启动流程分析及自制启动镜像

0.背景

本文所属目录层次为：

```
-> 2.系统移植
    -> 1.lichee sdk
        -> 3. 启动流程分析及自制启动镜像
```

主要介绍逆向分析lichee sdk生成镜像的方式，自制镜像以尝试启动系统。
（交流QQ群：573832310，上车口令：爱荔枝）

自制镜像起因

lichee sdk生成的是img文件，是进入FEL后USB下载用的，应该是经过了某种压缩处理。
而前面的构建流程分析里也看到，关键的固件打包部分的程序是二进制可执行文件形式提供的，所以无法或者固件下载到实际存储介质里后的分布情况。
理论上是可以先用FEL下载固件到TF卡，再dump出TF卡内容来分析镜像的组成情况，但是实际操作的时候发现FEL下载总是不明原因失败，所以这种方式暂时不行。
所以为了获得可以运行的系统原始镜像，本节就边分析启动流程，边逆向固件的拼接方式，最后尝试自己构建出原始的系统镜像。

启动**BOOT0**

user manual展示的BROM启动流程图：

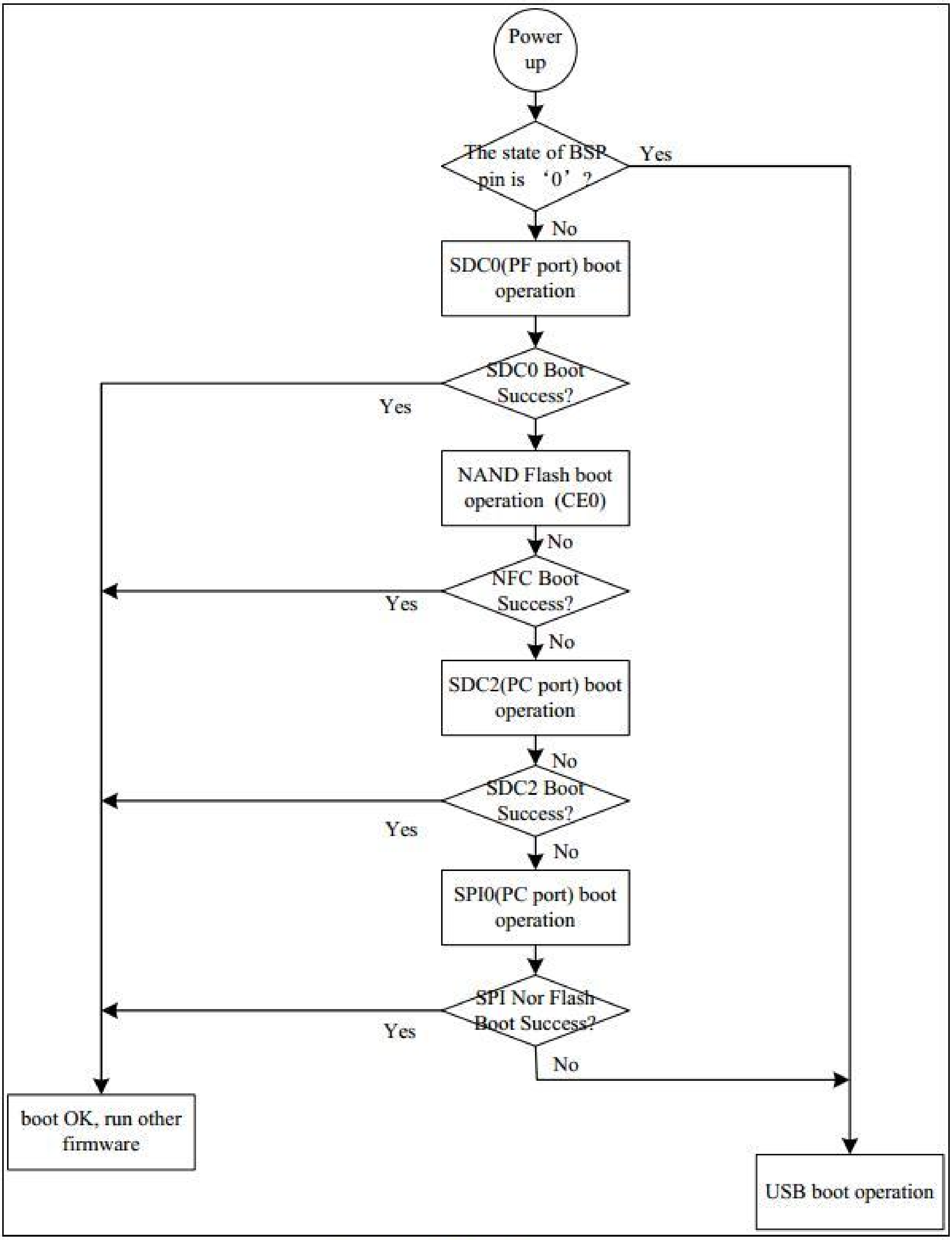
