

嵌入式系统仿真实验第十讲

从今天开始我们将正式进入到系统软件，前面我们一致参照的 U-Boot 的结构写的 boot 程序，今天我们将直接开始使用 U-Boot 了。有关 bootloader 的原理，我的理论课程讲义里面写的很清楚，分为两个 stage，第一个 stage 我们已经介绍过，现在开始第二个 stage，具体的地方如果不清楚的，请自行学习理论课程讲义，这里只是简单再贴一贴我画的第二阶段大致流程图。

我们在理论课程里讲过 stage 1 的功能：RAM 初始化，设置各个部件的时钟和片选，将 BootLoader 拷贝到 RAM 中，设置堆栈，调用 Stage 2。

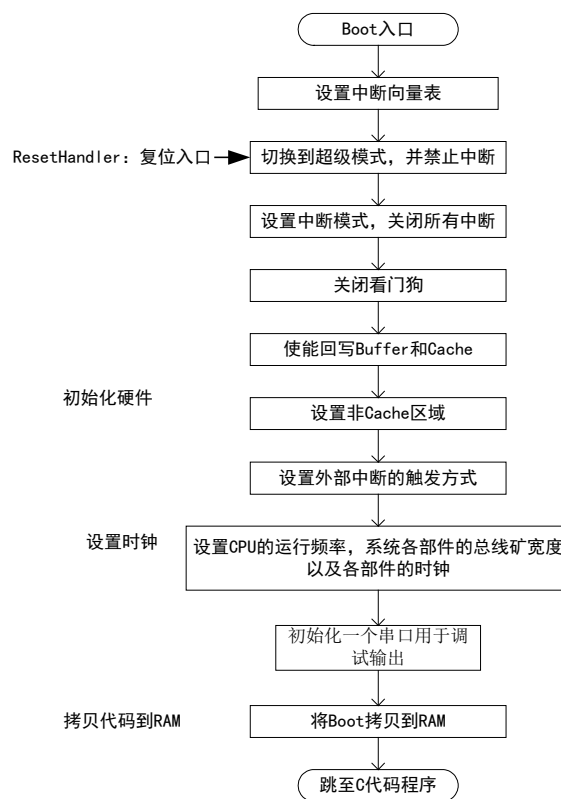


图 1 Stage 1 流程图

图 1 是我们在第 6 讲就画过的第一阶段的流程，对应 U-Boot 的具体流程见图 2。

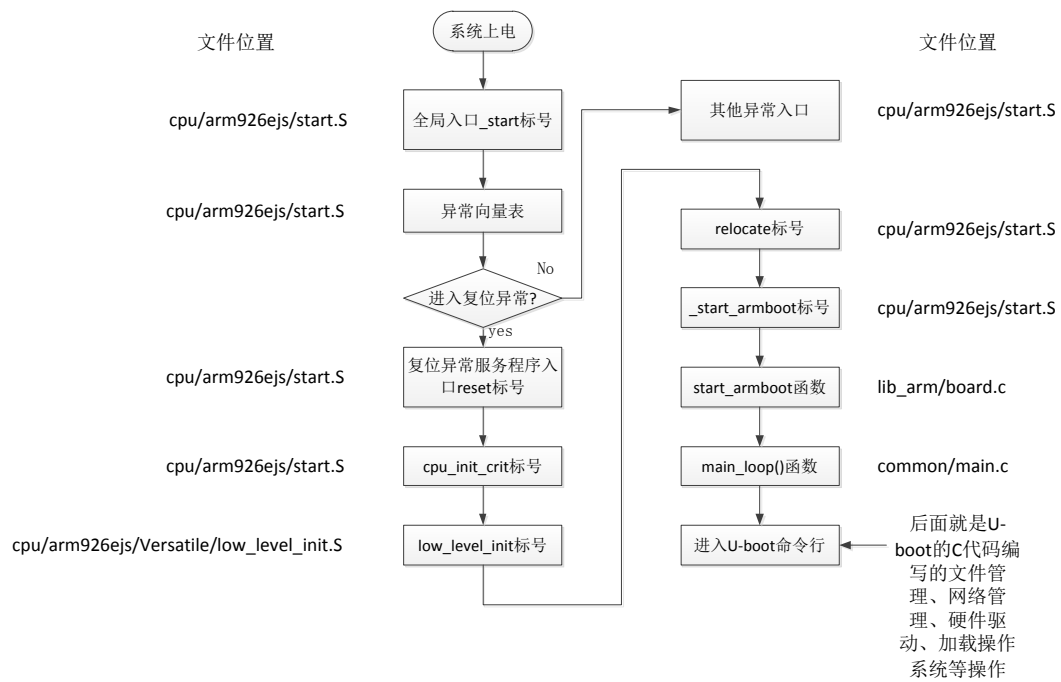


图 2 U-Boot 的流程图

U-Boot 的第二阶段从 `start_armboot` 算起，完成硬件初始化，完成 stage2 的第一步。函数在 `lib_arm/board.c`，大致是如下的内容：

```

void start_armboot (void)
{
    .....

    size = flash_init ();

    display_flash_config (size);

    .....

    for (;;)

        main_loop ();

}

