CS315 Lab8

Name: 王奕童

SID: 11910104

2 Background

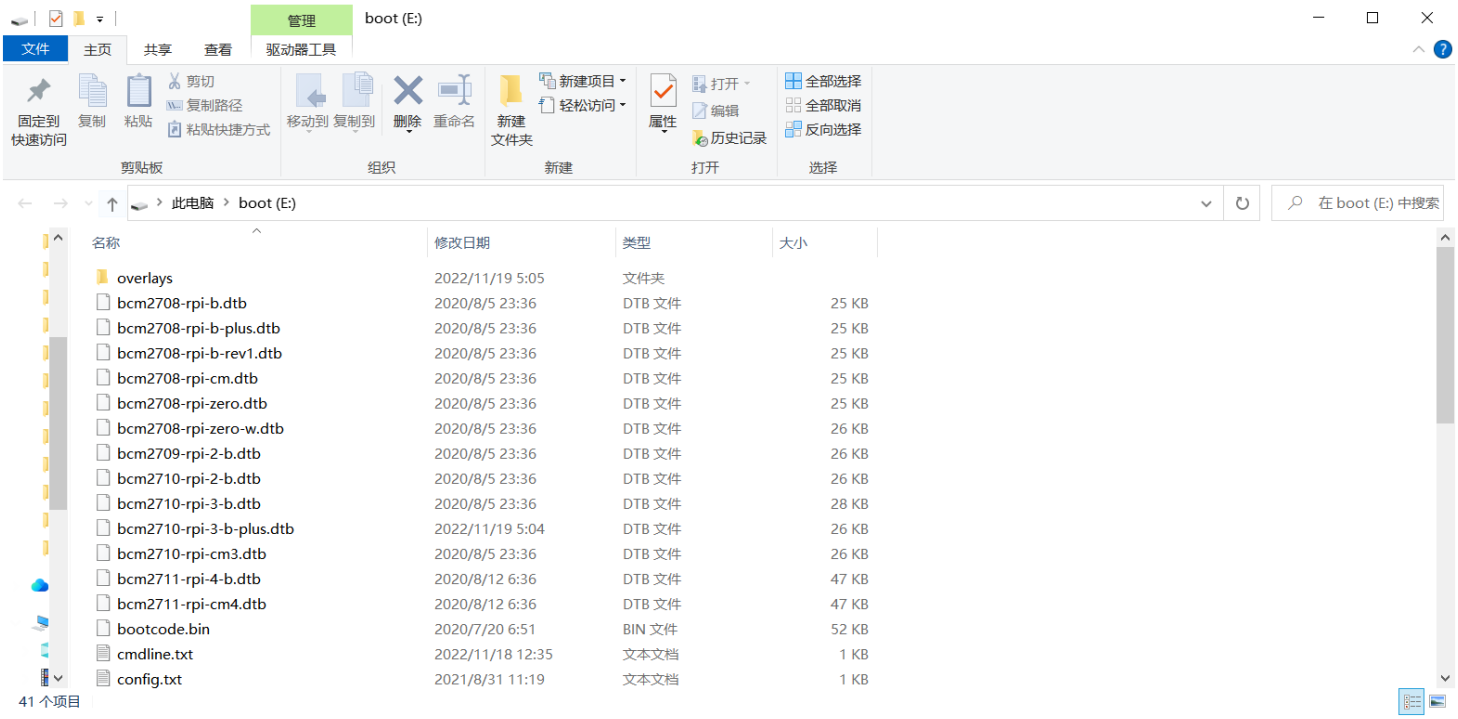
2.1 Your Tools

2.1.1 Hardware

我做这个实验使用了张睿豪同学的树莓派板子。

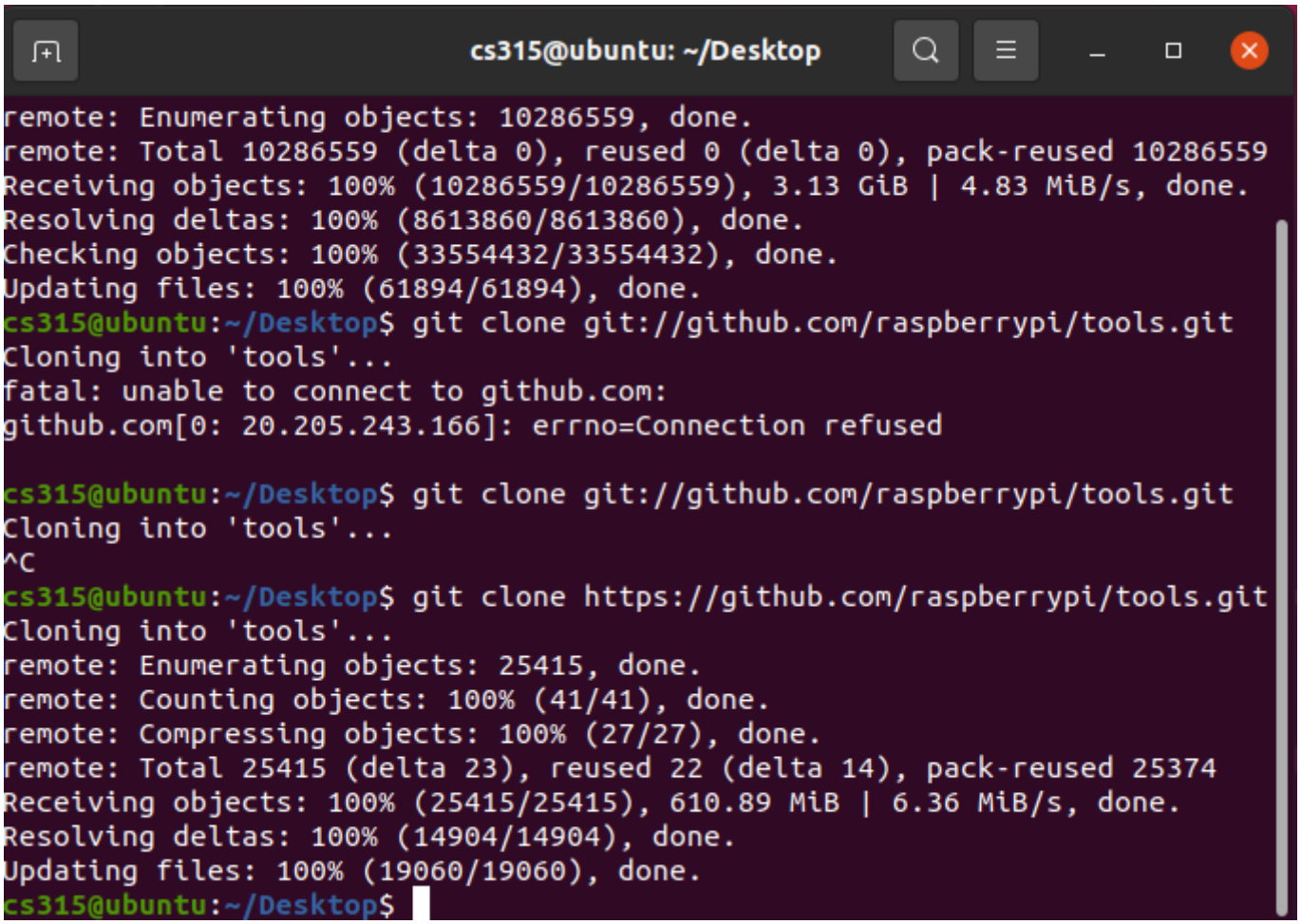
2.1.2 Boot directory

我将树莓派的sd卡拔出来放到电脑上连接，可以正确显示boot目录。



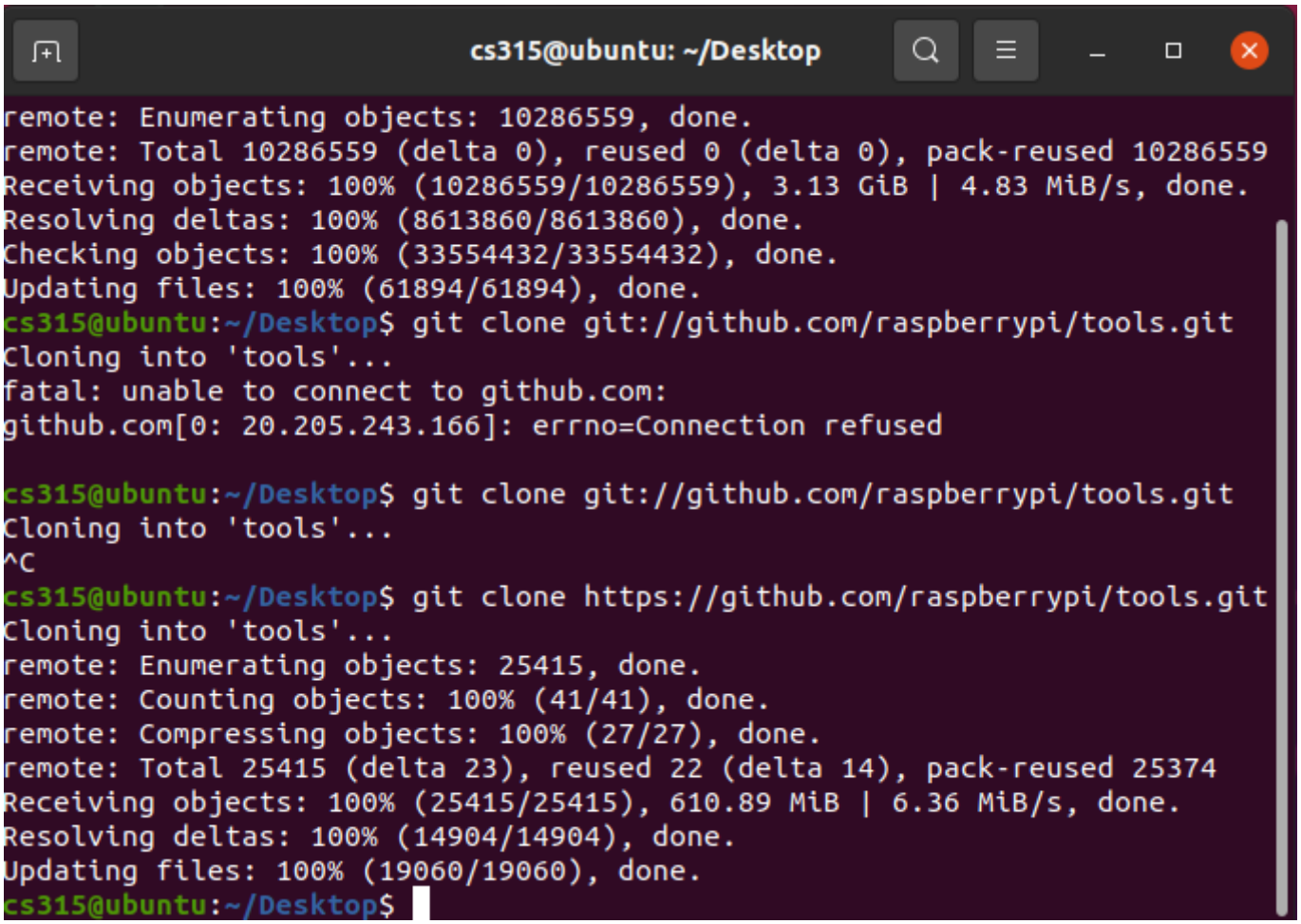
2.1.3 Source Code of Linux Kernel

已clone至ubuntu虚拟机中：



2.1.4 Cross-compile Tools