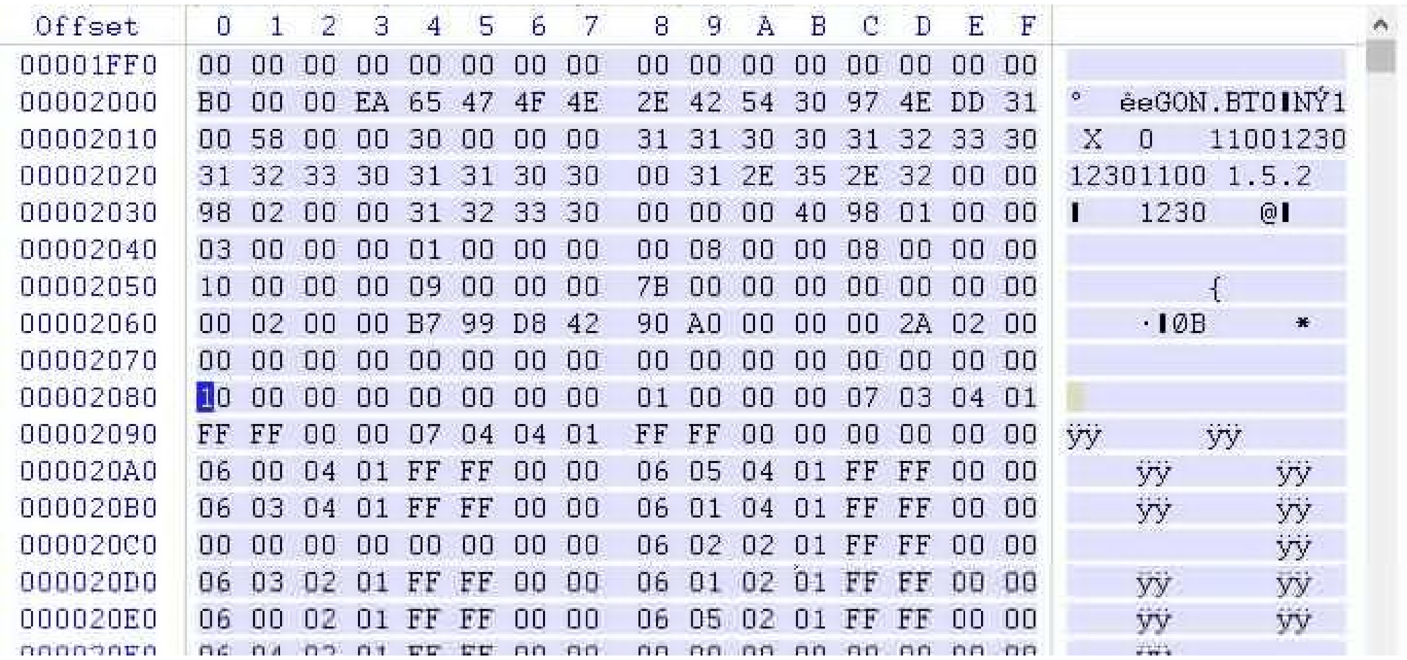

Power up
The state of BSP pin is '0'?
Yes
No
SDC0(PF port) boot operation
SDC0 Boot Success?
Yes
NAND Flash boot operation (CE0)
No
NFC Boot Success?
Yes
No
SDC2(PC port) boot operation
No
SDC2 Boot Success?
Yes
No
SPI0(PC port) boot operation
SPI Nor Flash Boot Success?
Yes
No
boot OK, run other firmware
USB boot operation

Figure 4 Boot Diagram

A13上电启动最先从内部固化的BROM启动，如上图所示，首先会从SD卡的8KB偏移处读取4KB数据到内部SRAM，检验是否是有效的BOOT0，若有效则运行，否则尝试从下一个存储介质读取BOOT0.