```

由 `start_armboot` 进入到 `main_loop ()` 函数，其流程大致如图 3 所示。

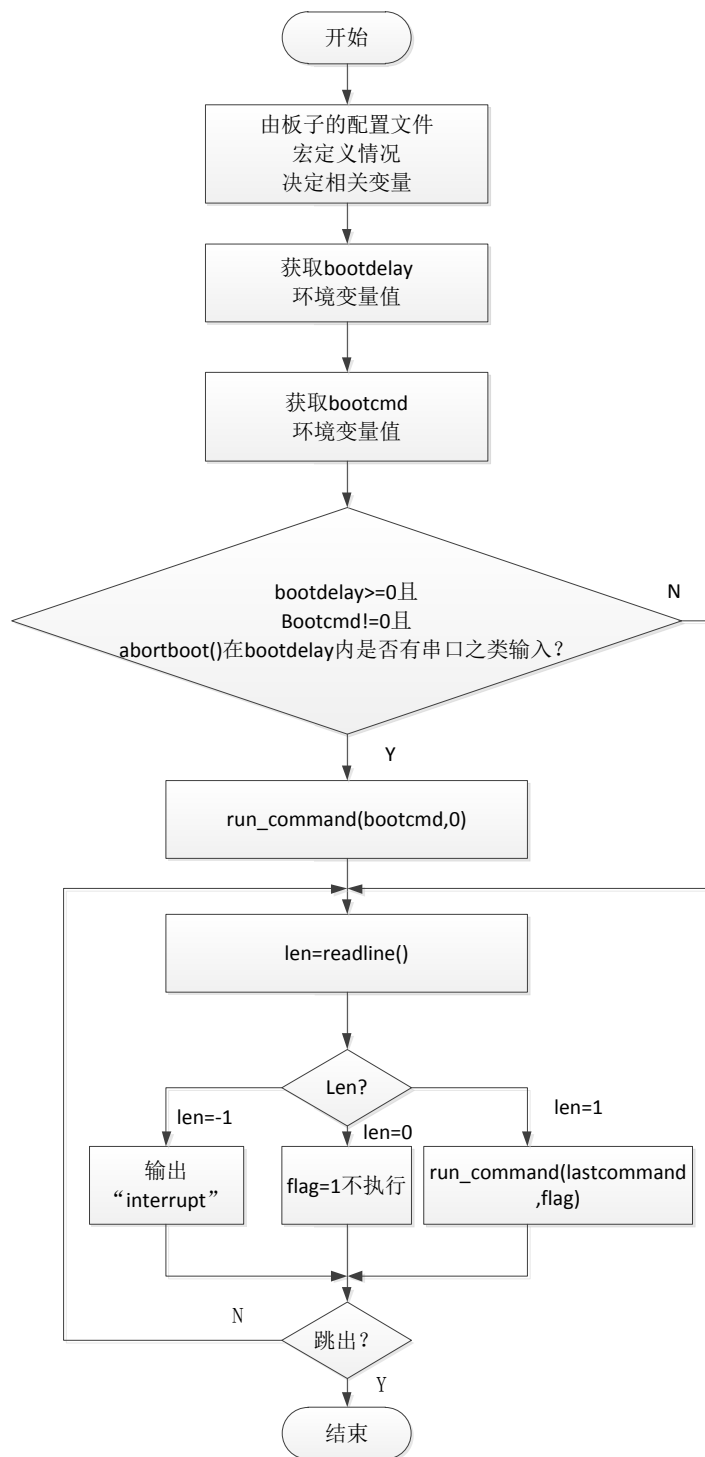


图 3 main_loop 函数的大致流程

在我前面的范例中，main_loop 函数只是串口打印了一个菜单，然后演示了几个接口器件的驱动，而 U-Boot 则要复杂的多，我们不可能一一去讲，这里面主要是 run_command 函数里面实现了很多的 U-Boot 命令函数。比如 do_boot_linux 函数，就是将 linux 内核拷贝到内存，然后启动内核，对应的是 bootm

等命令。其他的一些主要命令请参考 U-Boot 的手册。

1 U-Boot 下载、编译、调试

我们还是直接来做实验，我选择了 u-boot-2010.09 这个版本的 U-Boot 来做实验，首先到网上去找源代码压缩包，哪里方便下载，你就去哪里下。下完之后，解压，后面边做实验边阅读体会代码。

```
tar -jxvf u-boot-2010.09.tar.bz2