已clone至ubuntu虚拟机中：



2.2 Armv8-A Exception Levels

2.3 Armv8-A Address Translation

3 Implementation

3.1 Compile the Kernel

3.1.1 Warn

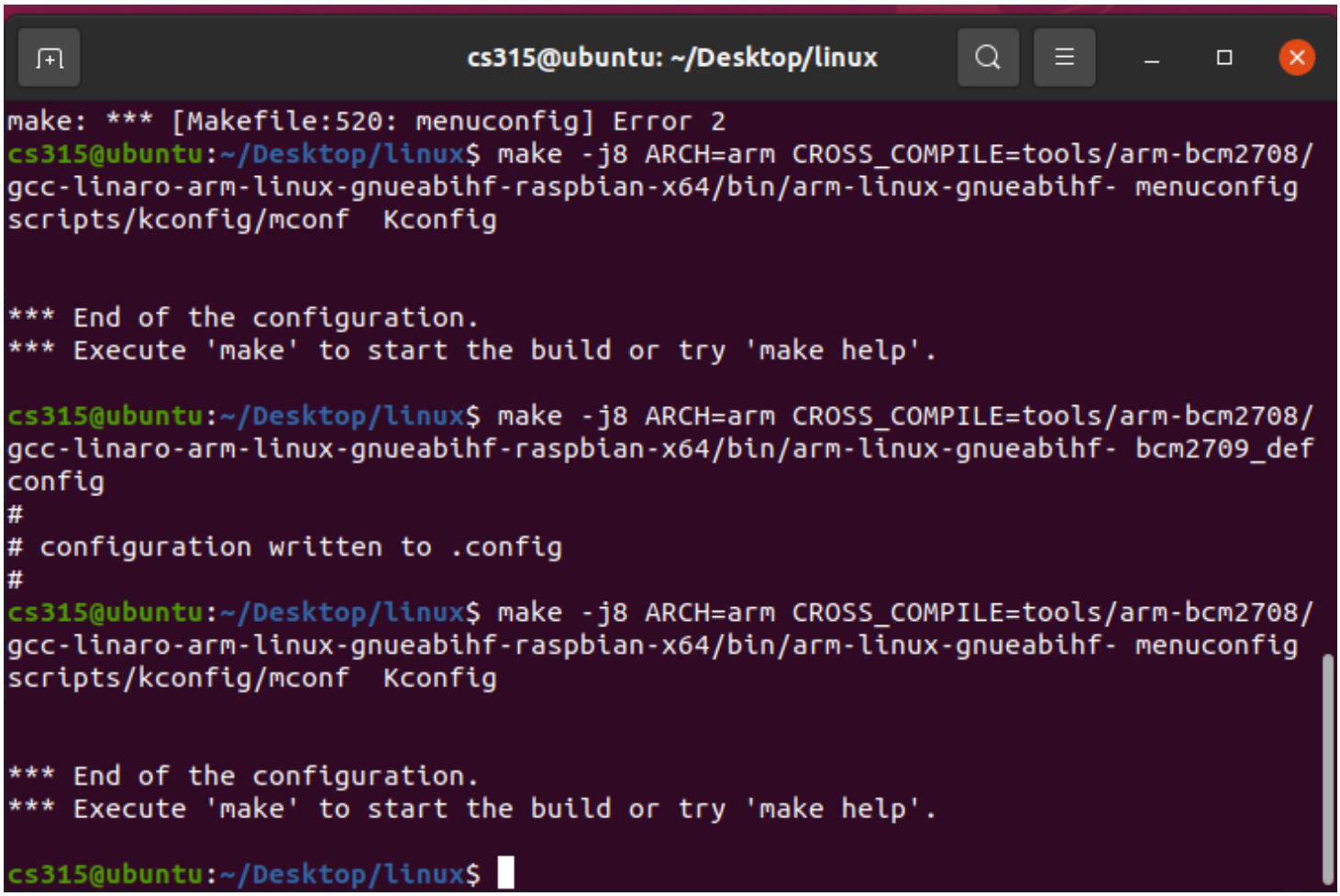
张睿豪同学已经配置树莓派为命令行模式。

3.1.2 Compile

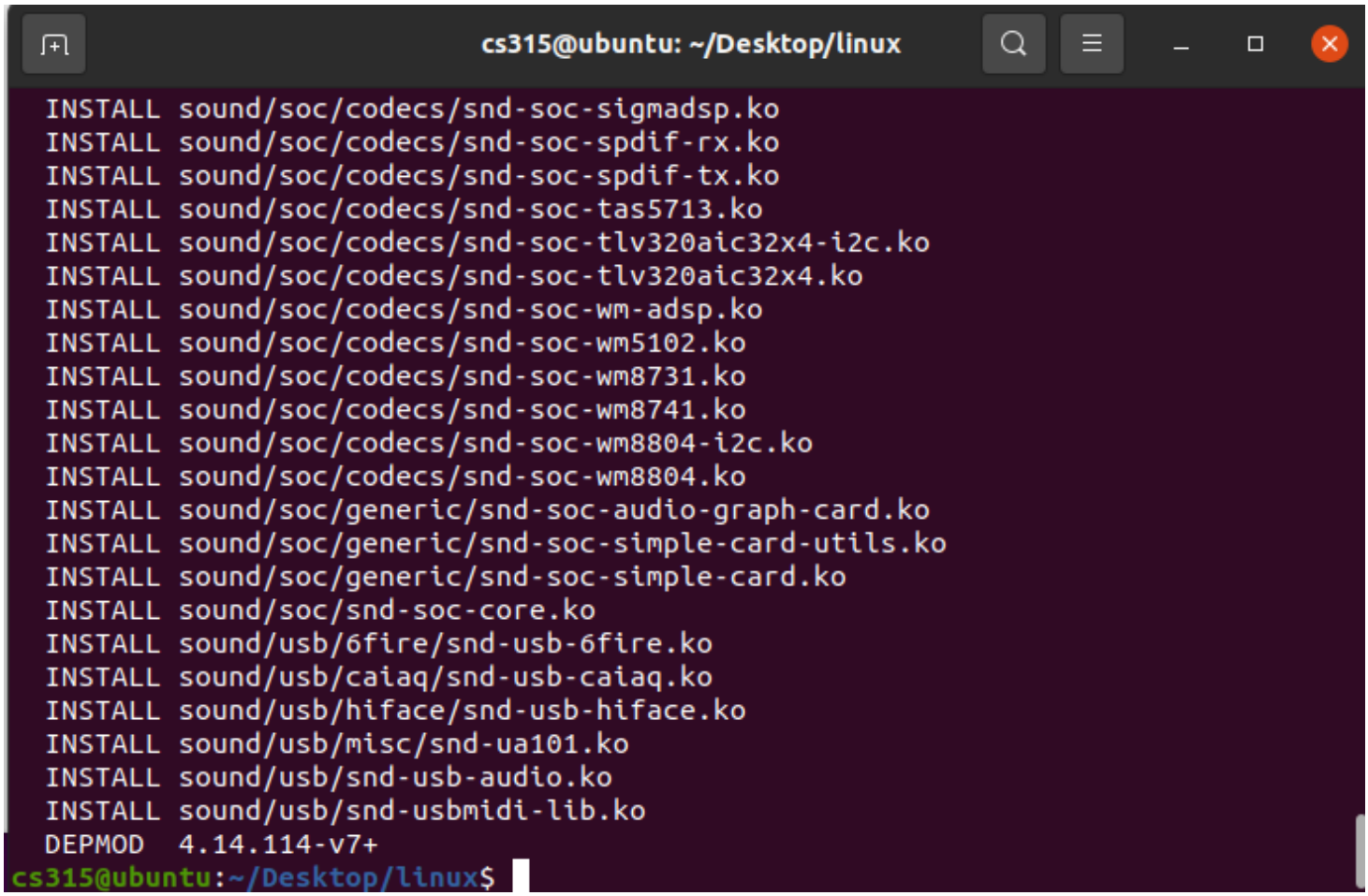
执行课件上的命令：

```
make -j8 ARCH=arm CROSS_COMPILE=tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnuea
```

```
make -j8 ARCH=arm CROSS_COMPILE=tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnuea
```

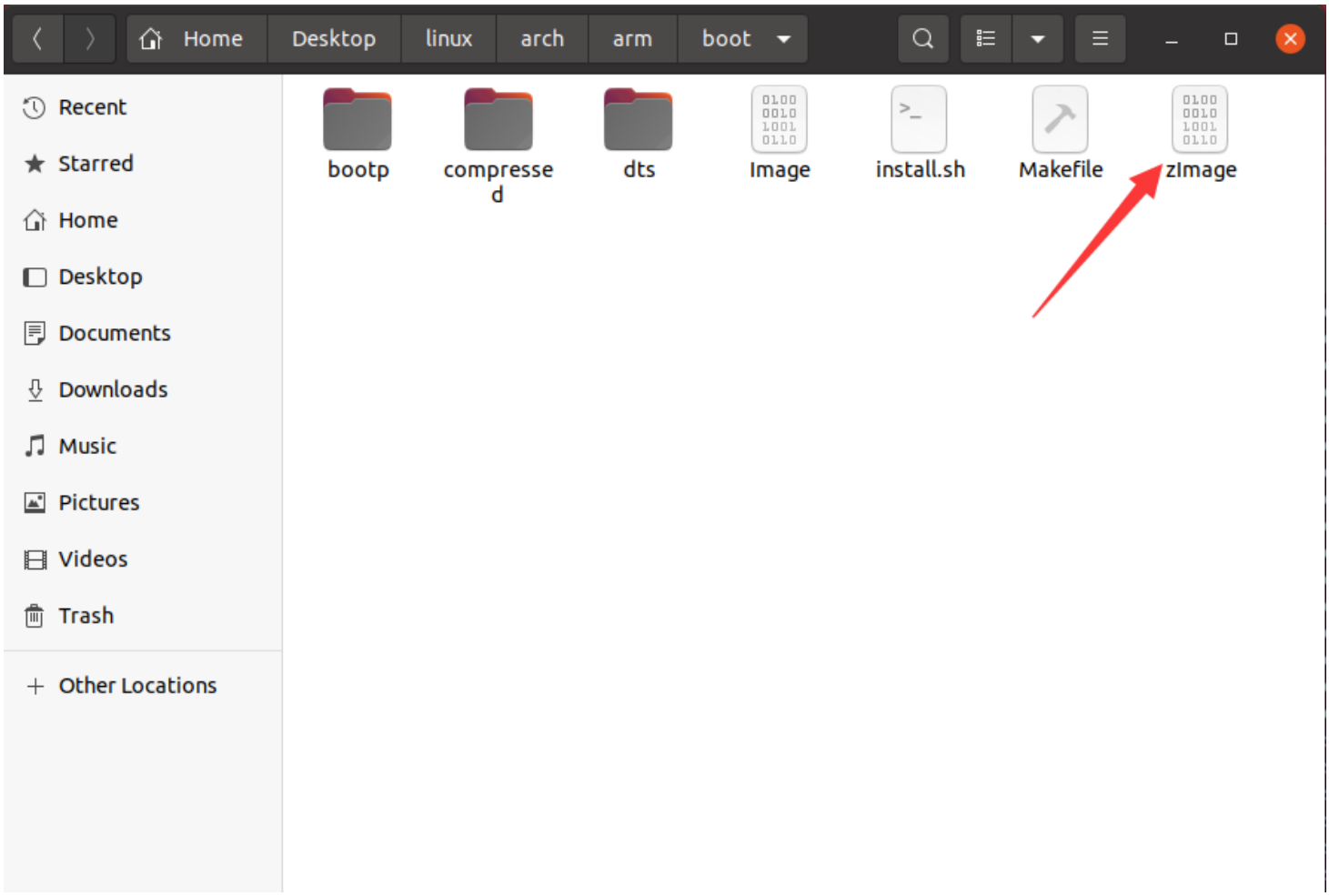



```
mkdir ../modulespath

make -j8 ARCH=arm CROSS_COMPILE=tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnuea

make -j8 ARCH=arm CROSS_COMPILE=tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnuea
```