所以在打包文件夹下找到card_boot0.fex,使用dd(linux)或者winhex(windows)，把它写入到tf卡的8KB偏移处



```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00001FF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00002000  B0 00 00 EA 65 47 4F 4E  2E 42 54 30 97 4E DD 31  °  êeGON.BT0▮NÝ1
00002010  00 58 00 00 30 00 00 00  31 31 30 30 31 32 33 30   X  0   11001230
00002020  31 32 33 30 31 31 30 30  00 31 2E 35 2E 32 00 00  12301100 1.5.2
00002030  98 02 00 00 31 32 33 30  00 00 00 40 98 01 00 00  ▮   1230   @▮
00002040  03 00 00 00 01 00 00 00  00 08 00 00 08 00 00 00
00002050  10 00 00 00 09 00 00 00  7B 00 00 00 00 00 00 00          {
00002060  00 02 00 00 B7 99 D8 42  90 A0 00 00 00 2A 02 00      ·▮ØB     *
00002070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00002080  10 00 00 00 00 00 00 00  01 00 00 00 07 03 04 01
00002090  FF FF 00 00 07 04 04 01  FF FF 00 00 00 00 00 00  ÿÿ      ÿÿ
000020A0  06 00 04 01 FF FF 00 00  06 05 04 01 FF FF 00 00      ÿÿ      ÿÿ
000020B0  06 03 04 01 FF FF 00 00  06 01 04 01 FF FF 00 00      ÿÿ      ÿÿ
000020C0  00 00 00 00 00 00 00 00  06 02 02 01 FF FF 00 00              ÿÿ
000020D0  06 03 02 01 FF FF 00 00  06 01 02 01 FF FF 00 00      ÿÿ      ÿÿ
000020E0  06 00 02 01 FF FF 00 00  06 05 02 01 FF FF 00 00      ÿÿ      ÿÿ
000020F0  06 04 02 01 FF FF 00 00  00 00 00 00 00 00 00 00      ÿÿ
```

上图中就可以看到BOOT0的magic字符串：eGON.BT0 然后使用 Win32 Disk Imager将做好的包含boot0的镜像写入TF卡

板子插上TF卡，接上UART1的串口，上电，查看启动信息：

```
dram size =512
0xffff0000,0xffff0000
super_standby_flag = 0
HELLO! BOOT0 is starting!
boot0 version : 1.5.2
The size of Boot1 is 0x00038000.
Succeed in loading boot1 from sdmmc flash.
Ready to disable icache.
ERROR! NOT find the head of Boot1.
Fail in loading Boot1.
Jump to Fel.
```

