

分享

100ask_imx6ul 开发板实战记录



星星之火 最后更新于 20200218

写在前面

本文档作者为韦东山老师粉丝，曾学习 1、2、3、4 期后，找到理想工作，目前从事 linux 内核网络相关工作。自己曾画过 imx6u 核心板，对该芯片有一些情怀。百问网新出 imx6ull 开发板后，本人决定基于此开发板，结合曾经做过的实验，在该板上重新操作并记录下来，并长期为入门学员提供答疑。

请大家学习首先还是参考官方提供的文档和视频教程为准，该文档仅作为个人分享及辅助学习。

- **文档目的：**

记录基于此开发板的**实战过程**，为新人入门提供便捷，把知识总结下来，把问题记录下来，并和大家一起深入学习 linux。

- **主要包括：**

开发**环境搭建**、驱动实验、**内核知识点**、调试技巧、问题记录...

- **组织形式**

一个知识点一个小节，不区分大分章节，命名为：第*节 XXXX 实验。

如（第 1 节 一步一步定位三者互 ping 问题实验），随着积累，不断更新新的小节。每节包括 **1、实验目的；2、实验过程；3、实验结果；4、实验总结；5、相关问题汇总。**

- **更新方式**

由作者构思一些小实验，预计每周更新一节，由于为业余时间更新，又要保证详细程度和质量，所以不会太快，但也不会停止，可以私聊沟通。同时，希望大家发散知识点，我会根据建议，更新相应知识点到该文档。提出知识点请到官方论坛（<http://bbs.100ask.net/>）发帖，**发帖统一**

版块: 100ASK_IMX6ULL; **发帖统一标题:** 百问网 imx6ul 开发板实战文档

发散 XXXX 实验; **发帖内容:** 描述希望得到的实验, 如: 请教如何一步一步实现三者互 ping。请相信我一定会尽快回复你。也希望, 可以由你来补充该文档, 有心思加入的同学请联系作者一起完成文档并增加章节作者署名。本文档将定期发布在论坛及百问网售后交流群。

● 百问网信息

淘宝店铺: <https://100ask.taobao.com/>

论坛: <http://bbs.100ask.org/forum.php>

公众号:



● 作者信息

CSDN: <https://xingxingzhihuo.blog.csdn.net/>

微信公众号:



有意发散实验思路及一起补充文档的同学除了发帖也可[公众号](#)或者 [CSDN](#)留言[与作者沟通](#)。

第0节 如何学习嵌入式 Linux 开发板

1、拿到开发板首先要做什么

除了最先欣赏一下**开发板的外观**，第一件事自然是上电看看**预装的固件效果**。最直观的就是上电后看屏幕的输出，正如我们新买了笔记本似的。

其一，下载开发板资料，其中一定有一个文档叫做**用户手册**。用户手册是帮助你搭建环境进行开发最便捷的文档，一定要先准备好，并浏览。

其二，就是看**调试串口的输出**，因为 Linux 跑起来，会通过串口打印出系统启动的过程，在串口里，你可以看到 uboot 启动，linux 被引导，文件系统被挂载，甚至是出错信息。所以搭建串口环境，并查看串口输出，自然要放在前面。

其三，欣赏完预装的固件，自然会会如何**烧写固件**，因为开发板是拿来学习的，学习中要对 uboot、内核、驱动做一些修改，自然避不开烧写。

所以你是初学者，拿到开发板后请首先完成以下：

- 下载资料，找到用户手册
- 搭建串口环境，观察系统启动
- 烧写固件，达到与预装同样的效果

2、嵌入式 linux 开发板我们要学习什么

我们搭建了开发板最基础的环境，那么我们重点学习什么呢，根据企业对嵌入式人才的需要，那么一定是要进行深度学习的，可以总结有如下学习的内容，大家可以按照如下思路，逐渐的深入学习

● 裸机学习

比如我要实现通过裸机点亮一个 LED，完成第一个裸机，也就点亮了后续裸机的明灯。然后进行其他裸机的一点点学习

● 驱动学习

裸机可能是基础，或者先跳过上面的裸机，先体验一下驱动，那第一个自然是 hello world 驱动，完成第一个驱动，你就知道了驱动如何写，包含哪些组成部分，发现驱动满满的套路，而并不难。可以深度学习的驱动有很多，慢慢开展即可。

● 实现项目

裸机和驱动就是为项目做准备的，也就是实现完整的功能，可以构想一个应用，在开发板上去实现。

● 深入 linux 内核

学了裸机，驱动，做了项目，这些可能更追求功能。而此时你想从事嵌入式 linux，建议开始深入分析 linux 内核子系统，比如 USB 子系统，网络子系统，内存管理等

等，这些对代码的要求高了一些，需要有一定代码阅读能力。深入学习后，代码阅读水平提升，对内核理解加深后，自身的能力开始达到更高水平。

初学者可以顺着本文档更新的章节，进行顺序学习或选择性的学习，这些是本人在开发板上一点点操作的记录，也算是一个学习的思路。

第 1 节 100ask_imx6ull 第一次上电环境搭建

收到开发板，第一件事自然是上电，看看预装的系统是不是能跑起来，把基础的开发环境搭起来，为后续开发学习做好准备。考虑有一些新接触嵌入式开发的同学，第一节描述的话多，有基础的可以扫过，没基础的可以顺着文字的思路进行实验。

1、实验目的

- 上电检查
- 安装串口驱动
- 熟悉当前的环境

2、实验过程

(1) 安装串口驱动

初学者首先要知道的一点就是，嵌入式设备一般会引出**调试串口**，在系统启动的 uboot、内核及挂载文件系统后的操作都可以在串口查看到并完成与用户的交互。因此串口环境必不可少。接上开发板串口线后，**右击我的电脑，打开设备管理器**。如下图一，显然是缺少串口驱动的。要指出的是针对不同开发板使用的串口芯片不同，所需要的驱动也不尽相同，同样的思路去解决。

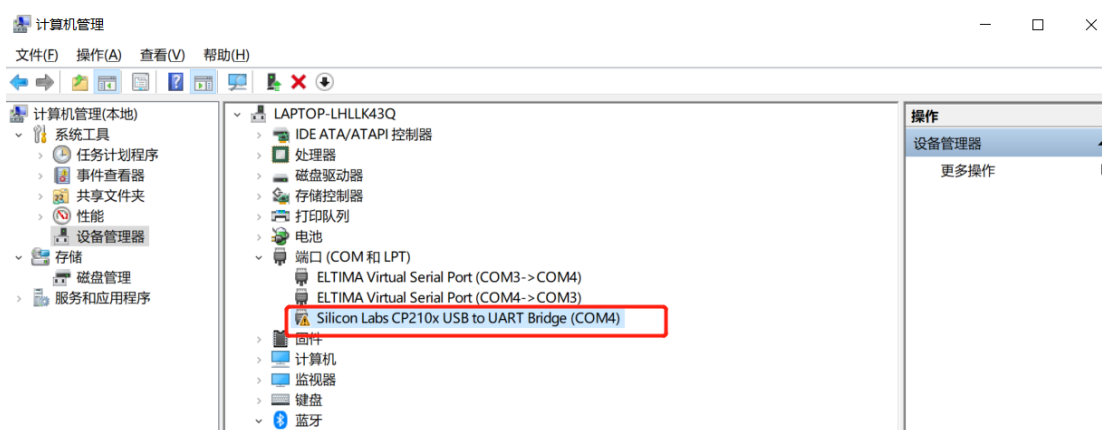


图 1 串口设备检测缺少驱动

解决如上缺少驱动的方法有很多种，比如安装驱动精灵软件，下载官方驱动等。一般遇到这个问题查看开发板用户手册一定告诉你如何解决了，比如《嵌入式 Linux 应用开发完全手册第二版_韦东山全系列视频文档全集 VXXX.pdf》。这里另一个思路就是直接去网上找到