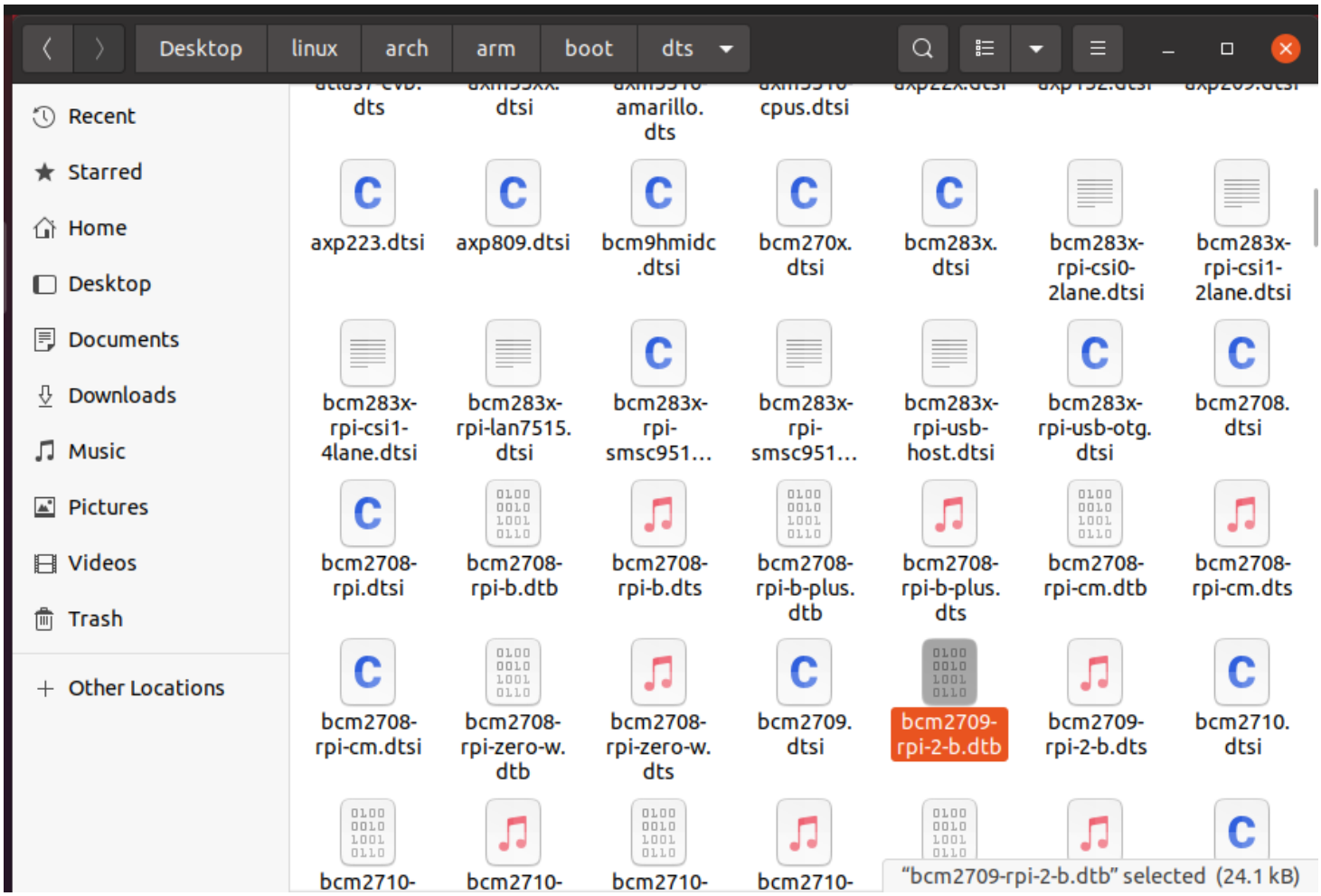


3.1.3 Replace

zImage文件：

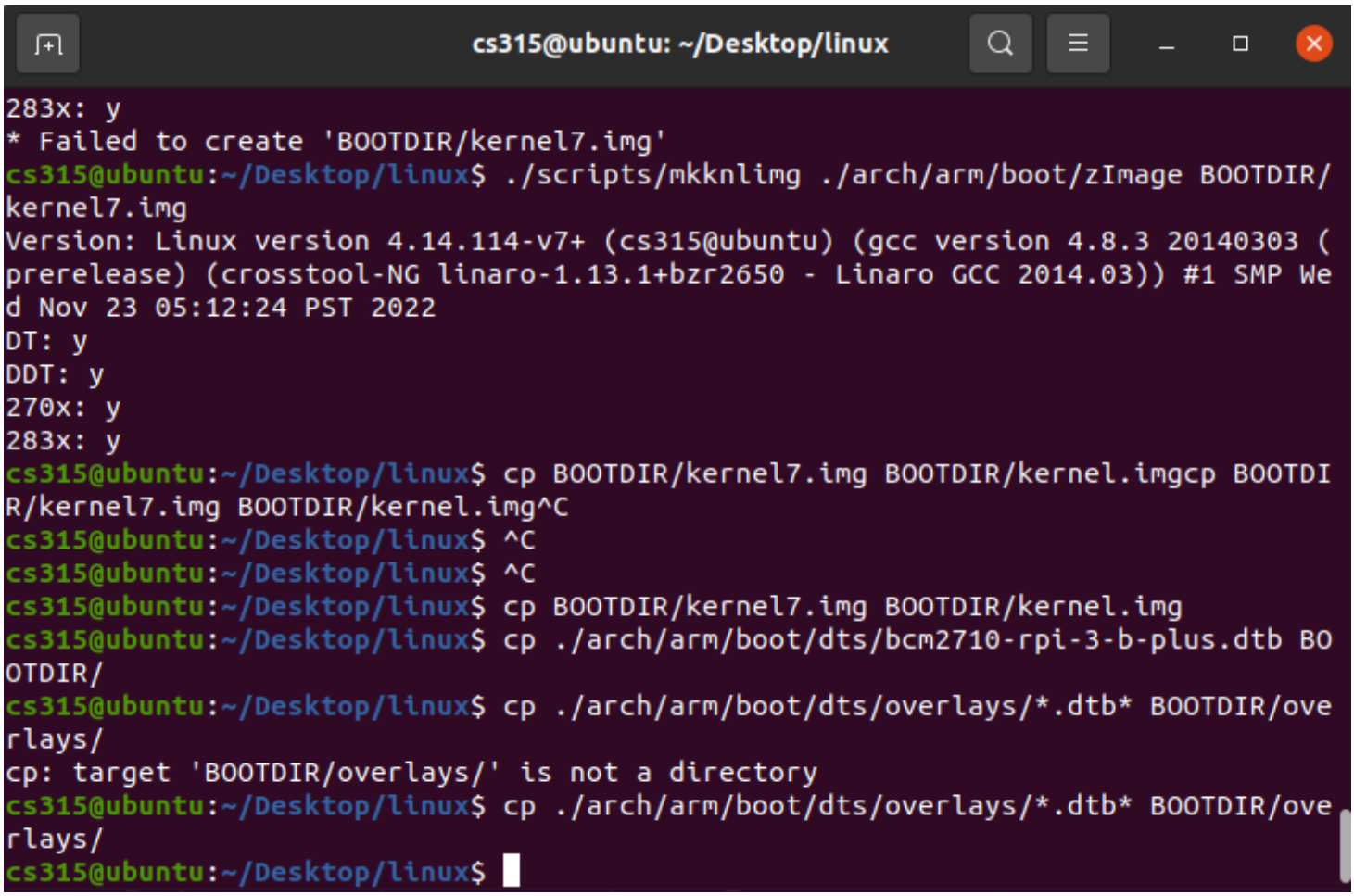

Home
Desktop
linux
arch
arm
boot
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
bootp
compressed
dts
Image
install.sh
Makefile
zImage