cd u-boot-2010.09/
```

进入解压目录，然后指定 CPU 架构类型，交叉编译器以及板子配置文件：

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- versatilepb_config
```

编译：

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- -s
sudo ln -sf $PWD/tools/mkimage /usr/local/bin/mkimage
```

编译下面那句命令是建立链接文件 mkimage 这个工具，我们后面需要用到这个工具将二进制执行文件生成 image 映像文件。

编译成功后，源代码解压的目录里会出现几个文件。

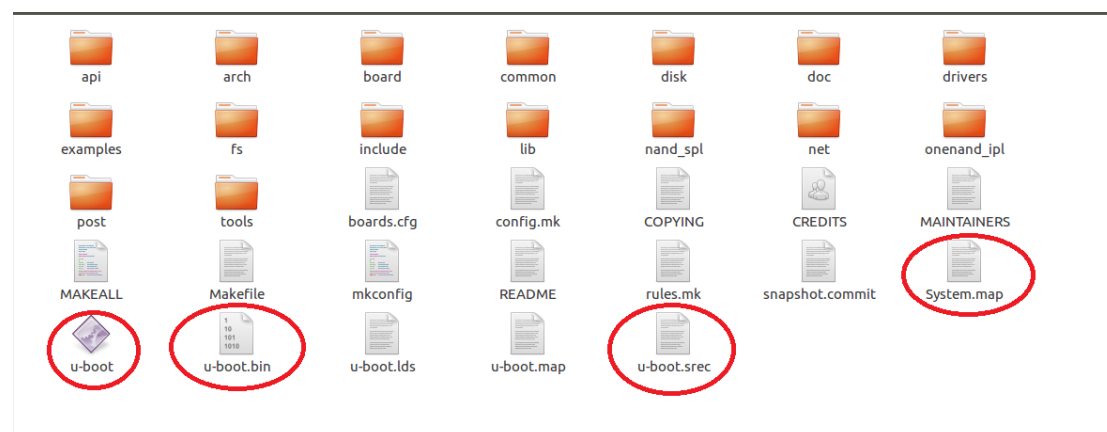


图 4 编译成功

u-boot.map 地址和符号的映射关系（System.map）

u-boot ELF 格式的可执行文件 (这个可以用我们前面讲的方法调试)