这个芯片的驱动。直接搜索 CP210X，如下图 2，自然是圈住的更正规一点，像是串口芯片官网提供的驱动连接。（<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>）



图 2 上网寻找驱动

下载后得到 CP210x_Universal_Windows_Driver.zip，解压，然后设备管理器中，右击选择驱动目录，即可安装成功。如下如 3



图 3 更新驱动

安装成功后，即可看到开发板串口被识别出的串口号，此处 com5：

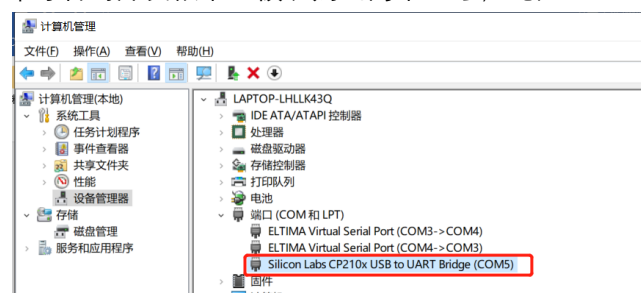


图 4 更新串口驱动后

(2) 查看串口输出

对于一个有过嵌入式开发的人员来说，电脑上一定安装有 SecureCRT，MobaXterm，xshell 这样的工具，打开，新建一个串口工程，选择串口 5，OK，创建成功。进入此操作的关键是一定要看到上图中的驱动安装成功，无感叹号。

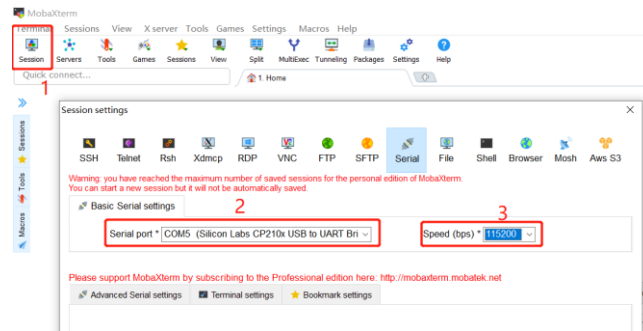


图 5 打开串口

上电，可以观察到串口输出如下，可以看到 uboot 的编译时间很新鲜。

```
U-Boot 2017.03 (Jan 12 2020 - 23:34:01 +0800)

CPU: Freescale i.MX6ULL rev1.1 528 MHz (running at 396 MHz)
CPU: Commercial temperature grade (0C to 95C) at 39C
Reset cause: POR
Model: Freescale i.MX6 ULL 14x14 EVK Board
Board: MX6ULL 14x14 EVK
DRAM: 512 MiB
MMC: FSL_SDHC: 0, FSL_SDHC: 1
*** Warning - bad CRC, using default environment

Display: TFT43AB (480x272)
Video: 480x272x24
In: serial
Out: serial
Err: serial
switch to partitions #0, OK
mmc1(part 0) is current device
Net: No ethernet found.
Normal Boot
Hit any key to stop autoboot: 0
=> pri
```

图 6 uboot 输出

如果什么都没有输出，很可能是你的启动方式不对，需要从 mmc 启动，才可以，通过右下角红色拨码开关来设置，在开发板后面有说明如何波动开发选择启动方式。

按下回车，可以查看到 uboot 的环境变量：

```
=> pri
baudrate=115200
board_name=EVK
board_rev=14X14
boot_fdt=try
bootcmd=run findfdt;run findtee;mmc dev ${mmcdev};mmc dev ${mmcdev}; if mmc rescan; then if run loadbootscript; then run bootscript; else if run loadimage; then run mmcboot; else run netboot; fi; fi; else run netboot; fi
bootcmd_mfg=run mfgtool_args; if test ${tee} = yes; then bootm ${tee_addr} ${initrd_addr} ${fdt_addr}; else bootz ${loadaddr} ${initrd_addr} ${fdt_addr}; fi;
bootdelay=3
bootdir=/boot
bootscript=echo Running bootscript from mmc ...; source
console=ttyMXC0
ethprime=eth1
fdt_addr=0x83000000
fdt_file=100ask_imx6ull-14x14.dtb
fdt_high=0xffffffff
fdtcontroladdr=9ef40478
findfdt=if test $fdt_file = undefined; then if test $board_name = EVK && test $board_rev = 9X9; then setenv fdt_file imx6ull-9x9-evk.dtb; fi; if test $board_name = EVK && test $board_rev = 14X14; then setenv fdt_file imx6ull-14x14-evk.dtb; fi; if test $fdt_file = undefined; then setenv fdt_file imx6ull-14x14-alpha.dtb; fi; fi;
image=zImage
initrd_addr=0x83000000
initrd_high=0xffffffff
ip_dyn=yes
loadaddr=0x80000000
loadbootscript=fatload mmc ${mmcdev}:${mmcpart} ${loadaddr} ${script};
loadfdt=ext2load mmc ${mmcdev}:${mmcpart} ${fdt_addr} ${bootdir}/${fdt_file}
loadimage=ext2load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${bootdir}/${image}
loadtee=fatload mmc ${mmcdev}:${mmcpart} ${tee_addr} ${tee_file}
mfgtool_args=setenv bootargs console=${console},${baudrate} rdinit=/linuxrc g_mass_storage.stall=0 g_mass_storage.removable=1 g_mass_storage.file=/fat g_mass_storage.ro=1 g_mass_storage.idVendor=0x066F g_mass_storage.idProduct=0x37FF g_mass_storage.iSerialNumber="" clk_ignore_unused
mmcargs=setenv bootargs console=${console},${baudrate} root=${mmcroot}
mmcautoload=yes
mmcboot=echo Booting from mmc ...; run mmcargs; if test ${tee} = yes; then run loadfdt; run loadtee; bootm ${tee_addr} - ${fdt_addr}; else if test ${boot_fdt} = yes || test ${boot_fdt} = try; then if run loadfdt; then bootz ${loadaddr} - ${fdt_addr}; else if test ${boot_fdt} = try; then bootz; else echo WARN: Cannot load the DT; fi; fi; else bootz; fi; fi;
mmcdev=1
```

图 7 uboot 环境变量

(3) 进行 shell 操作

等系统起来，输入 root，可以成功进行 shell 指令操作。


```

Welcome to imx6ull buildroot system !

* Documentation:  http://wiki.100ask.org/100ask_imx6ull
* SourceCode:    https://dev.tencent.com/u/weidongshan/
* Support:       https://support@100ask.net
* Shop:         https://100ask.taobao.com/
* Login:        name: root Passwd:
imx6ull login: root
[root@imx6ull:~]# pwd
/root
[root@imx6ull:~]#
[root@imx6ull:~]# cd
[root@imx6ull:~]# ls
[root@imx6ull:~]# cd /
[root@imx6ull:/]# ls
bin      dev      home     lib32    linuxrc  media    opt      root     samba    sys      usr
boot     etc      lib      libexec  lost+found mnt      proc     run      sbin     tmp      var
[root@imx6ull:/]#

```

图 8 shell 操作

(4) 网络测试

接上网线到路由器，可以看到如下网卡信息：

```

[root@imx6ull:/]# [ 204.085541] fec 2188000.ethernet eth1: Link is Up - 100Mbps/Full - flow control rx/tx
[ 204.093516] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
[root@imx6ull:/]#

```

如果你的电脑接到了路由器，然后你的开发板也是接到路由器，可以请求路由器分配 IP 地址给如上识别到的 eth1 网卡：

```

[root@imx6ull:/]# udhcpc -i eth1
udhcpc: started, v1.29.3
udhcpc: sending discover
udhcpc: sending select for 192.168.0.159
udhcpc: lease of 192.168.0.159 obtained, lease time 604800
deleting routers
adding dns 192.168.0.1
[root@imx6ull:/]#