可见BOOT0已经成功启动了，只是BOOT未被找到。

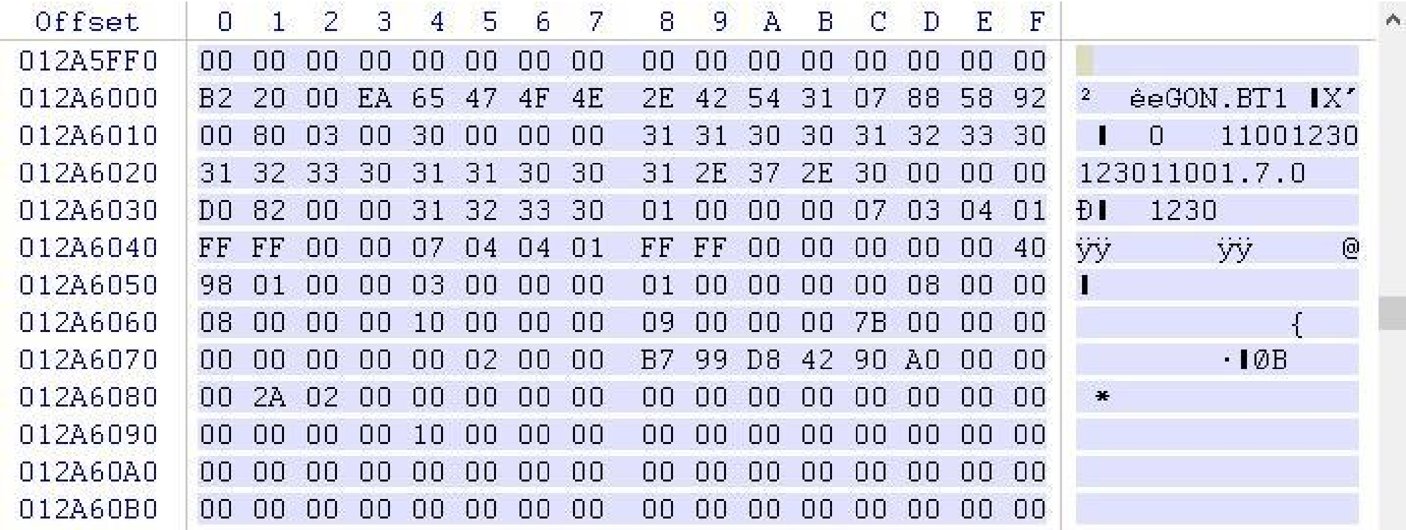
启动BOOT1

根据上面的启动信息，在boot0代码中反查出错位置，发现

```
SDMMC_PhyRead( BOOT1_START_SECTOR_IN_SDMMC, length/512, (void *)BOOT1_BASE, card

#define BOOT1_START_SECTOR_IN_SDMMC    38192
```

所以BOOT1在TF卡中的偏移是固定的38192扇区，即0x12A6000处，约18多MB的偏移。

于是将card_boot1.fex写入0x12A6000偏移处，烧入TF卡中



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
012A5FF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
012A6000  B2 20 00 EA 65 47 4F 4E  2E 42 54 31 07 88 58 92  ²   êeGON.BT1 ▌X’
012A6010  00 80 03 00 30 00 00 00  31 31 30 30 31 32 33 30   ▌  0    11001230
012A6020  31 32 33 30 31 31 30 30  31 2E 37 2E 30 00 00 00  123011001.7.0
012A6030  D0 82 00 00 31 32 33 30  01 00 00 00 07 03 04 01  Ð▌  1230
012A6040  FF FF 00 00 07 04 04 01  FF FF 00 00 00 00 00 40  ÿÿ      ÿÿ     @
012A6050  98 01 00 00 03 00 00 00  01 00 00 00 00 08 00 00  ▌
012A6060  08 00 00 00 10 00 00 00  09 00 00 00 7B 00 00 00              {
012A6070  00 00 00 00 00 02 00 00  B7 99 D8 42 90 A0 00 00          ·▌ØB
012A6080  00 2A 02 00 00 00 00 00  00 00 00 00 00 00 00 00   *
012A6090  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00
012A60A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
012A60B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

上图可见BOOT1的magic字符串：eGON.BT1

板子插上TF卡，接上UART1的串口，上电，查看启动信息：

```
dram size =512
0xffff0000,0xffff0000
super_standby_flag = 0
HELLO! BOOT0 is starting!
boot0 version : 1.5.2
The size of Boot1 is 0x00038000.
Succeed in loading boot1 from sdmmc flash.
Ready to disable icache.
Succeed in loading Boot1.
Jump to Boot1.
[       0.165] boot1 version : 1.7.0
[       0.165] pmu type = 3
[       0.167] bat vol = 311
[       0.199] axi:ahb:apb=3:2:2
[       0.199] set dcdc2=1400, clock=1008 successed
[       0.201] key
[       0.213] no key found
[       0.213] flash init start
[       0.231] flash init finish
[       0.232] fs init fail
[       0.232] fs init fail, jump to fel
```

可见BOOT0已经成功引导起来了BOOT1，BOOT1已经简单设置了电压，时钟频率，初始化了tf卡
只是还没有找到bootfs的文件系统，无法引导uboot

MBR及BOOTFS挂载

由前面的构建流程分析可知bootfs的打包文件即**bootloader.fex**，现在要确认该文件所处的tf卡偏移。 eFG_printf; 为了获取出错信息，需要在boot1中的代码里加打印信息来追踪。

