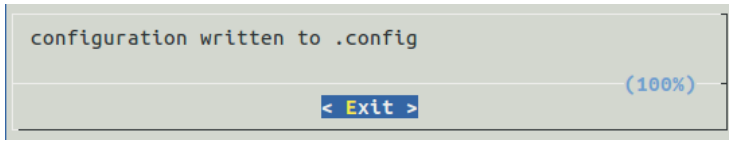


找到 `Network device support> PHY Device support and infarstructure> Drivers for SMSC PHYs` 按y选中



之后按一下回车，在下方使用左右箭头移动到`Save`，回车，保存并退出





- **修改smsc.c文件。**不修改的话文件系统挂在成功率会很低。在`drivers/net/phy/`目录中找到smsc.c文件。找到`smsc_phy_reset`函数，这个函数是 SMSC_PHY 的复位函数。可以看到默认只有驱动在power down工作模式时才会进行软复位，现在我们将其修改为任何情况都软复位。将执行软复位的代码从if中提取出来。

```
69      if ((rc & MII_LAN83C185_MODE_MASK) == MII_LAN83C185_MODE_POWERDOWN) {
70          int timeout = 50000;
71
72          /* set "all capable" mode and reset the phy */
73          rc |= MII_LAN83C185_MODE_ALL;
74          phy_write(phydev, MII_LAN83C185_SPECIAL_MODES, rc);
75      }
76      phy_write(phydev, MII_BMCR, BMCR_RESET);
77
78      /* wait end of reset (max 500 ms) */
79      do {
80          udelay(10);
81          if (timeout-- == 0)
82              return -1;
83          rc = phy_read(phydev, MII_BMCR);
84      } while (rc & BMCR_RESET);
85      return 0;
86  }
```

3.内核编译

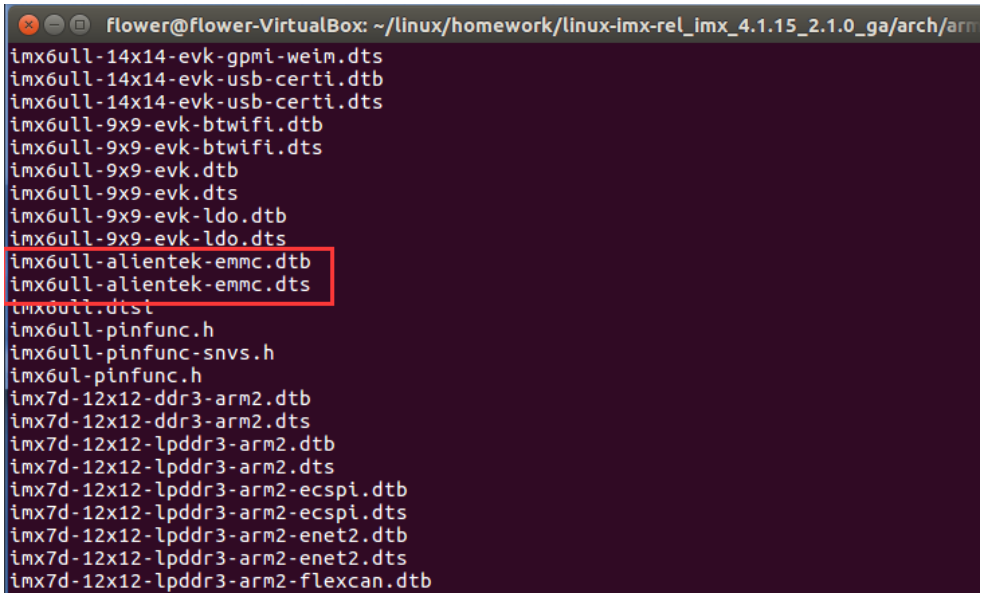
配置成功后，就可以进行编译了。输入`make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- imx_alientek_emmc_defconfig`命令，使用我们设置好的配置。

```
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga$ ma
ke ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- imx_alientek_emmc_defconfig
#
# configuration written to .config
#
flower@flower-VirtualBox:~/linux/nxp-Linux/linux-imx-rel_imx_4.1.15_2.1.0_ga$ ma
ke ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all -j16
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
  CC      scripts/mod/empty.o
  CC      scripts/mod/devicetable-offsets.s
```

输入`make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all -j16`命令进行编译。

```
flower@flower-VirtualBox: ~/linux/homework/linux-imx-rel_imx_4.1.15_2.1.0_ga
  CC      lib/libcrc32c.mod.o
  LD [M]  drivers/i2c/algos/i2c-algo-pca.ko
  LD [M]  drivers/i2c/algos/i2c-algo-pcf.ko
  LD [M]  drivers/input/evbug.ko
  LD [M]  drivers/input/mouse/psmouse.ko
  LD [M]  drivers/input/serio/serport.ko
  LD [M]  drivers/net/wireless/brcm80211/brcmutil/brcmutil.ko
  LD [M]  drivers/net/wireless/brcm80211/brcmfmac/brcmfmac.ko
  LD [M]  drivers/rpmsg/imx_rpmsg_pingpong.ko
  LD [M]  drivers/rpmsg/imx_rpmsg_tty.ko
  LD [M]  fs/binfmt_misc.ko
  LD [M]  fs/fat/msdos.ko
  LD [M]  fs/isofs/isofs.ko
  LD [M]  lib/crc-ccitt.ko
  LD [M]  lib/libcrc32c.ko
  LD [M]  fs/udf/udf.ko
  LD [M]  lib/crc-itu-t.ko
  LD [M]  lib/crc7.ko
  LD [M]  fs/nls/nls_iso8859-15.ko
  AS      arch/arm/boot/compressed/piggy.lzo.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
flower@flower-VirtualBox:~/linux/homework/linux-imx-rel_imx_4.1.15_2.1.0_ga$
```

等待一段时间后，生成了zImage文件，编译成功。



同时，也生成了dtb文件。

linux内核移植过程结束。

总结

本次大作业，我成功在linux环境下进行了linux内核的移植。使用了NXP提供的官方源码，并加以修改使之能在正点原子开发板上运行。在此过程中，我遇到过很多问题，如：编译的时候出现了 `recipe for target '.dtbs'` 之类的错误，经过排查发现出现了语法错误。我也学到了很多linux知识，掌握了linux内核移植的过程和方法，对我以后linux的学习有很大的帮助。