```

然后实现与电脑互 ping。

```

[root@imx6ull:/]# ifconfig
eth0      Link encap:Ethernet  HWaddr E6:C9:91:CE:AC:71
          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)

eth1      Link encap:Ethernet  HWaddr 06:09:07:1C:9E:61
          inet addr:192.168.0.159  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::9400:87ff:fe1c:9e61/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:465 errors:0 dropped:0 overruns:0 frame:0
          TX packets:488 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:41456 (40.4 KiB)  TX bytes:39002 (38.0 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1024 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1024 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:70128 (68.4 KiB)  TX bytes:70128 (68.4 KiB)

[root@imx6ull:/]# ping 192.168.0.129
PING 192.168.0.129 (192.168.0.129): 56 data bytes
64 bytes from 192.168.0.129: seq=0 ttl=128 time=3.308 ms
64 bytes from 192.168.0.129: seq=1 ttl=128 time=2.940 ms
^C
--- 192.168.0.129 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.940/3.124/3.308 ms
[root@imx6ull:/]#

```

互 ping 很简单，但是出错的点也很多，后面会专门写一节如何一步一步搭建环境实现互 ping，从而可以进行网络操作。

(5) 查看屏幕

除了通过以上操作，我们可以知道开发板正常运转起来了，另一个更便捷的方法就是看屏幕。系统正常起来后，可以看到系统预装的应用界面。如下，可以尝试点击操作，实现下面的界面并不难，后面的章节，也会写一个简单的例程。



3、实验结果

- 成功安装串口驱动
- 可以查看 uboot 环境变量
- 进行简单的 shell 操作
- 测试网络
- 体验预装的应用

4、实验总结

- 学会如何解决遇到的驱动问题：找开发板用户手册，或者直接网上直接找驱动。
- 开始思考：系统是如何一步一步引导起来的？预装的系统带了哪些驱动？后面的章节将围绕知识点一点一点展开。同时配套的嵌入式教程将会详细的通过视频为大家开展。

开发学习中，把系统搞坏，是最常见的，如何重烧系统是第一个要熟悉并解决的任务，下一节将开始编译源码，并熟悉重烧整个系统的过程。

5、相关问题汇总

暂无

第2节 如何编译源码并重烧整个系统

重烧整个系统是正式开始学习第一步必须要掌握的，不然系统坏了、误删除了文件、学习驱动开发、系统发布等，都需要重烧系统。百问网提供好了移植好的所有固件，**第一步只需要实现编译并烧写即可，至于如何移植的，是后面深入学习要理解的内容，初学者不必在此纠结，但要记住自己的疑惑。**

1、实验目的

- 掌握嵌入式系统**编译**方法
- 掌握**烧写**系统方法
- 掌握通过 **busybox 制作文件系统**的方法
- 实现挂载**网络文件系统**
- 测试系统启动正常

2、实验过程

(1) 编译源码

实验到这里**一定要打开**《嵌入式 Linux 应用开发完全手册第二版_韦东山全系列视频文档全集 VXXX.pdf》第三章 构建系统。配套文档提供了详细的过程，按照操作即可。以下仅列出重点的几步。

● 获取源码

百问网提供了在线和离线两种方法，如果你的虚拟机可以访问外网，那一定选择，在线获取，无法上网的话那就离线吧，请参考嵌入式《嵌入式 Linux 应用开发完全手册第二版_韦东山全系列视频文档全集 VXXX.pdf》详细描述，不再赘述。最后得到了 100ask_imx6ull-sdk 目录，里面有需要的 uboot、内核、文件系统等源码。

● 准备工具链环境

安装工具链特别简单，简单说只要保证设置好的 **PATH 环境变量指向了工具链所在目录（只需把握好这一点就行）**。

如果你感觉工具链安装好了，但是 arm-linux-gnueabi-hf-gcc -v 没有得到版本信息，请按照如下思路检查

◆ 执行 echo \$PATH

查看 PATH 变量是否包含了 gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi-hf/bin，如果没包含，继续设置 PATH，直到包含该路径。

◆ 如果包含了，ls 查看

如本人的：ls /opt/100ask_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi-hf/bin/

◆ 看到如下，再执行 arm-linux-gnueabi-hf-gcc -v 才会成功看到版本号。

```
arm-linux-gnueabi-hf-addr2line  arm-linux-gnueabi-hf-g++      arm-linux-gnueabi-hf-gdb      arm-linux-gnueabi-hf-objdump
arm-linux-gnueabi-hf-ar         arm-linux-gnueabi-hf-gcc      arm-linux-gnueabi-hf-gfortran arm-linux-gnueabi-hf-ranlib
arm-linux-gnueabi-hf-as         arm-linux-gnueabi-hf-gcc-6.2.1 arm-linux-gnueabi-hf-gprof    arm-linux-gnueabi-hf-readelf
arm-linux-gnueabi-hf-c++        arm-linux-gnueabi-hf-gcc-ar   arm-linux-gnueabi-hf-ld       arm-linux-gnueabi-hf-size
arm-linux-gnueabi-hf-c++filt    arm-linux-gnueabi-hf-gcc-nm   arm-linux-gnueabi-hf-ld.bfd   arm-linux-gnueabi-hf-strings
arm-linux-gnueabi-hf-cpp        arm-linux-gnueabi-hf-gcc-ranlib arm-linux-gnueabi-hf-ld.gold   arm-linux-gnueabi-hf-strip
arm-linux-gnueabi-hf-dwp        arm-linux-gnueabi-hf-gcov     arm-linux-gnueabi-hf-nm       gdbserver
arm-linux-gnueabi-hf-elfedit    arm-linux-gnueabi-hf-gcov-tool arm-linux-gnueabi-hf-objcopy  runtest
```

- ◆ 所以说如果按照失败了，要么是 PATH 没设置好，或者 PATH 所指的目录里面根本没有工具链文件，那自然执行 arm-linux-gnueabi-gcc -v 会找不到命令。

- **编译 uboot**

```
cd Uboot-2017.03
make distclean
make mx6ull_14x14_evk_defconfig
make
生成 u-boot-dtb.imx
```

- **编译内核**

```
cd Linux-4.9.88
make mrproper
make 100ask_imx6ull_defconfig
make zImage
make dtbs
```

在 arch/arm/boot 目录下生成 zImage 内核文件，在 arch/arm/boot/dts 目录下生成设备树的二进制文件 100ask_imx6ull-14x14.dtb。

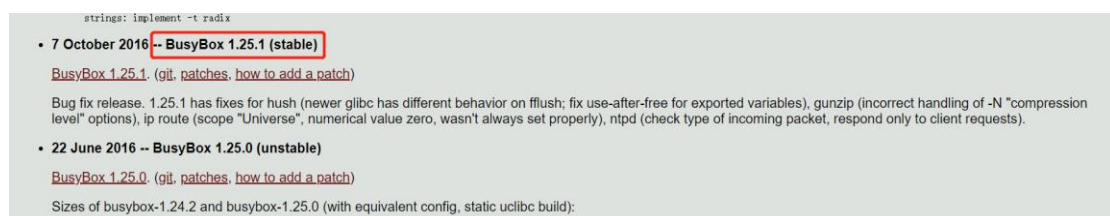
- **构建文件系统**

我们参考用户手册，实现了 uboot 和内核的编译，此处可以参考用户手册继续进行文件系统的构建，用户手册构建的文件系统过程相对复杂，其中包含的工具也比较多，大家最好尝试一下。此处通过 busybox 源码，按照之前 2440 的方法构建一个简单的文件系统，并通过 nfs 来挂载它。

10 分钟通过 busybox 制作文件系统方法:

- 下载 busybox 源码

根据之前安装的工具链版本 gcc version 6.2.1 20161016 这里建议选择与工具链时间相近的 busybox 版本。官网首页，可以看到 1.25.1 版本合适。就下载它了，<https://busybox.net/> 右侧选择 Download Source，然后下载。