u-boot.bin 二进制文件它就是可以直接烧入 ROM、NOR Flash 的文件

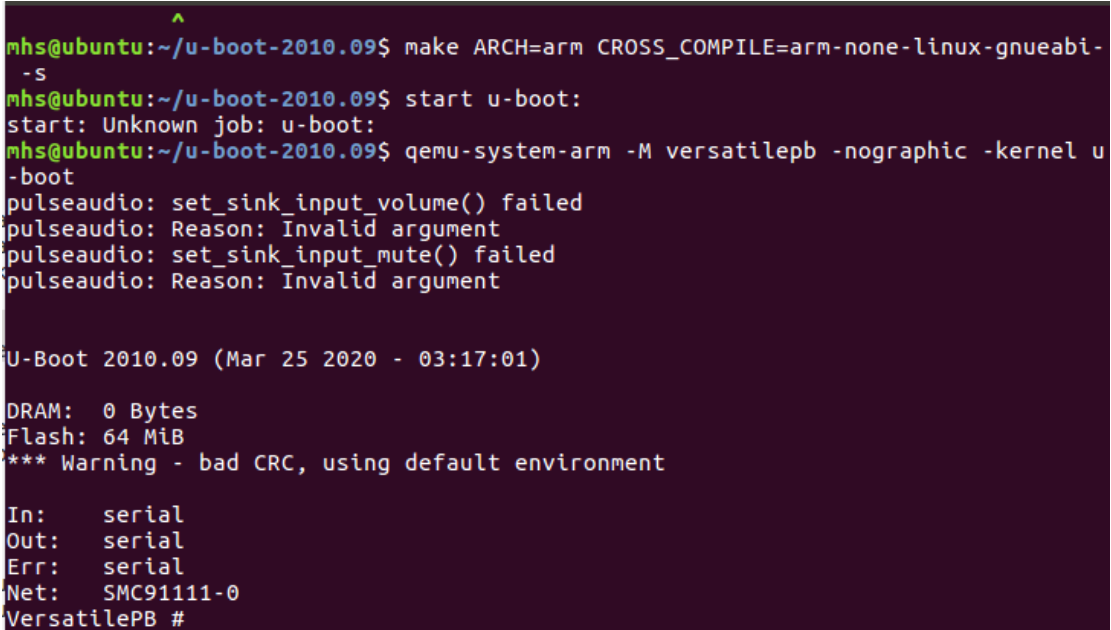
u-boot.srec Motorola S-Record 格式的可执行文件

生成这些文件后，接下来我们就可以把这个 U-Boot 烧写到机器人 flash 里面去，然后我们就可以告别 bare metal programming 了。

同样的，我们可以用 gdb 和 qemu 来仿真调试这个 U-Boot 文件了。

先运行一下 u-boot:

qemu-system-arm -M versatilepb -nographic -kernel u-boot



```
mhs@ubuntu:~/u-boot-2010.09$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
-S
mhs@ubuntu:~/u-boot-2010.09$ start u-boot:
start: Unknown job: u-boot:
mhs@ubuntu:~/u-boot-2010.09$ qemu-system-arm -M versatilepb -nographic -kernel u
-boot
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument

U-Boot 2010.09 (Mar 25 2020 - 03:17:01)

DRAM:  0 Bytes
Flash: 64 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:    serial
Err:    serial
Net:    SMC91111-0
VersatilePB #
```

图 5 qemu 仿真运行 U-Boot

可以看到 U-Boot 成功运行，出现提示后，停留在命令行等待状态。可以测试一下 printenv 之类命令看看配置情况。

使用 DDD 和 gdb 我们也是可以调试 U-Boot 代码的：

1) qemu-system-arm -M versatilepb -m 128M -gdb tcp::1024 -serial stdio -kernel u-boot -S