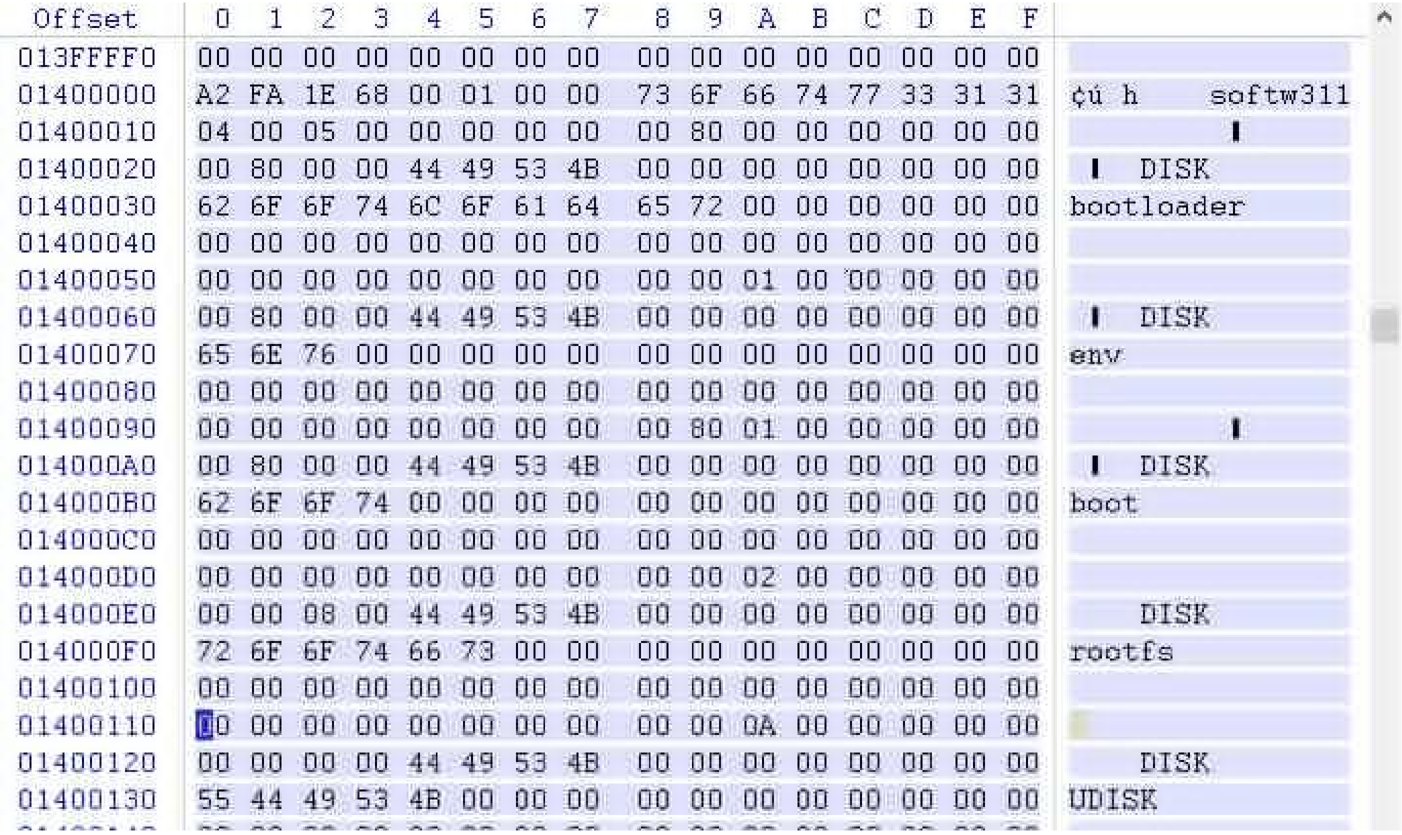
> 这里注意，在boot目录下编译后，还需要运行一次pack命令才能将原始的bin文件加入fex文件中的配置信息，才能运行起来，最后烧录的应该是card_boot1.fex。