dtb文件：

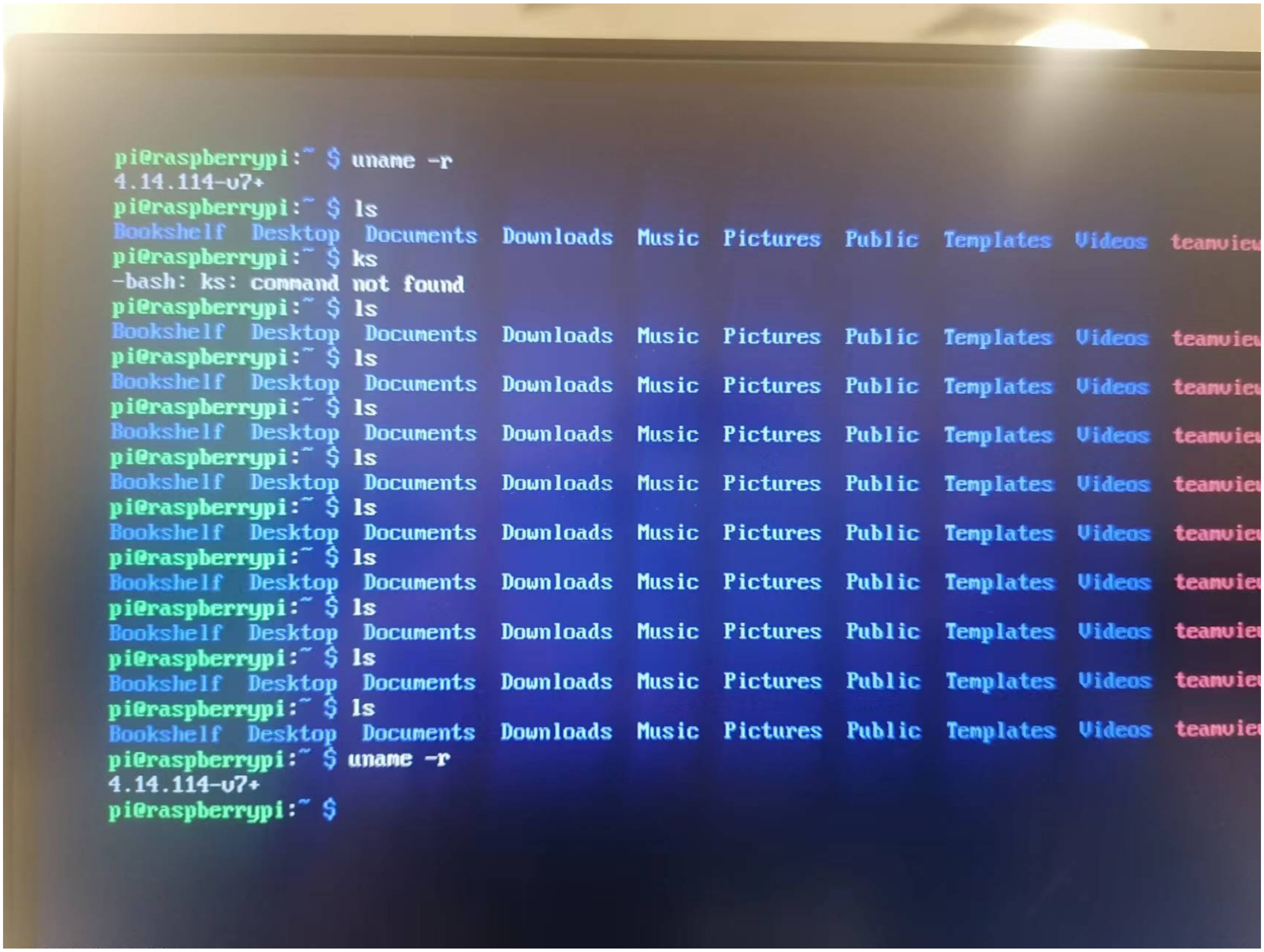


以下命令的执行必须先创建 BOOTDIR 目录：

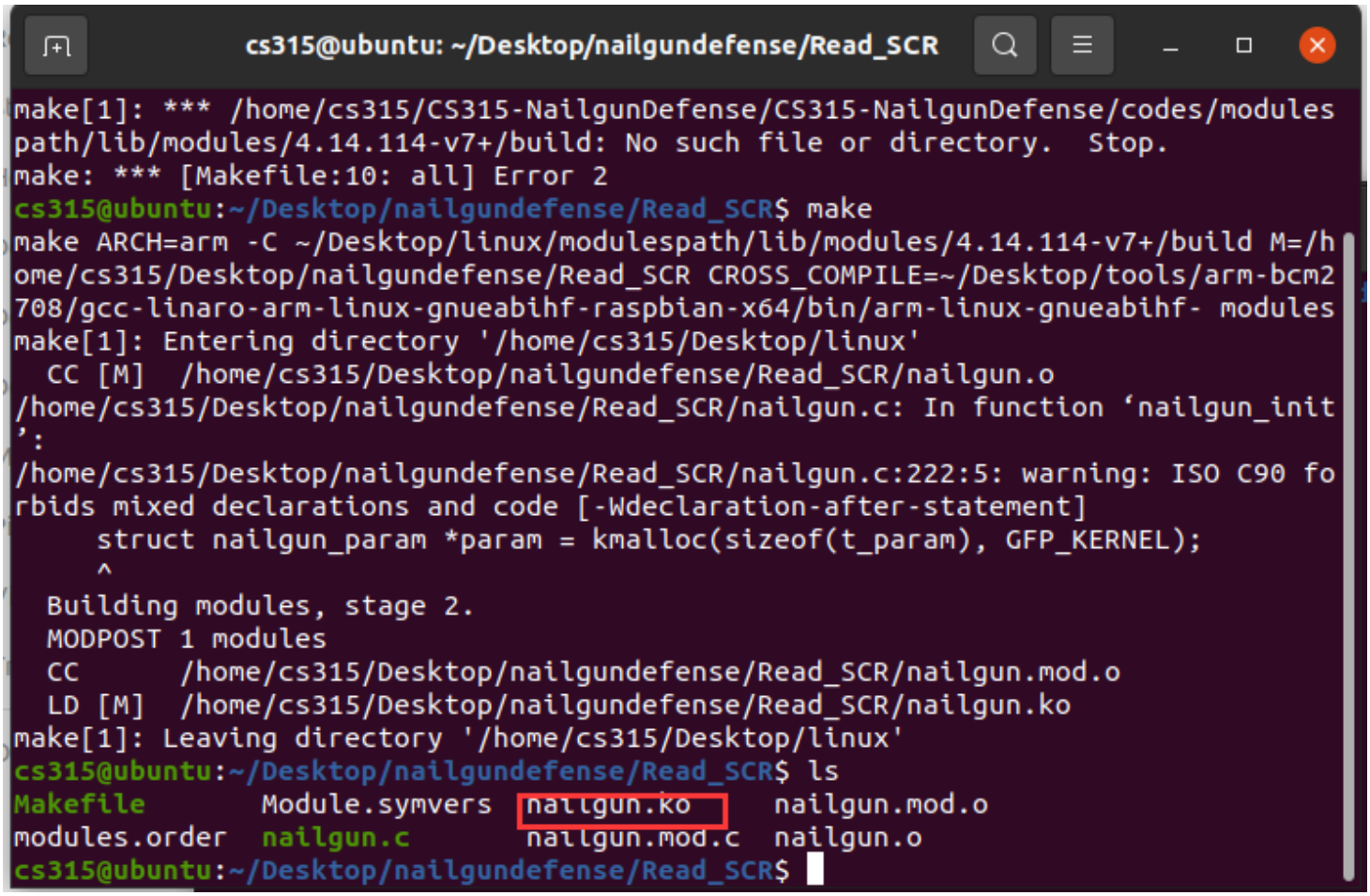
```
./scripts/mkknlimg ./arch/arm/boot/zImage BOOTDIR/kernel7.img
cp BOOTDIR/kernel7.img BOOTDIR/kernel.img
cp ./arch/arm/boot/dts/bcm2710-rpi-3-b-plus.dtb BOOTDIR/
cp ./arch/arm/boot/dts/overlays/*.dtb* BOOTDIR/overlays/
```