2) ddd --debugger arm-none-linux-gnueabi-gdb u-boot

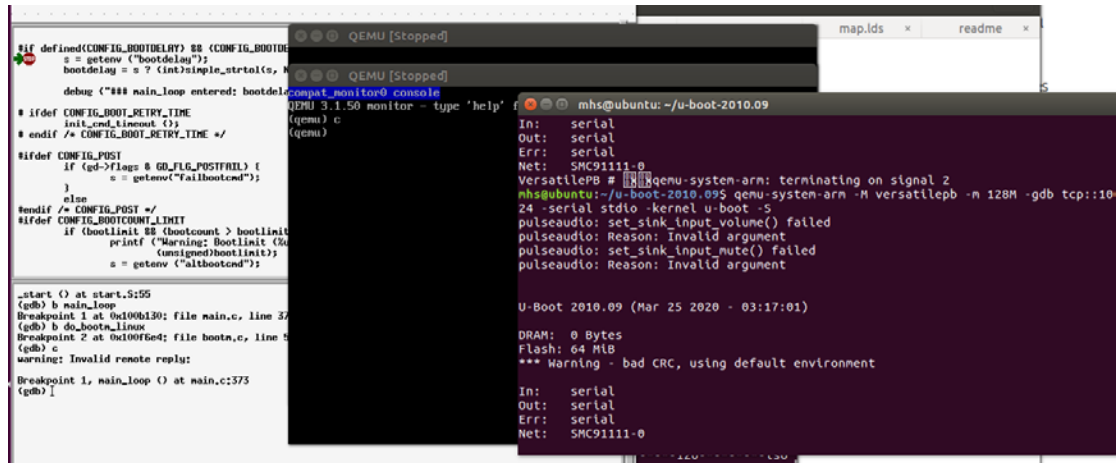


图 6 gdb 调试 U-Boot

需要注意的是由于我用的虚拟机，qemu 的窗口总是被 stopped 掉的，需要我手工在 qemu 的命令窗口(按 ctrl+alt+2 会出现)按‘c’，所以我没有使用 -nographic 命令。

2 U-Boot 加载自编写程序的仿真

Bootloader 就是 boot+loader，目前我们还没有 loader 一个操作系统，目前操作系统我们还没有讲到，那我们能不能体会一下 bootloader 是如何加载系统的呢？

接下来我们来尝试做一个实验。前面我们一致都是认为自己是做了很多编程的，特别是到了 ex14 的实验，一步一步已经小有规模了，这个程序能不能当系统程序用呢？我们说这应该是完全可以的。我们可以将 U-Boot 作为我们的系统版的 BIOS 程序，然后如果我们并没有多线程、多任务的环境需求，而仅仅只是一个 while 循环轮询程序就可以解决的话，我们是没有必要非要弄一个操作系统来的。这样的话我们既可以让 U-Boot 来做我们的程序管理、也可以用 U-Boot 来当我们的 flash update 程序完成版本更新。当然，如果只是为了学习，有时间我们其实也可以把 ex14 接着往下，加入时钟、多线程、信号量、上下文切换、任务调度、内存管理等函数一一实现，自己去写一个操作系统出来。

我们知道 U-Boot 有个命令是 bootm 命令是从内存中加载操作系统内核，现在我们假设 ex14 就是一个操作系统的内核，我们来用 U-Boot 去加载它运行，看

可不可以。

目前有多种方法去加载这个实验，U-Boot 支持内存文件系统加载、tftp 下载加载、以及 nfs 等方法进行加载，这一次我们选择用内存加载。

首先我们需要理清一下，如果我们把 U-Boot 和 ex14 的执行文件都放到内存中去仿真，我们需要将这两个文件连在一起加载才行，linux 里面有最简单的 cat 命令，将多个文件输入到一个文件中。

在 cat 之前，我们将 ex14 的执行文件 test.bin 生成 image 文件：

```
mkimage -A arm -C none -O linux -T kernel -d test.bin -a 0x00300000 -e 0x00300000
test.uimg
```

我们把 test.bin 打算放到内存地址 0x300000 处，这个主要是考虑到 U-Boot 本身的大小，以及堆栈开辟情况来决定把 test.bin 文件放到哪里去。