追踪fs init fail的原因，发现是读取MBR的magic字符串softw311出错 查找out文件夹，**mbr.fex**文件应该就是。 按之前的常识MBR应该是放在硬盘的0偏移处？但实际试了却发现仍然读不到。 （MBR是Master Boot Record的简称,又叫主引导记录.它是硬盘上最重要的一个数据结构。当用分区软件创建分区的时候,分区软件会自动创建MBR.MBR处于硬盘的第一个扇区.即0柱面,0磁头,1扇区.在总共512byte的主引导记录中，MBR的引导程序占了其中的前446个字节(偏移0H~偏移1BDH)，随后的64个字节(偏移1BEH~偏移1FDH)为DPT(Disk PartitionTable，硬盘分区表)，最后的两个字节"55 AA"(偏移1FEH~偏移1FFH)是分区有效结束标志。）

追踪读取函数，发现这里使用的是SDMMC_LogicalRead，逻辑读相对物理读会有个偏移，

```
	start_sector += bootcard_offset;
```

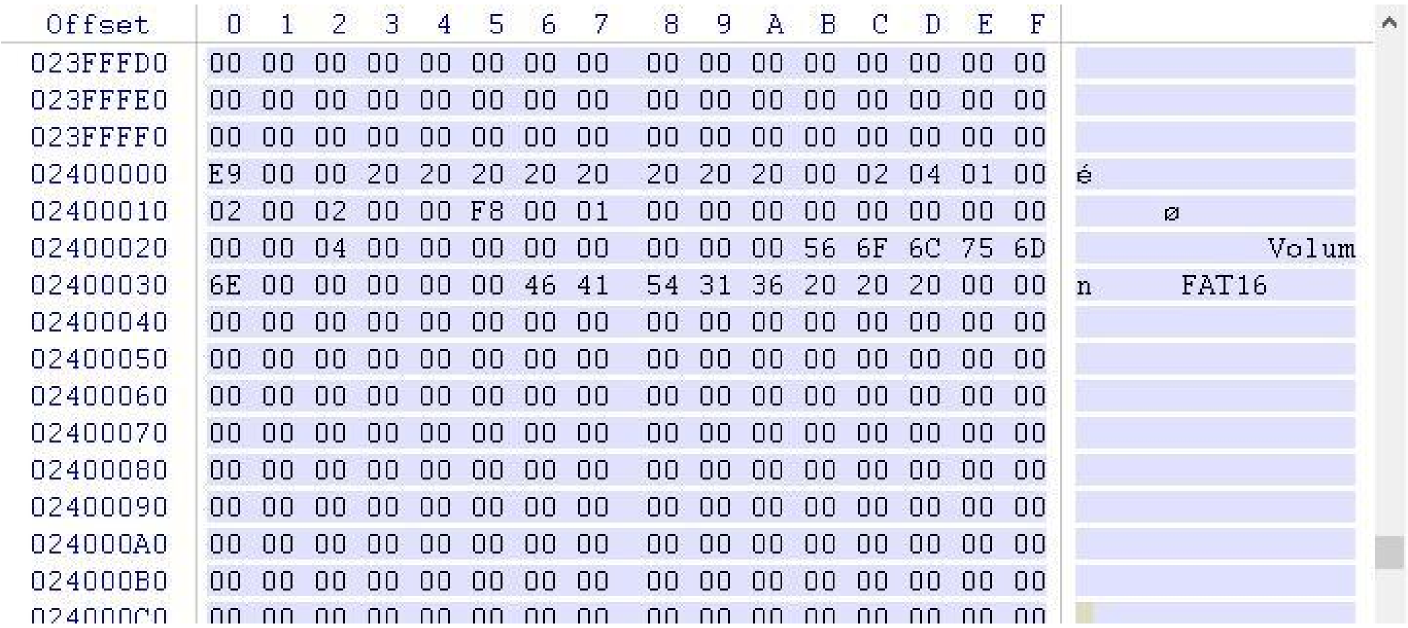
而这个bootcard_offset信息就包含在BOOT1的头部信息里，直接打印出来发现是0xa000,那么就是0x5000 KB = 20MB偏移处 于是将mbr.fex写入该偏移：



```
 Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
013FFFF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01400000  A2 FA 1E 68 00 01 00 00  73 6F 66 74 77 33 31 31  ¢ú h    softw311
01400010  04 00 05 00 00 00 00 00  00 80 00 00 00 00 00 00           ▮
01400020  00 80 00 00 44 49 53 4B  00 00 00 00 00 00 00 00   ▮  DISK
01400030  62 6F 6F 74 6C 6F 61 64  65 72 00 00 00 00 00 00  bootloader
01400040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01400050  00 00 00 00 00 00 00 00  00 00 01 00 00 00 00 00
01400060  00 80 00 00 44 49 53 4B  00 00 00 00 00 00 00 00   ▮  DISK
01400070  65 6E 76 00 00 00 00 00  00 00 00 00 00 00 00 00  env
01400080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01400090  00 00 00 00 00 00 00 00  00 80 01 00 00 00 00 00           ▮
014000A0  00 80 00 00 44 49 53 4B  00 00 00 00 00 00 00 00   ▮  DISK
014000B0  62 6F 6F 74 00 00 00 00  00 00 00 00 00 00 00 00  boot
014000C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
014000D0  00 00 00 00 00 00 00 00  00 00 02 00 00 00 00 00
014000E0  00 00 08 00 44 49 53 4B  00 00 00 00 00 00 00 00      DISK
014000F0  72 6F 6F 74 66 73 00 00  00 00 00 00 00 00 00 00  rootfs
01400100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
01400110  00 00 00 00 00 00 00 00  00 00 0A 00 00 00 00 00
01400120  00 00 00 00 44 49 53 4B  00 00 00 00 00 00 00 00      DISK
01400130  55 44 49 53 4B 00 00 00  00 00 00 00 00 00 00 00  UDISK
```