下载速度太慢了，可以找到 github 上下载 (<https://github.com/mirror/busybox>)。

- 传到虚拟机并解压

```
unzip busybox-1_25_1.zip; cd busybox-1_25_1
```

- make menuconfig

```
Busybox Settings ->
  general configuration ->
    [*] don't use /usr          -----选中
  build options->
    cross compiler prefix = arm-linux-gnueabi-
```

```
make
make install
```

- 创建 etc 目录，并在该目录下创建 fstab、inittab、profile、rc.local、init.d/ rcS

这些文件里面内容应该是什么请参考博客：都是标准的。

(<https://xingxingzhihuo.blog.csdn.net/article/details/80110912>)

- 创建 dev 目录，并在该目录下执行

```
sudo mknod console c 5 1
sudo mknod null c 1 3
```
- 创建 lib 目录，并拷贝工具链中的库到该目录下。
根据你工具链的环境变量，找到 gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi 目录所在位置，然后进入
gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi/arm-linux-gnueabi/libc/lib 目录和 gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabi/lib 目录下执行 **cp *so* /lib -d**
(/lib 是你制作文件系统的 lib 目录)
- 创建其他空目录

```
mkdir proc mnt tmp sys root usr lib
```
- 制作好了，如果你还有不清楚的地方，可以参考上面提供的博客，参考一下里面制作的过程。再不清楚可以联系作者。

(2) 烧写

这里希望初学者思考一个问题，上面编译的内核源码和 uboot 是哪来的？是开发板厂家自己从 kernel.org 上下载 linux 源码，一点一点移植吗？答案自然不是，芯片原厂，都提供了支持自家芯片的 sdk，里面做好了支持自家芯片的工作。对于 imx6ul，**NPX 官方**提供的资料连接地址如下，在这里可以找到提供的基础 SDK 及文档，以及原厂提供的烧录工具。

<https://www.nxp.com.cn/products/processors-and-microcontrollers/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx-6ultralite-processor-low-power-secure-arm-cortex-a7-core:i.MX6UL>。

怎么烧写呢？建议使用 mfgtools 烧写系统，参考百问网用户手册操作即可。总结如下：

- 按照开发板后的 boot 说明，修改拨码开关为 USB 启动
- 下载并解压 100ask_imx6ull-mfgtools.zip
- 将自己编译得到的 u-boot-dtb.imx、zImage、100ask_imx6ull-14x14.dtb 等文件替换掉 100ask_imx6ull-mfgtools\Profiles\Linux\OS Firmware\files 下的文件。
- 执行 buildroot-image-100ask_100ask-ddr512m-emmc4g.vbs
- 点击 Start，等待完成
- 上电，观察 uboot 的编译时间，确认系统已更新。

(3) 设置挂载网络文件系统

Uboot 下执行 pri，查看环境变量，发货版本文件系统是在 mmc 上的。有如下环境变量：

```
mmccargs=setenv bootargs console=${console},${baudrate} root=${mmccroot}
```

我们将启动的 root= 替换为网络文件系统，设置如下：

- 修改环境变量 mmccargs，里面的 ip 值和文件系统路径按照自己的修改。

```
setenv mmccargs 'setenv bootargs init=/linuxrc console=${console},${baudrate}
root=/dev/nfs ip=dhcp nfsroot=192.168.0.182:/home/xingxingzhihuo/src/busybox-
1_25_1/_install,v3,tcp'
```

如果需要修改为出厂时默认的挂载 mmc 的文件系统，则重新执行：

```
setenv mmccargs 'setenv bootargs console=${console},${baudrate} root=${mmccroot}'
```

- saveenv
- 将开发板接到路由器上，并确保可以 ping 通 ubuntu。具体如何实现三者互 ping，如果你还不会，该文档很快会增加三者互 ping 环境设置教程。
- 重启，挂载网络文件系统成功。

```
[ 6.165441] fec 20b4000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx
[ 6.194549] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 6.224764] Sending DHCP requests .. OK
[ 6.514592] IP-Config: Got DHCP answer from 192.168.0.1, my address is 192.168.0.154
[ 6.522421] IP-Config: Complete:
[ 6.525843] device=eth0, hwaddr=7a:f8:9a:02:be:0c, ipaddr=192.168.0.154, mask=255.255.255.0, gw=192.168.0.1
[ 6.536172] host=192.168.0.154, domain=, nis-domain=(none)
[ 6.542160] bootserver=0.0.0.0, rootserver=192.168.0.182, rootpath= nameserver0=192.168.0.1
[ 6.559273] can-3v3: disabling
[ 6.562425] ltemodule-pwr: disabling
[ 6.566429] wifi-pwr: disabling
[ 6.569645] ALSA device list:
[ 6.572668] #0: wm8960-audio
[ 6.667543] VFS: Mounted root (nfs filesystem) readonly on device 0:14.
[ 6.677972] devtmpfs: mounted
[ 6.684002] Freeing unused kernel memory: 1024K

Please press Enter to activate this console.
/ # ls
bin      dev      init     linuxrc  proc    /sbin    tmp
busybox  etc      lib      mnt      root     sys     usr
/ # busybox
BusyBox v1.25.1 (2020-01-19 07:25:01 PST) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.
```

- 启动时间较预装的文件系统快了很多哈，但是由于还缺少一些库的移植，板载硬件的部分功能将会出现问题。

3、实验结果

- 查看 uboot 打印的第一行，确认 uboot 更新为最新编译时间
- 更新了开发板 uboot 和内核及设备树
- 制作文件系统，并挂载网络文件系统

4、实验总结

预装的文件系统较为复杂，是不是感觉有点花哨，自己动手制作了文件系统，并通过网络挂载，可以方便自己后续的学习和开发。

需要说明的是，预装的文件系统虽然复杂，但是工具多，对于开发项目来说更为便捷，且预装的文件系统支持了板载的一些硬件，而自己制作的文件系统没有增加一些库，蓝牙，无线功能应该不能用。

作者的建议是，初学者可以使用自己的制作的文件系统，达到一定水平后，研究一下预装的文件系统更好。

5、相关问题汇总

(1) arm-linux-gnueabi-hf-gcc -v 错误

[见 2、实验过程 如上准备工具链环境](#)

第3节 PC机-开发板-ubuntu 三者互 ping 实验

Uboot 中使用网络命令，linux 中使用网络进行挂载，进行网络开发等，都需要实现开发板和主机的网络通信。尤其实现**开发板-宿主机 Windows PC-虚拟机中的 Ubuntu**三者之间的网络互通环境，对开发便捷性有很大帮助。

超级详细的教程：<http://wiki.100ask.org/VMwareAndUbuntuNetworkSetupGuide>

1、实验目的

实现**开发板-宿主机 Windows PC-虚拟机中的 Ubuntu**三者网络互通，具体通过 ping 命令来进行网络连通性测试。

2、实验过程

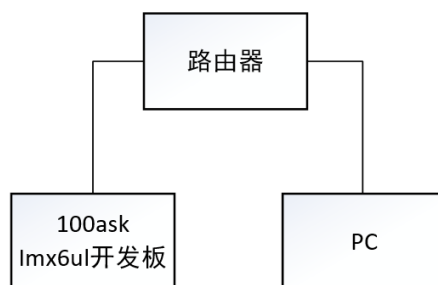
实现三者网络互通，并不难，但需要一些网络知识，都是网络环境问题。所以如过你遇到了网络互通问题，主要思路就是检查自己的**网络环境**，对于没有网络基础的同学，可以参考如下，一步一步实验：

(1) 实验环境准备

- 百问网 100ask imx6ul 开发板
- 路由器
- PC, windows 系统
- PC 中安装虚拟机及 ubuntu 系统

(2) 网络连接拓扑

曾有同学 ping 不通，寻求帮助，经过一番沟通，原因在于对方还不清楚需要连接网线，压根没接网线，有些同学还有些是将开发板直连 PC，然后 ubuntu 桥接到无线网卡，无法实现三者互 ping，这些都是不行的。首先讲解最常规的网络环境：