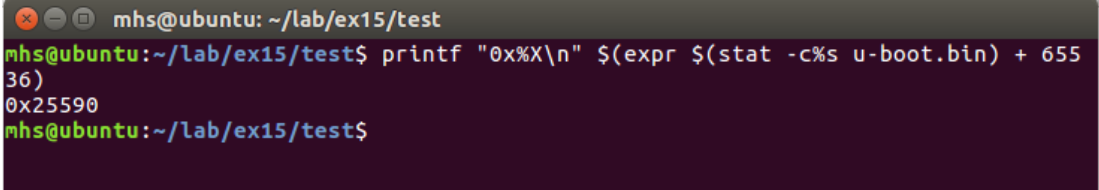
接下来把它们俩合并：

```
cat u-boot.bin test.uimg > flash.bin
```

生成一个合并了的 flash.bin 二进制执行文件，这个有点像早期的一些病毒程序，把自己附在某个执行文件后面，修改执行文件某条语句，让用户执行程序同时也把病毒加载到内存中去运行。

在仿真运行这个 flash.bin 之前，我们需要知道 test.uimg 的起始位置是什么，由于 u-boot 在前，所以我们使用下面的语句查看一下地址：

```
printf "0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536)
```



```
mhs@ubuntu: ~/lab/ex15/test
mhs@ubuntu:~/lab/ex15/test$ printf "0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536)
0x25590
mhs@ubuntu:~/lab/ex15/test$
```

图 7 查看 test.bin 的起始地址

我们编译的 u-boot.bin 有 0x25590 个字节。所以，后面我们启动 u-boot.bin 后，可以用 bootm 0x25590 命令将 test.bin 加载到 0x300000 处执行。

注意在 ex14 里面需要将 map.lds 文件的链接起始地址也改为 0x300000,同样, Makefile 里面也得改。

运行命令:

```
qemu-system-arm -M versatilepb -serial stdio -semihosting -kernel flash.bin
```