上图可见mbr的magic字符串：softw311

然后根据sys_config.fex和image.cfg里的四个分区的说明，在相对MBR的16MB偏移处写入**bootloader.fex**



```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
023FFFD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
023FFFE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
023FFFF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400000  E9 00 00 20 20 20 20 20  20 20 20 00 02 04 01 00  é
02400010  02 00 02 00 00 F8 00 01  00 00 00 00 00 00 00 00       ø
02400020  00 00 04 00 00 00 00 00  00 00 00 56 6F 6C 75 6D             Volum
02400030  6E 00 00 00 00 00 46 41  54 31 36 20 20 20 00 00  n     FAT16
02400040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
024000A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
024000B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
024000C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

上图可见bootfs是FAT16文件系统

更新tf卡内容，板子插上TF卡，接上UART1的串口，上电，查看启动信息：

```
[       0.163] boot1 version : 1.7.0
[       0.163] pmu type = 3
[       0.164] bat vol = 2
[       0.196] axi:ahb:apb=3:2:2
[       0.196] set dcdc2=1400, clock=1008 successed
[       0.198] key
[       0.211] no key found
[       0.211] flash init start
[       0.281] flash init finish   //tf卡初始化成功
[       0.281] begin init!
[       0.282] magic is softw311   //mbr识别成功
[       0.284] fs init ok
[       0.287] fattype FAT16
[       0.289] fs mount ok         //文件系统挂载成功
[       0.296] script finish       //script.bin解析成功
[       0.297] power finish        //按fex中的设置调整电压成功
[       0.297] storage_type=1      //获取存储类型，card0启动(0nand,1sdc0,2sdc2)
[       0.307] BootMain start      //进入Boot_Andriod分支，apps/Boot_Android/BootM
[       0.307] 0          //从串口读键值，2fel，-dbg，1usb，3lradc
[       0.340] init to usb pc
[       0.364] power_start=0x00000002
[       0.364] power trigger
[       0.364] startup status = 0
[       0.384] parser bmp file c:\os_show\bat0.bmp failed  //wboot_fopen失败？
[       1.885] no battery exist
[       1.885] key value = 1
[       1.885] recovery key high 6, low 4
[       1.889] unable to find fastboot_key key_max value
[       1.910] unable to open script file c:\linux\linux.ini
[       1.910] NO OS to Boot
[       1.917] try to fel in 1 secend
```

可见BOOT1已经成功挂载了BOOTFS，并从其文件系统中的script.bin获取到了系统配置信息，进一步配置了PMU，并执行eGon2_run_app,运行了**boot.axf**（Boot_Andriod），试图启动uboot。 只是在Boot_Andriod里，不知为何又读取文件系统出错了？

启动UBOOT

注意到boot.axf虽然是boot1目录下的源码生成的，但是实际是存放在bootfs中的可执行文件，所以下面需要查找bootloader.fex这个分区镜像的问题。

仔细追查发现，在bootfs的根目录下文件可以被访问到，但是二级目录下的文件却总是读取失败。 不由得怀疑是不是二级目录根本没打包进去。