如图，需要有一个**路由器**，然后将开发板的网口和 PC 机的网口都通过网线连接到路由器的口上。

(3) PC 机环境设置

- 关闭 windows 防火墙，不然外部 ping PC 的时候不通。

控制面板 > 系统和安全 > Windows Defender 防火墙 > 自定义设置

自定义各类网络的设置

你可以修改使用的每种类型的网络的防火墙设置。

专用网络设置



☐ 启用 Windows Defender 防火墙

☐ 阻止所有传入连接，包括位于允许应用列表中的应用

☐ Windows Defender 防火墙阻止新应用时通知我



☒ 关闭 Windows Defender 防火墙(不推荐)

公用网络设置



☐ 启用 Windows Defender 防火墙

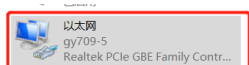
☐ 阻止所有传入连接，包括位于允许应用列表中的应用

☒ Windows Defender 防火墙阻止新应用时通知我



☒ 关闭 Windows Defender 防火墙(不推荐)

● 检查路由器是否能为 PC 分配到 IP



1, 找到本地有线网卡
右击选择状态



2 点击详细信息



收发包计数，可以知道是否有数据包发到本地网卡
以及本地网卡是否向外发送数据包

如上图，可以看到 PC 机通过 DHCP 从路由器处获得的 IP 为 192.168.0.129，此时可以在 cmd 终端 ping 路由器 192.168.0.1。

```
C:\Users\yangf>ping 192.168.0.1

正在 Ping 192.168.0.1 具有 32 字节的数据:
来自 192.168.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.0.1 的回复: 字节=32 时间<1ms TTL=128

192.168.0.1 的 Ping 统计信息:
    数据包: 已发送 = 2, 已接收 = 2, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms
```

该步骤验证通过，可以说明路由器 DHCP 是没有问题，正常也应该可以为开发板分配 IP。

(4) 开发板环境设置

● Uboot 阶段使用网络

在 uboot 中 ping 路由器。首先需要设置自己的 ip，说明一下该 ip 是在 uboot 阶段的 ip，并非内核起来后的 IP，这俩 ip 并没有关系。

执行如下命令设置：setenv ipaddr 192.168.0.111（注意这个 IP 设置一个和 PC 同网段的，也是和路由器同网段的）


```
initrd_high=0xffffffff
ip_dyn=yes
ipaddr=192.168.0.111
loadaddr=0x80800000
```

Ping PC 机和路由器测试：

```
=> ping 192.168.0.129
Using ethernet@020b4000 device
host 192.168.0.129 is alive
=> ping 192.168.0.1
Using ethernet@020b4000 device
host 192.168.0.1 is alive
=> █
```

PC机IP
路由器

● Linux 系统起来后

如果你使用的预装文件系统，且启动前接好了网线，那么起来后 ifconfig 可以看到网卡已经获取到路由器分配的 IP 了。

```
[root@imx6ull:~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:FC:90:B3:BE:70
          inet addr:192.168.0.152  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::50fc:90ff:feb3:be70/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:84 errors:0 dropped:0 overruns:0 frame:0
          TX packets:96 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8448 (8.2 KiB)  TX bytes:8350 (8.1 KiB)
```

也可以执行 udhcpc -i eth0 请求路由器重新分配 IP。

```
[root@imx6ull:~]# udhcpc -i eth0
udhcpc: started, v1.29.3
udhcpc: sending discover
udhcpc: sending select for 192.168.0.152
udhcpc: lease of 192.168.0.152 obtained, lease time 604800
deleting routers
adding dns 192.168.0.1
[root@imx6ull:~]#
```

此时 ping PC 和路由器测试,可以看到网络是通的。

```
[root@imx6ull:~]# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: seq=0 ttl=64 time=1.897 ms
^C
--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.897/1.897/1.897 ms
[root@imx6ull:~]# ping 192.168.0.129
PING 192.168.0.129 (192.168.0.129): 56 data bytes
64 bytes from 192.168.0.129: seq=0 ttl=128 time=5.505 ms
64 bytes from 192.168.0.129: seq=1 ttl=128 time=3.262 ms
^C
--- 192.168.0.129 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 3.262/4.383/5.505 ms
[root@imx6ull:~]#
```

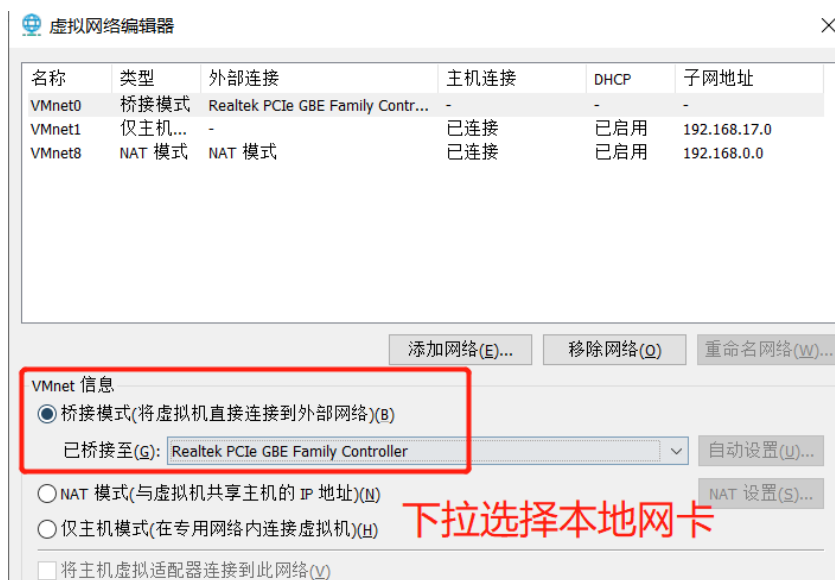
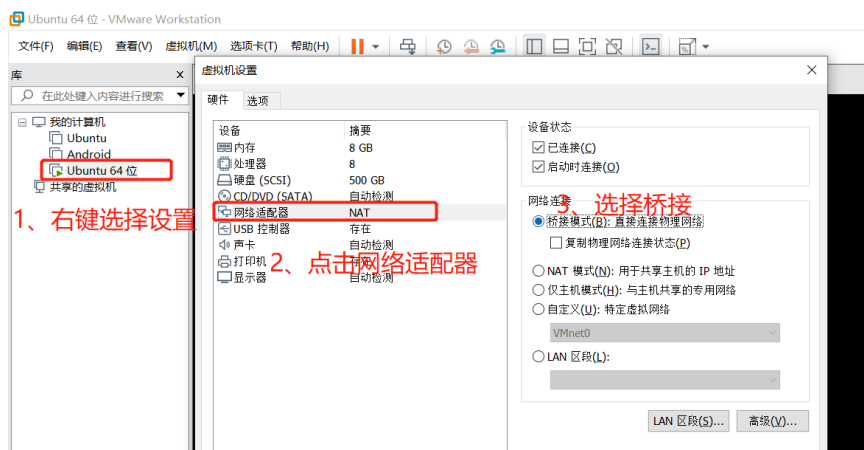
(5) ubuntu 设置

超级详细的教程:<http://wiki.100ask.org/VMwareAndUbuntuNetworkSetupGuide>

建议打开上面的 WIKI，根据自己的环境对照设置。

相对来说 PC 机和开发板的网络环境比较简单，基本上配置好 IP 在同一网段后就可以直接 ping 通了。Ubuntu 在虚拟机中启动，这次需要借助虚拟机来配置 ubuntu 的环境。