在开发板上 `uname -r` 查看kernel版本



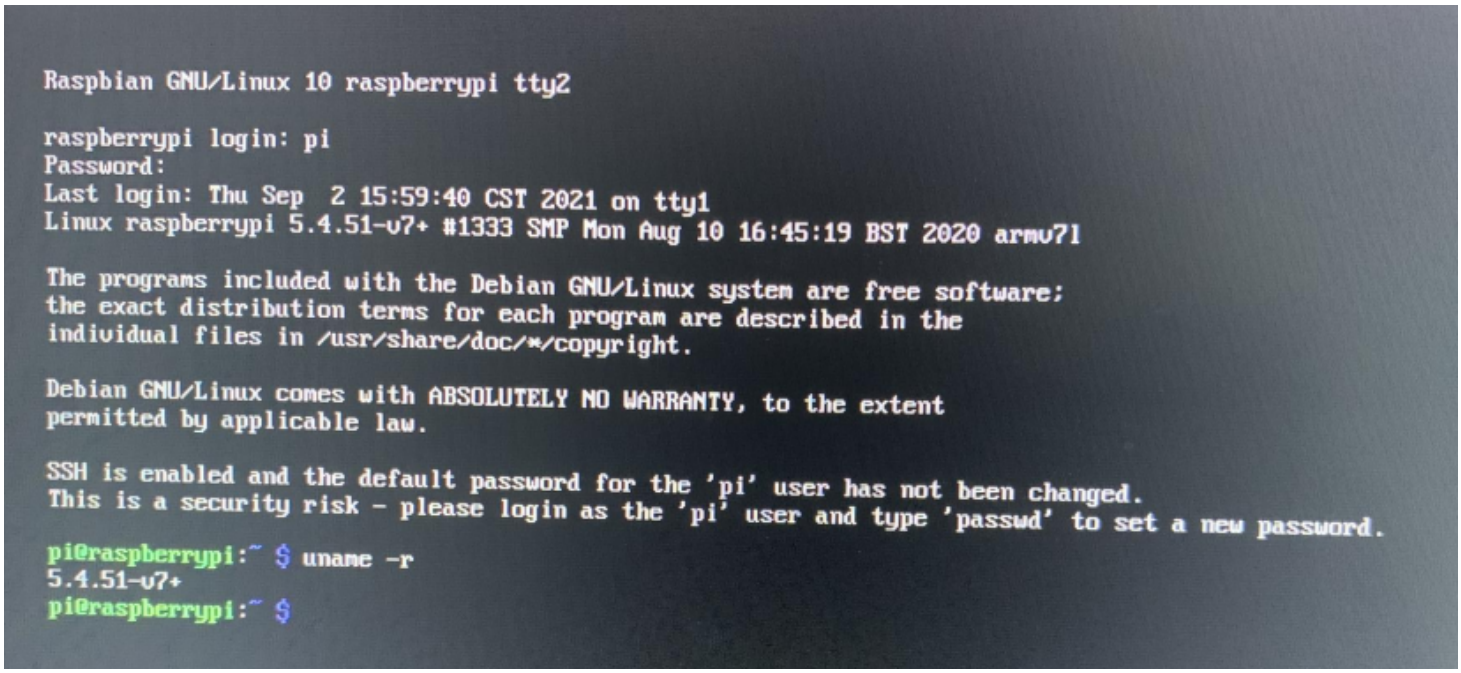
3.1.4 About the Nailgun module

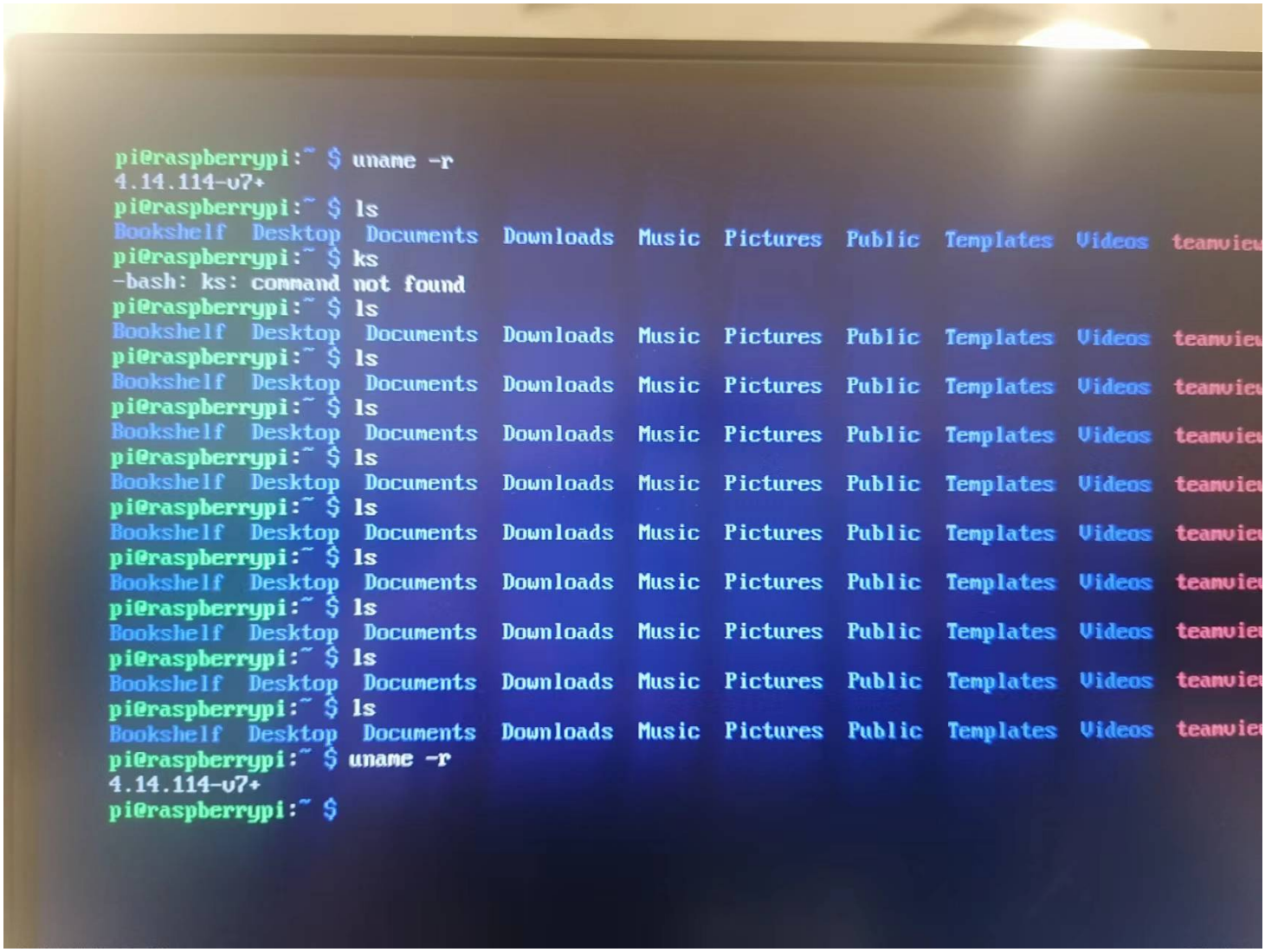


Question 1(20%) Can you prove that (1) you have replaced the kernel (with "uname -r" or other approaches), and (2) you have built the nailgun module with new headers? Please provide a figure.