出现 U-Boot 界面, 见图 8。

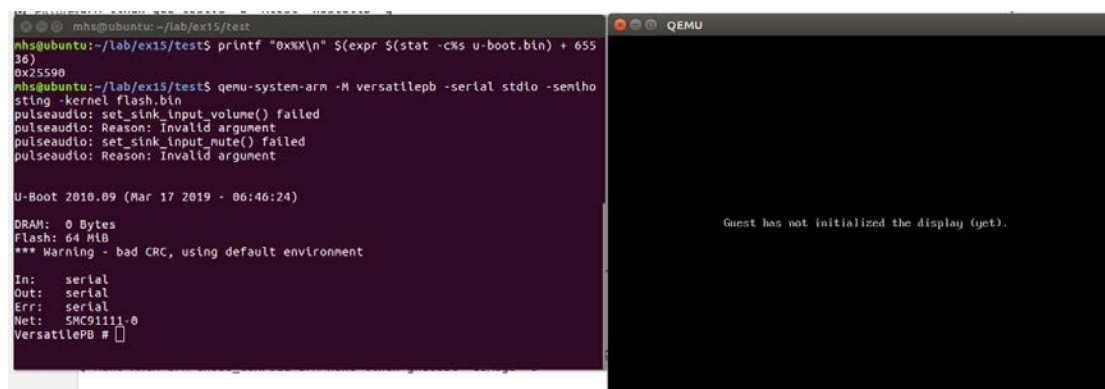


图 8 仿真 U-Boot

在 U-Boot 里面键入: iminfo 0x25590

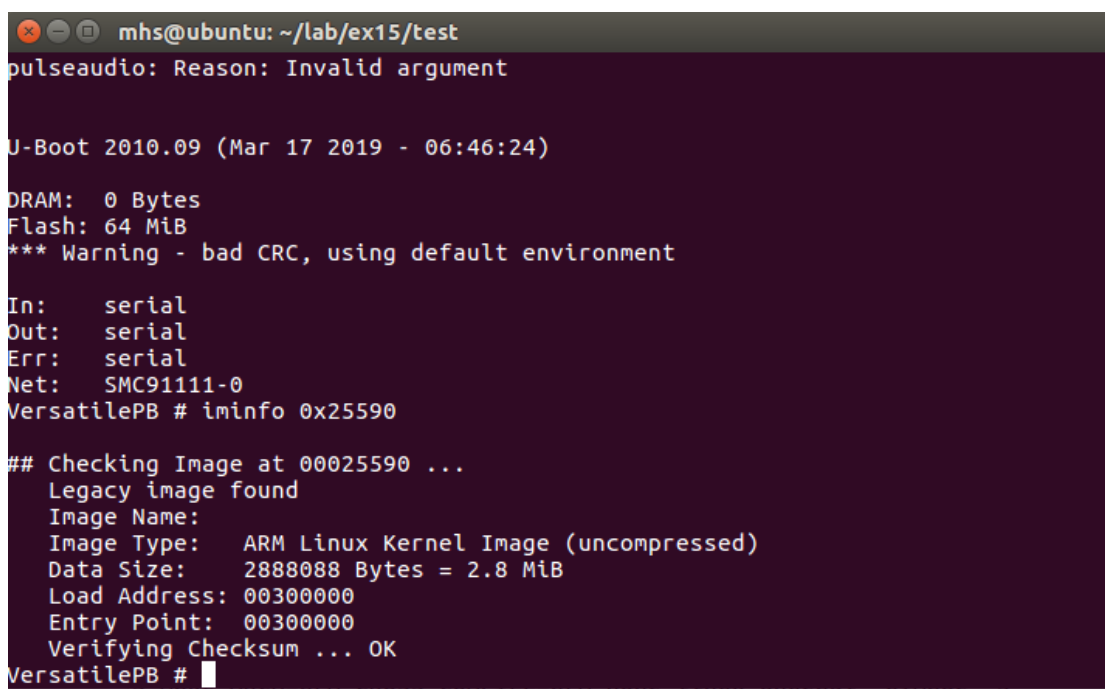


图 9 查看内核文件

可以看到 U-Boot 把我们的 test.bin 看作是 ARM Linux Kernel 的未压缩文件，2.8M，加载到内存入口点是 0x300000，好了，我们用 bootm 命令加载看看吧。

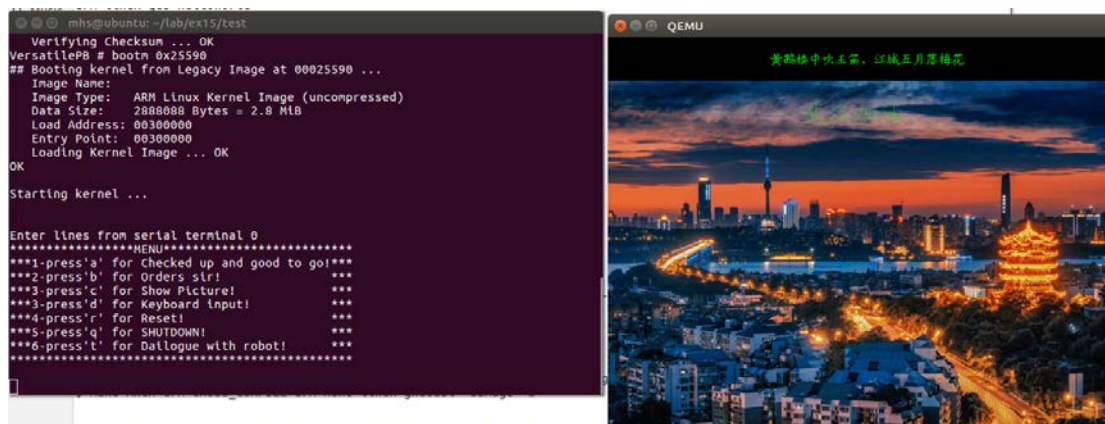


图 10 加载运行我们自己的程序

看到了吧，U-Boot 成功加载了我们前面的程序，运行出来了我们的菜单以及界面，程序所有功能一切正常。

OK，今天的仿真实验暂时就到这里了,下次再见！

注：本实验教程为武汉科技大学机器人与智能系统研究院闵华松老师的网络课程教学文档，可以复制，不做商业用途。