- 虚拟机设置桥接本地网卡为桥接模式



这里一定注意是桥接到本地网卡，不要选择无线网卡，不然无法实现三者互ping。因为数据包到达的是本地网卡，由本地网卡给到虚拟机，如果桥接到了无线

网卡，开发板的网络包无法到达虚拟机。

- 重启，查看 ubuntu IP

在 ubuntu 中 ping PC 和开发板。

```
xingxingzhihuo@ubuntu:~$ ping 192.168.0.152
PING 192.168.0.152 (192.168.0.152) 56(84) bytes of data.
64 bytes from 192.168.0.152: icmp_seq=1 ttl=64 time=6.88 ms
64 bytes from 192.168.0.152: icmp_seq=2 ttl=64 time=3.30 ms
^C
--- 192.168.0.152 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 3.309/5.097/6.885/1.788 ms
xingxingzhihuo@ubuntu:~$ ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.48 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.35 ms
^C
--- 192.168.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.357/1.420/1.484/0.073 ms
xingxingzhihuo@ubuntu:~$ ping 192.168.0.129
PING 192.168.0.129 (192.168.0.129) 56(84) bytes of data.
64 bytes from 192.168.0.129: icmp_seq=1 ttl=128 time=0.401 ms
64 bytes from 192.168.0.129: icmp_seq=2 ttl=128 time=0.234 ms
^C
--- 192.168.0.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1028ms
rtt min/avg/max/mdev = 0.234/0.317/0.401/0.085 ms
xingxingzhihuo@ubuntu:~$
```

开发板

路由器

PC

(6) 无法实现互 ping 的定位方法

- PC 相关 ping 操作问题定位

如果 ping PC 不通，或者 PC ping 外接不通，可以找到网卡右键选择状态，打开如下图，然后执行 ping 操作的时候，盯着如下红框的收发包计数，确定是 ping 没有发出去，还是 ping 回包没收到。如果能 ping 通，收发包计数在 ping 期间包的字节数会不断增长。



- 安装抓包工具 Wireshark

安装 **Wireshark** 抓包工具，抓取本地网卡的网络数据包，查看是否收到 ping 包，或者回包。如外部 ping PC，如下图，192.168.0.152 是 ubuntu，192.168.0.129 是本地网卡 ip。本地网卡收到了来自 152 的请求包，并回复了 ICMP 包。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	WistronI_e8:1b:11	D-LinkIn_a1:86:a8	ARP	42	Who has 192.168.0.1? Tell 192.168.0.129
2	0.000010	WistronI_e8:1b:11	D-LinkIn_a1:86:a8	ARP	42	Who has 192.168.0.1? Tell 192.168.0.129
3	0.000731	D-LinkIn_a1:86:a8	WistronI_e8:1b:11	ARP	60	192.168.0.1 is at 10:62:eb:a1:86:a8
4	5.400279	192.168.0.152	192.168.0.129	ICMP	98	Echo (ping) request id=0x4e01, seq=0/0, ttl=64 (reply in 5)
5	5.400443	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=0/0, ttl=128 (request in 4)
6	5.400451	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=0/0, ttl=128
7	6.401184	192.168.0.152	192.168.0.129	ICMP	98	Echo (ping) request id=0x4e01, seq=1/256, ttl=64 (reply in 8)
8	6.401254	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=1/256, ttl=128 (request in 7)
9	6.401260	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=1/256, ttl=128
10	7.401472	192.168.0.152	192.168.0.129	ICMP	98	Echo (ping) request id=0x4e01, seq=2/512, ttl=64 (reply in 11)
11	7.401681	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=2/512, ttl=128 (request in 10)
12	7.401706	192.168.0.129	192.168.0.152	ICMP	98	Echo (ping) reply id=0x4e01, seq=2/512, ttl=128

Wireshark 是很好用的抓包工具，如果和 PC 相关的网络问题，可以通过抓包工具来定位相关网络包是否接收和发送成功。

- 开发板中安装抓包工具

一般来说嵌入式系统常用的网络抓包工具是 tcpdump，交叉编译即可在开发板上进行抓包。

命令: tcpdump -i eth0.

执行后，所有 eth0 进出的网络数据包都会被抓取并列出来。

通过抓包工具，我们可以定位到是否收到相关数据包，从而定位到出问题的点。

(7) 补充实验

- 开发板直连电脑

如果你没有路由器，直接将开发板和 PC 机接到一起，这个时候外接就无法分配 IP 了，PC 机和开发板的 IP 都需要手动设置，只需要设置到同一个网段即可。如果设置好没 ping 通，那就抓包，看数据包是否收到。

- PC 需要通过无线上网

如果路由器无法上网，按照上面的桥接方式，ubuntu 是无法上网的。而 PC 机仅能通过无线上网的话，需要桥接到无线网卡上。上网后再桥接到本地网卡。

3、实验结果

实现三者互 ping。

4、实验总结

- 实现三者互 ping
- 可以使用抓包工具抓取 ping 过程中的 ICMP 包
- 可以通过抓包来分析故障点

5、相关问题汇总

暂无

第 4 节 编译并加载第一个 Hello World 驱动

1、 实验目的

对于初学者，本节重点掌握，如何编译驱动，加载驱动，完成一个最简单的驱动，并通过测试用例测试驱动。实现从应用到驱动的一个简单而又重要的实验。重点：

- 熟悉**编译**驱动
- 如何**加载**驱动
- 驱动简单的调试技巧

2、 实验过程

(1) 编写驱动

这里就不一点点写驱动了，使用老师写好的来实验，可以到 github 上下载。
(https://github.com/100askTeam/01_all_series_quickstart) 路径如下：

01_all_series_quickstart/04_快速入门(正式开始)/02_嵌入式 Linux 驱动开发基础知识
/source/ 01_hello_drv

驱动模板很重要，驱动都是有套路的，按照驱动的模板把需要内容填进去就是一个简单的驱动，详细请学习老师的视频教程。

下面说一下**一个简单的驱动需要包含哪些内容**：

- 头文件

驱动中可能用到一些在内核中提供的 API 函数，所以要把驱动中需要使用的 API 头文件包含进来。比如实现一个网卡驱动，可以参考内核网卡驱动包含哪写头文件，引用过来即可

- 入/出口函数

这两个函数是在驱动加载和卸载的时候会调用到的，这是驱动需要包含的必要元素。在驱动加载的时候会调用 module_init 修饰的函数，驱动卸载的时候会调用 module_exit 修饰的函数，可以看出，这俩函数主要用于资源的初始化和资源释放。

```
module_init(hello_init);
```

```
module_exit(hello_exit);
```

- file_operations 及相关函数

应用程序主要通过一些 API 接口访问到驱动，比如 open，read，close 等。于是驱动需要实现这些函数，当应用调用 open 的时候，驱动中的 open 就会被调用。

- 驱动模块声明

```
MODULE_LICENSE("GPL");
```

(2) 编译驱动

初学者要知晓一点，**编译驱动是依赖内核环境**，也就是说内核需要先编译成功，然后在驱动的 Makefile 里面的内核路径指到自己的内核路径上。具体要好好分析一下 Makefile。

编译内核：

见本文档实验《如何编译源码并重烧整个系统》章节

内核编译成功后，pwd 查看内核路径，然后将该路径更新到驱动的 Makefile 中。

执行 **make** 完成驱动编译产生 ko 文件，同时测试用例也编译好了。

```
xingxingzhihuo@ubuntu:~/src/imx6u_driver/01_hello_drv$ make
make -C /home/xingxingzhihuo/src/100ask_imx6ull-sdk/Linux-4.9.88 M=`pwd` modules
make[1]: Entering directory '/home/xingxingzhihuo/src/100ask_imx6ull-sdk/Linux-4.9.88'
CC [M] /home/xingxingzhihuo/src/imx6u_driver/01_hello_drv/hello_drv.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/xingxingzhihuo/src/imx6u_driver/01_hello_drv/hello_drv.mod.o
LD [M] /home/xingxingzhihuo/src/imx6u_driver/01_hello_drv/hello_drv.ko
make[1]: Leaving directory '/home/xingxingzhihuo/src/100ask_imx6ull-sdk/Linux-4.9.88'
arm-linux-gnueabihf-gcc -o hello_drv_test hello_drv_test.c
xingxingzhihuo@ubuntu:~/src/imx6u_driver/01_hello_drv$ ls
hello_drv.c  hello_drv.mod.c  hello_drv.o  hello_drv_test.c  modules.order
hello_drv.ko  hello_drv.mod.o  hello_drv_test  Makefile  Module.symvers
xingxingzhihuo@ubuntu:~/src/imx6u_driver/01_hello_drv$
```