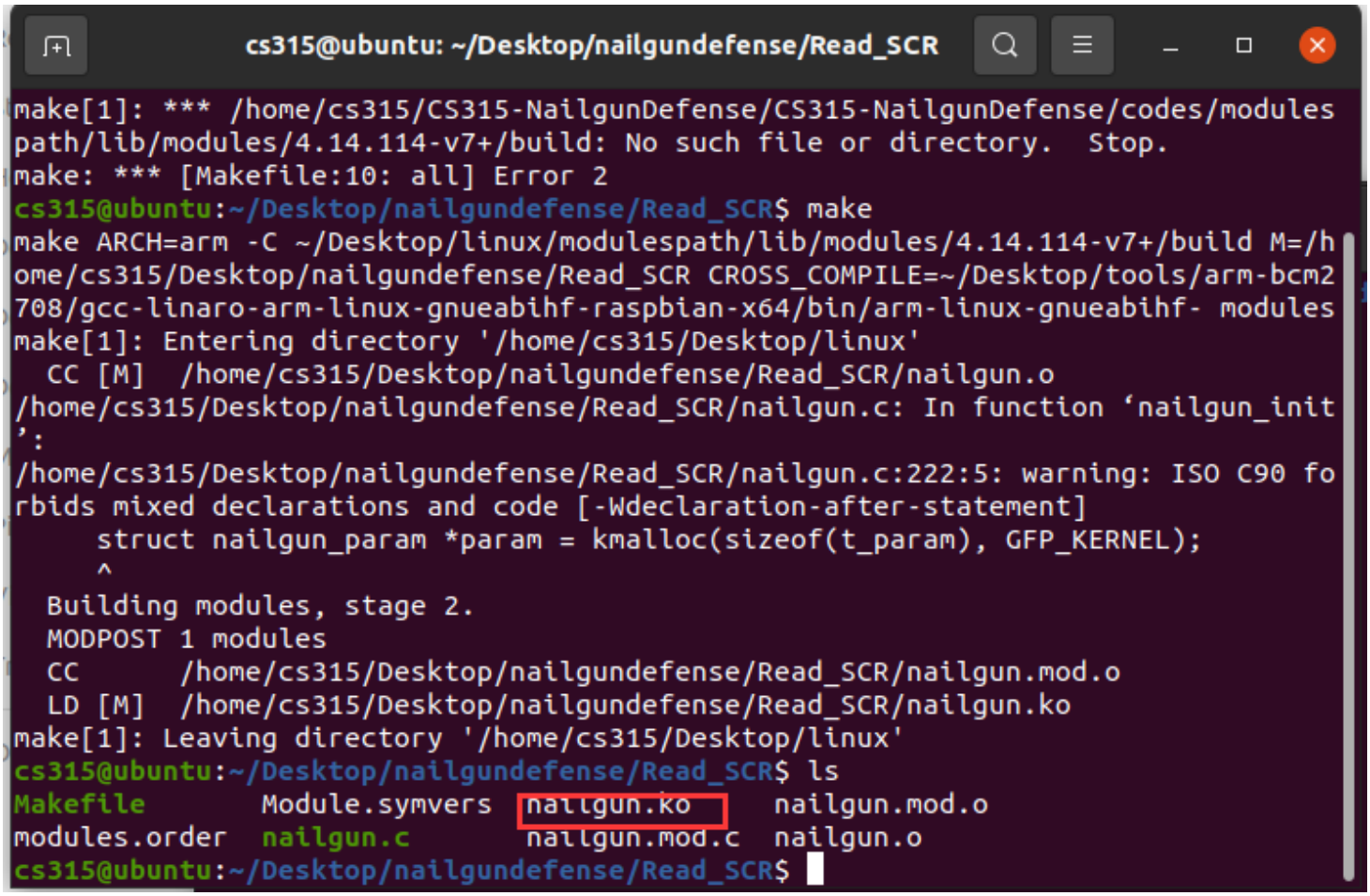
前面的截图已经能够回答该问题。在此重复一次截图：

(1)在开发板上 `uname -r` 查看kernel版本





(2)创建nailgun的module。



Question 2(20%) Can you run the Nailgun Attack on your new kernel? Please provide a figure. You can use "dmesg" to show the execution result of Nailgun Attack.

使用如下命令进行执行：

```
sudo insmod nailgun.ko
dmesg
uname -r
```

其中如果需要再次执行，需要卸载之前的安装：

```
sudo rmmod nailgun
```

```
[ 3041.567031] Step 6: Switch to EL3
[ 3041.567037] Step 7: Read SCR
[ 3041.567042] Step 8: Restore context
[ 3041.567047] Step 9: Send restart request to the target processor
[ 3041.567053] Step 10: Wait the target processor to restart
[ 3041.567061] All done! The value of SCR is 0x00000131
[ 3045.111460] Under-voltage detected! (0x00050005)
[ 3055.511415] Voltage normalised (0x00000000)
[ 3089.311889] Goodbye!
[ 3134.551430] rpi_firmware_get_throttled: 1 callbacks suppressed
[ 3134.551437] Under-voltage detected! (0x00050005)
[ 3138.711426] rpi_firmware_get_throttled: 1 callbacks suppressed
[ 3138.711431] Voltage normalised (0x00000000)
[ 3176.120313] Nailgun Attack Start
[ 3176.120380] Using smp_call_function
[ 3176.120396] Step 1: Unlock debug and cross trigger registers
[ 3176.120402] Step 2: Enable halting debug
[ 3176.120407] Step 3: Halt the target processor
[ 3176.120412] Step 4: Wait the target processor to halt
[ 3176.120417] Step 5: Save context
[ 3176.120423] Step 6: Switch to EL3
[ 3176.120427] Step 7: Read SCR
[ 3176.120432] Step 8: Restore context
[ 3176.120438] Step 9: Send restart request to the target processor
[ 3176.120443] Step 10: Wait the target processor to restart
[ 3176.120450] All done! The value of SCR is 0x00000131
[ 3188.631444] Under-voltage detected! (0x00050005)
[ 3194.871424] Voltage normalised (0x00000000)
[ 3195.590581] Goodbye!
[ 3209.939037] Nailgun Attack Start
[ 3209.939097] Using smp_call_function
[ 3209.939112] Step 1: Unlock debug and cross trigger registers
[ 3209.939118] Step 2: Enable halting debug
[ 3209.939122] Step 3: Halt the target processor
[ 3209.939127] Step 4: Wait the target processor to halt
[ 3209.939132] Step 5: Save context
[ 3209.939138] Step 6: Switch to EL3
[ 3209.939142] Step 7: Read SCR
[ 3209.939147] Step 8: Restore context
[ 3209.939152] Step 9: Send restart request to the target processor
[ 3209.939158] Step 10: Wait the target processor to restart
[ 3209.939166] All done! The value of SCR is 0x00000131
[ 3213.591460] Under-voltage detected! (0x00050005)
pi@raspberrypi:~/Desktop/lab8 $ uname -r
4.14.114-v7+
pi@raspberrypi:~/Desktop/lab8 $ _
```

3.2 Implementation of the Defense

3.3 Codes of Defense

Question 3(30%) With the provided source codes, can you explain the process of traslating an IPA, 0x40030000+"last 3 numbers of your student ID", to the same value of PA? (e.g., if your ID is 12150073, then you should translate 0x40030073). In this question, you should mention the (1) address of each descriptor, and (2) value of each descriptor.

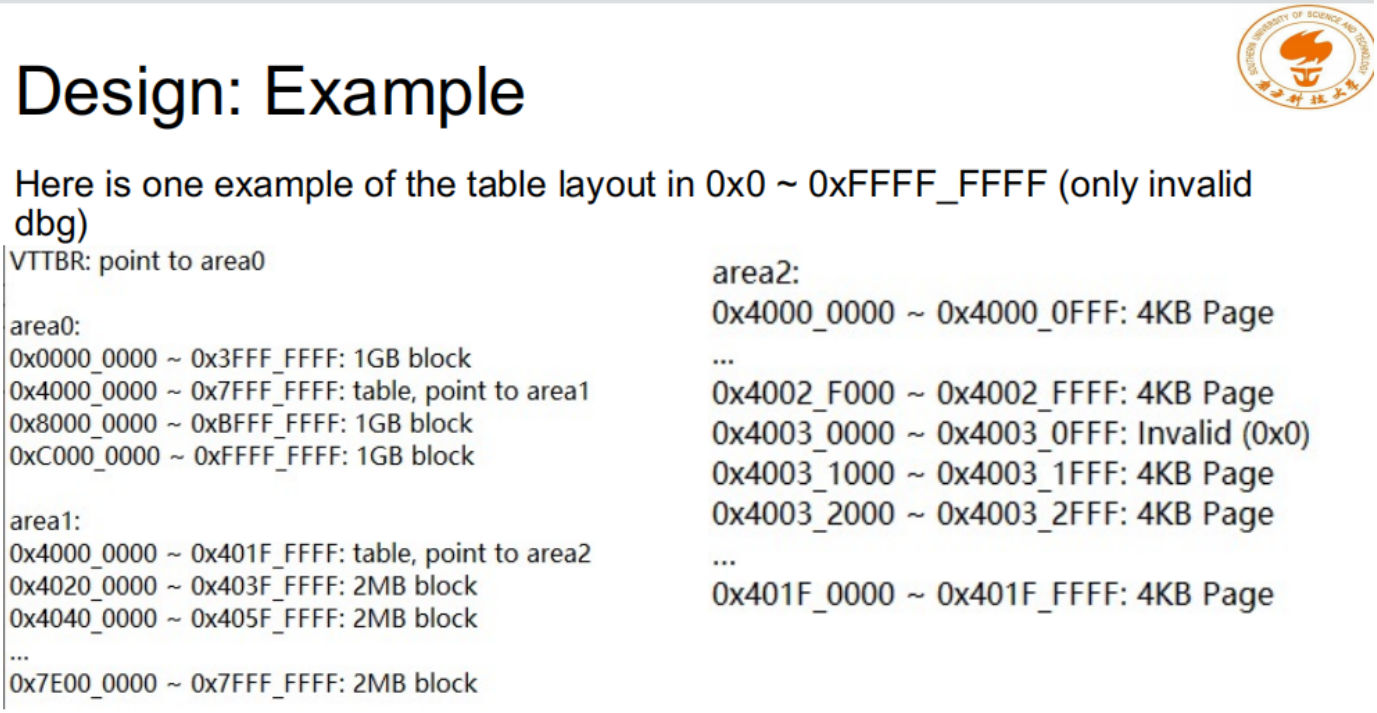
我的SID是11910104，（我假定都是104是16进制数）因此要翻译的IPA是：

```
0x40030000+0x104 = 0x40030104
```

翻译为二进制为：

```
0b 0100_0000_0000_0011_0000_0001_0000_0100
```

我的设计直接参考了大课课件上的内容，分为三个部分：



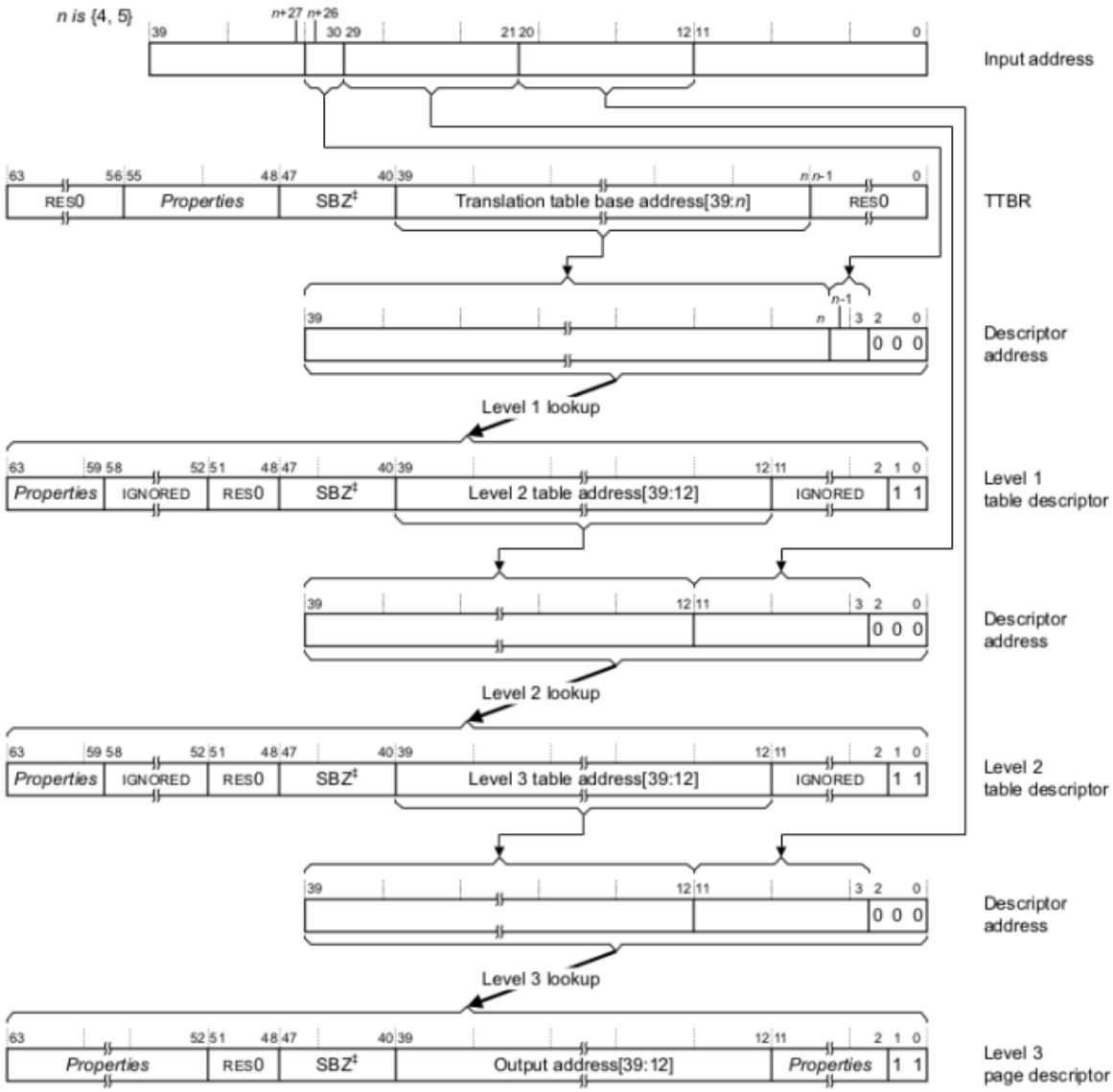


Figure 13: A translation example

VTTBR的数值是 `0x32000000`

First, we fill the VTTBR register, we directly put the start address (0x32000000) into the BADDR bits:

```
ldr r0,=0x32000000
ldr r1,=0x0
mcrr p15, 6, r0, r1, c2
```

这个二进制需要分为好几个部分：

- 一级页表项，31-30位：01
- 二级页表项，29-21位：00_0000_000
- 三级页表项，20-12位：0_0011_0000

- offset项，11-0位：0001_0000_0100

在翻译之前，还要算一下各个页表的首地址，计算结果如下：

- 一级页表：area0，首地址是0x32000000
- 二级页表：area1，首地址是0x32001000
- 三级页表：area2，首地址是0x32005000

页表翻译流程：（翻译过程中有加0b11的操作，这个是手册上规定的数值）

- Step 1: 一级页表项为01

跳转至area1，地址区域范围是0x4000_0000至0x7FFF_FFFF

(1)address: (VTTBR[39:5], IPA[31:30], 0b000) = (0x3200000, 0b01, 0b000)
= 0x32000008

(2)value: 它指向area2区域，地址是：

A2 = area1 + offset * block_size + 0b11 = 0x32001000 + 1 * 2^12 + 0x3 = 0x32002003

最末两位是11

- Step 2: 二级页表项为全0

跳转至area2，地址区域范围是0x4000_0000至0x401F_FFFF

(1) address: 考虑其为二级页表项的首项，因此即为Step 1中的value，为0x32002000

(2) value: 它指向area3区域，地址是：

A3 = area2 + offset * block_size + 0b11 = 0x32005000 + 0 * 2^12 + 0x3 = 0x32005003

最末两位是11

- Step 3: 三级页表项为0_0011_0000

跳转至area3，地址区域范围是0x4003_0000至0x4003_0FFF

(1) address: (A3[39:12], IPA[20:12], 0b000) = 0x32005180

(2) value: 它指向的地址是：

0x40030 << 12 + 0b11 = 0x40030003，最末两位是11

- Step 4: offset项是0001_0000_0100

翻译结果就是0x40030 << 12 + 0001_0000_0100 = 0x40030104，和原来的IPA地址相同。

Question 4(30%) With the provided source codes, can you explain the process of traslating an IPA, 0x40000000+"last 7 numbers of your student ID", to the same value of PA? (e.g., if your ID is 12150073, then you should translate 0x42150073). In this question, you should mention the (1) address of each descriptor, and (2) value of each descriptor.

该题流程和Question 3类似，设计保留Question 3的想法。

我的SID是11910104，（我假定都是1910104是16进制数）因此要翻译的IPA是：

```
0x40030000+0x1910104 = 0x41940104
```

翻译为二进制为：

```
0b 0100_0001_1001_1000_0000_0001_0000_0100
```

这个二进制需要分为好几个部分：

- 一级页表项，31-30位：01
- 二级页表项，29-21位：00_0001_100
- 三级页表项，20-12位：1_1000_0000
- offset项，11-0位：0001_0000_0100

页表翻译流程：

- Step 1: 一级页表项为01

跳转至area1，地址区域范围是0x4000_0000至0x7FFF_FFFF

(1) address: (VTTBR[39:5], IPA[31:30], 0b000) = (0x3200000, 0b01, 0b000)
= 0x32000008

(2) value: 它指向area2区域，地址是：

A2 = area1 + offset * block_size + 0b11 = 0x32001000 + 1 * 2^12 + 0x3 = 0x32002003

最末两位是11

- Step 2: 二级页表项为1100

跳转至第12个2 MB block，地址区域范围是0x4180_0000至0x419F_FFFF

(1) address: (A2[39:12], IPA[29:21], 0b000) = 0x32005C00

(2) value:

A3 = area2 + offset * block_size + 0b11 = 0x32005000 + 12 * 2^12 + 0x3 = 0x32011003

最末两位是11

- Step 3: 三级页表项为0_0011_0000 1_1000_0000

跳转至area3，地址区域范围是0x4194_0000至0x4194_0FFF

(1) address: (A3[39:12], IPA[20:12], 0b000) = 0x32011C00

(2) value: 它指向的地址是：

0x41940 << 12 + 0b11 = 0x41940003，最末两位是11

- Step 4: offset项是0001_0000_0100

翻译结果就是0x41940 << 12 + 0001_0000_0100 = 0x41940104，和原来的IPA地址相同。

Acknowledgement

本次实验感谢张睿豪同学提供开发板环境配置，谢岳臻同学提供Q3和Q4的设计与算法讲解。

我这次是和刘晟淇同学一起完成了本次实验。