(3) 加载驱动

insmod *.ko （下面的警告是新版本内核对驱动的校验检测，无任何影响，可以忽略）

```
[root@imx6ull:~]# insmod hello_drv.ko
[ 36.622254] hello_drv: loading out-of-tree module taints kernel.
[root@imx6ull:~]#
[root@imx6ull:~]#
```

lsmod 查看驱动加载成功

```
[root@imx6ull:~]# lsmod
Module                  Size  Used by    Tainted: G
hello_drv               3542  0
[root@imx6ull:~]#
[root@imx6ull:~]#
[root@imx6ull:~]# ls /dev/hello
/dev/hello
[root@imx6ull:~]#
```

(4) 驱动加载失败解决

前面说，驱动编译依赖内核，那么**驱动加载也依赖内核**，如果当前运行的内核和编译驱动的内核不一致的时候会出现如下错误，一些 api 无法找到。更新一下开发板的内核为编译驱动时指定的内核即可解决。

```
[root@imx6ull:/mnt]# insmod hello_drv.ko
[ 107.288615] hello_drv: loading out-of-tree module taints kernel.
[ 107.295317] hello_drv: disagrees about version of symbol device_create
[ 107.301916] hello_drv: Unknown symbol device_create (err -22)
[ 107.310018] hello_drv: disagrees about version of symbol device_destroy
[ 107.317857] hello_drv: Unknown symbol device_destroy (err -22)
[ 107.329543] hello_drv: disagrees about version of symbol device_create
[ 107.336536] hello_drv: Unknown symbol device_create (err -22)
[ 107.342416] hello_drv: disagrees about version of symbol device_destroy
[ 107.351387] hello_drv: Unknown symbol device_destroy (err -22)
insmod: can't insert 'hello_drv.ko': Invalid argument
```

解决：

参考完全手册，更新内核

执行如上操作后，重启开发板。并检查是否更新成功

■ 开发板执行 cat /proc/version

```
[root@imx6ull:~]# cat /proc/version
Linux version 4.9.88-g13a4717-dirty (xingxingzhihuo@ubuntu) (gcc version 6.2.1 20161016 (Linaro GCC 6.2-2016.11) ) #2 SMP PREEMPT Fri Jan 31 06:
06:31 PST 2020
[root@imx6ull:~]#
```

■ Ubuntu 虚拟机执行：date

对比内核编译的时间和当前运行内核的时间是不是一致，确保运行的内核更新最新的成功。

(5) 驱动调试

老师在第一个简单的驱动里面，通过如下方式进行函数打印。该句话很实用，可以很方便的定位到驱动执行到了哪里，很清晰的打印了执行的函数和位于哪一行。以后大家在驱动

开发中, 希望定位函数执行过程的时候, 可以实用该句增加到驱动中, 是定位很常用的方法。

```
printf("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
```

3、 实验结果

执行测试用例效果如下:

```
[root@imx6ull:~]# ./hello_drv_test -w "hello world"
[root@imx6ull:~]# ./hello_drv_test -r
APP read : hello world
[root@imx6ull:~]# █
```

该驱动和测试用例都比较简单, 但是包含了一个简单驱动的所有元素。初学者一定要把这个驱动理解好, 然后复杂的驱动只不过在此基础上实现更多逻辑而已。

4、 实验总结

以上是该驱动基础实验的内容, 但还有一些细节的知识点需要初学者好好吸收, 往往第一个驱动也是进步比较大的, 该驱动初学者需要掌握:

- **编译驱动**, 编译驱动的 makefile 是什么样的, 知道编译驱动依赖于内核
- **加载驱动**, 解决加载驱动可能出现的问题
- **测试驱动**, 执行驱动测试用例, 由应用层掉到驱动层的函数
- 掌握一个简单驱动的**最小组成**需要包含哪些内容

5、 相关问题汇总

(1) Unknow symbol -22 错误

[见如上 2、实验过程 \(4\) 驱动加载失败解决](#)

第 5 节 通过 uboot 使用网络启动系统

1、 实验目的

Uboot 中使用网络来加载内核、设备树等系统文件, 然后启动内核后, 再挂载网络文件系统。

便于开发, 快速替换内核及文件移动。

2、实验过程

预装的系统使用烧写在 mmc 中的固件来启动，使用网络来启动，可以方便更新内核和驱动等，在开发阶段十分有力。

如何通过网络来加载固件呢？首先固件放在虚拟机 ubuntu 的某个目录，然后 uboot 中使用网络（如 tftp）**将固件加载到内存**，然后启动；内核起来后再去**挂载网络文件系统**，所以这一节**网络**环境尤为重要。

分解任务：

- **Uboot 通过 tftp 获取 ubuntu 中的文件**

要想实现这个首先，首先要按照第 3 节的章节，先确保三者互 ping 没问题，然后 ubuntu 上要启动一个 tftp 服务器，然后 uboot 中使用 tftp 命令加载文件。

- Uboot 通过网络启动内核

Uboot 环境变量已经支持好了，run netboot 即可。

- 网络文件系统制作

既然内核起来后希望挂载网络文件系统，那么 ubuntu 上要准备好网络文件系统。

任务一、在 uboot 中通过 tftp 加载文件到内存

① 在 ubuntu 中安装 tftp 服务

Uboot 中希望通过网络加载文件，那一定要选择一个网络协议来传出文件，是 TCP/IP 网络模型里面应用层的事。这里我们选择 tftp 协议。传输文件需要一个服务端一个客户端，首先实现在 ubuntu 中安装 tftp 服务。

- Ubuntu 中安装 tftp 服务

执行安装：sudo apt-get install tftp-hpa tftpd-hpa

安装好了，查看是否运行起来了呢？

查询服务：sudo service tftpd-hpa status

```
xingxingzhihuo@ubuntu:~$ sudo service tftpd-hpa status
● tftpd-hpa.service - LSB: HPA's tftp server
   loaded: loaded (/etc/init.d/tftpd-hpa; bad; vendor preset: enabled)
   Active: active (running) since Mon 2020-02-17 18:36:59 PST; 10h ago
     Docs: man:systemd-sysv-generator(8)
   Process: 1384 ExecStart=/etc/init.d/tftpd-hpa start (code=exited, status=0/SUCCESS)
    CGroup: /system.slice/tftpd-hpa.service
            └─1401 /usr/sbin/in.tftpd --listen --user tftp --address :69 --secure /home/xingxingzhihuo/tftpboot

Feb 17 18:36:59 ubuntu systemd[1]: Starting LSB: HPA's tftp server...
Feb 17 18:36:59 ubuntu tftpd-hpa[1384]: * Starting HPA's tftpd in.tftpd
Feb 17 18:36:59 ubuntu tftpd-hpa[1384]:   ...done.
Feb 17 18:36:59 ubuntu systemd[1]: Started LSB: HPA's tftp server.
xingxingzhihuo@ubuntu:~$
```

可以看到服务运行中，且 tftp 共享目录在下面可以看到，这里的目录是我修改后的，当然你也可以使用安装后默认的共享目录，如果更改这个目录为你自己的路径，可以修改服务的配置文件，sudo vi /etc/default/tftpd-hpa，增加以下内容。

TFTP_DIRECTORY="/home/book/tftpboot"

TFTP_OPTIONS="-l -c -s"

如果使用 tftp 默认的路径，那么增加在配置文件中增加这么一条即可：

TFTP_OPTIONS="-l -c -s"

- 重启服务，再次检查服务状态为 running，且目录里面包含你希望获取的文件。

② Uboot 中使用 tftp 加载文件到内存。

- 网络检查

进行网络操作前一定要进行网络检查，确保网络没问题，不然一切都是 0。

Ping 测试，ping 路由，ping ubuntu 都是通的，这是一切的基础。

```

=> ping 192.168.0.1
Using ethernet@020b4000 device
host 192.168.0.1 is alive
=> ping 192.168.0.131
Using ethernet@020b4000 device
host 192.168.0.131 is alive
=> █

```

● Tftp 检查

执行 tftp 加载文件比如我的 ubuntu 上 tftp 服务路径，包含一个 core.c 文件，那么通过 tftp 加载到内存中。当然可以把内核放到 tftp 服务路径，这里直接加载内核到内存。

```

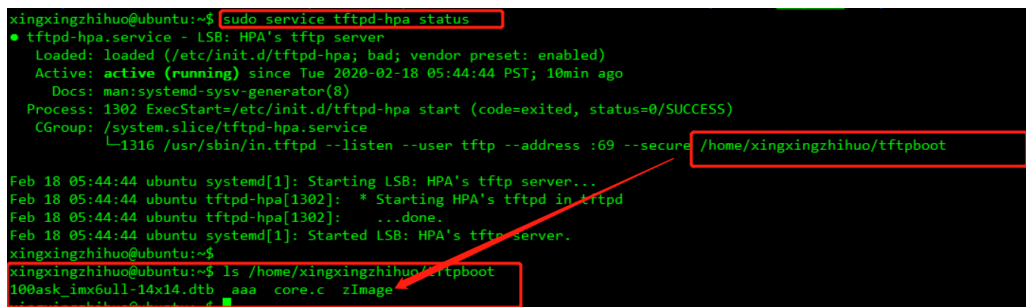
=> tftp core.c
Using ethernet@020b4000 device
TFTP from server 192.168.0.131; our IP address is 192.168.0.111
Filename 'core.c'.
Load address: 0x80800000
Loading: #####
                296.9 KiB/s
done
Bytes transferred = 50868 (c6b4 hex)
=> █

```

任务二、通过网络启动内核

这里的目的是在 uboot 中加载内核到内存中，然后启动内存中的内核。而不是之前的从存储设备上加载内核。如何做到，uboot 环境变量已经设置好了，只需要在 uboot 中执行 run netboot 即可。

① 准备内核及设备树到 tftp 路径。



```

xingxingzhihuo@ubuntu:~$ sudo service tftpd-hpa status
● tftpd-hpa.service - LSB: HPA's tftp server
   Loaded: loaded (/etc/init.d/tftpd-hpa; bad; vendor preset: enabled)
   Active: active (running) since Tue 2020-02-18 05:44:44 PST; 10min ago
     Docs: man:systemd-sysv-generator(8)
   Process: 1302 ExecStart=/etc/init.d/tftpd-hpa start (code=exited, status=0/SUCCESS)
    CGroup: /system.slice/tftpd-hpa.service
            └─1316 /usr/sbin/in.tftpd --listen --user tftp --address :69 --secure /home/xingxingzhihuo/tftpboot

Feb 18 05:44:44 ubuntu systemd[1]: Starting LSB: HPA's tftp server...
Feb 18 05:44:44 ubuntu tftpd-hpa[1302]: * Starting HPA's tftp server in tftpd
Feb 18 05:44:44 ubuntu tftpd-hpa[1302]: ..done.
Feb 18 05:44:44 ubuntu systemd[1]: Started LSB: HPA's tftp server.
xingxingzhihuo@ubuntu:~$
xingxingzhihuo@ubuntu:~$ ls /home/xingxingzhihuo/tftpboot
100ask_imx6ull-14x14.dtb  aaa  core.c  zImage

```

② run netboot

uboot 中这么一执行，会把内核加载到内存，并启动内核。为什么呢？分析 uboot 环境变量一目了然。看看 netboot 在环境变量是什么内容。

```

netboot=echo Booting from net ...; run netargs; setenv get_cmd tftp; ${get_cmd} ${image}; ${get_cmd} ${fdt_addr} ${fdt_file}; bootz ${loadaddr}
- ${fdt_addr};
nfsroot=/home/xingxingzhihuo/src/busybox-1.25.1/install

```

这么一框就很清楚了，run netboot，就会运行这些命令，一方面去设置其他环境变量，一方面就是加载内核，然后就是引导内核。没必要一点一点分析了，找规律也能把整个环境变量理清楚。

[illegible]

设备树被加载到内存

内核继续执行，会挂接网络文件系统，这个时候我们还没准备网络文件系统环境，系统会卡住，如下：

任务三、挂载网络文件系统

参考本文档第二节,《10 分钟通过 busybox 制作文件系统方法》,或者参考《嵌入式 Linux 应用开发完全手册第二版 韦东山全系列视频文档全集 VXXX.pdf》

内核启动最后阶段，会去挂载网络文件系统，网络文件系统的目录可以通过修改 uboot 环境变量来指定。

```
setenv nfsroot /home/xingxingzhihuo/src/baiwen/busybox/_install
```

如果要解释为啥设置 `nfsroot` 环境变量就行了，那么看下图。所以 `run netboot` 一定要分析一下它做了什么。

- 检查 ubuntu 上网络文件系统是否准备好

如何检查：ubuntu 执行 showmount -e

```
xingxingzhihuo@ubuntu:~/src/baiwen/busybox/_install$ showmount -e
Export list for ubuntu:
/home/xingxingzhihuo/src/baiwen/_install *
/home/xingxingzhihuo/src/baiwen/busybox/_install *
xingxingzhihuo@ubuntu:~/src/baiwen/busybox/_install$
```

可以看到刚才设置的 nfsroot 路径在这里，那么就 OK。

- 再次 run netboot 启动

大功告成！

```
[ 6.504538] IP-Config: Got DHCP answer from 192.168.0.1, my address is 192.168.0.109
[ 6.512369] IP-Config: Complete:
[ 6.515793]     device=eth0, hwaddr=a6:1b:82:65:b0:98, ipaddr=192.168.0.109, mask=255.255.255.0, gw=192.168.0.1
[ 6.526129]     host=192.168.0.109, domain=, nis-domain=(none)
[ 6.532117]     bootserver=0.0.0.0, rootserver=192.168.0.131, rootpath=     nameserver=192.168.0.1
[ 6.549136] can-3v3: disabling
[ 6.552291] ltemodule-pwr: disabling
[ 6.556297] wifi-pwr: disabling
[ 6.559514] ALSA device list:
[ 6.562542]   #0: vm8960-audio
[ 6.634787] VFS: Mounted root (nfs filesystem) readonly on device 0:14.
[ 6.643962] devtmpfs: mounted
[ 6.649430] Freeing unused kernel memory: 1024K

Please press Enter to activate this console.
/ #
/ # ls
bin      etc      linuxrc  proc     sbin     tmp
dev      lib      mnt      root     sys      usr
/ #
```

3、实验结果

挂接网络文件系统成功。

4、实验总结

本实验没什么难的，特别简单，其实就是网络环境设置要对。但网络环境对一些同学就是很难！？是不是要花时间补充一点网络知识，重要的一点，那就是一定要有个路由器，方便很多。

本实验成功的关键点：

- 可以实现三者互 ping
- Ubuntu 安装 tftp 成功并检查（如何检查已说）
- 网络文件系统制作好并检查（如何检查已说）
- 会分析 uboot 环境变量，知道 run netboot 干了啥

5、相关问题总结

更新记录：

20200116 文档起草，第一版诞生。

20200118 完成第 1 节。

20200120 完成第 2 节。

20200121 完成第 3 节。

20200201 完成第 4 节。

20200204 补充第 0 节。

20200218 更新第 5 节。