```
sudo mount bootloader.fex mnt/
ls mnt/
```

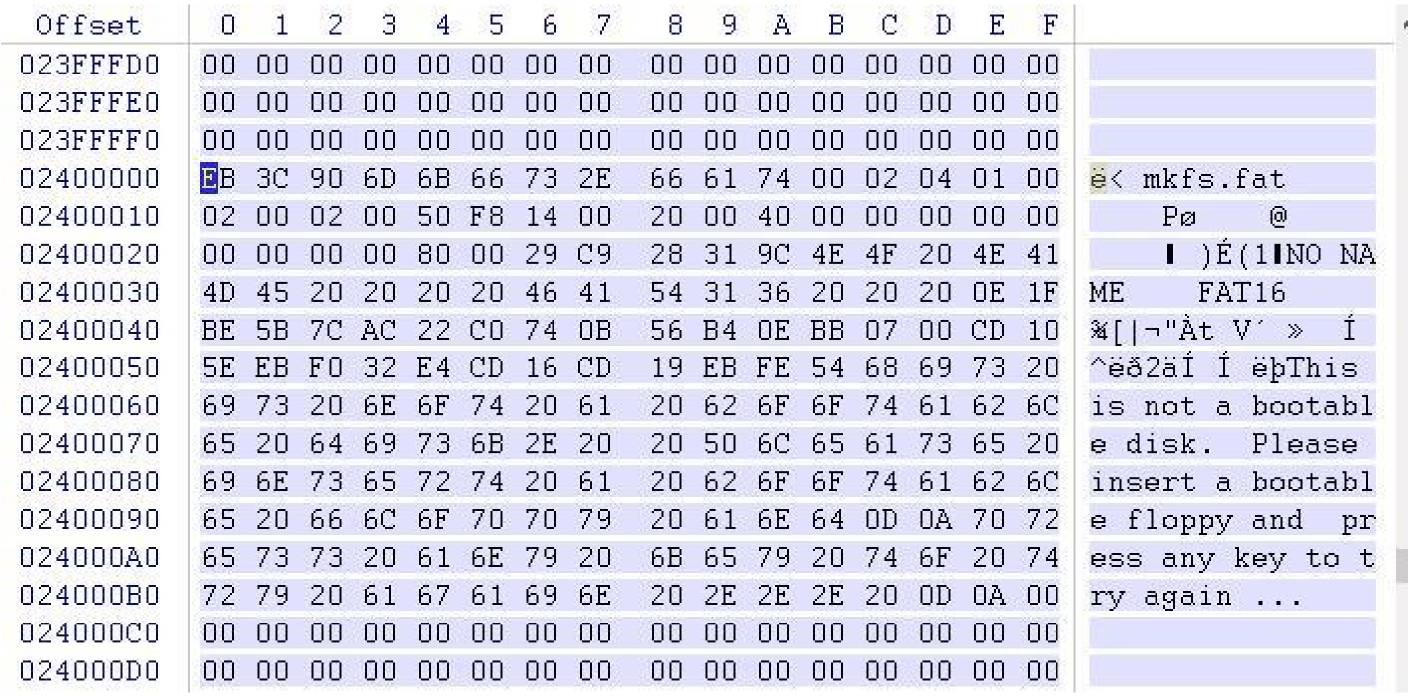
挂载该分区查看，果然实际上二级目录就是没有被打包进去 仔细查看编译时的输出发现：

```
fail:/home/zp/develop/a13_android4.2_v1.5.0/v1.5.0/lichee/tools/pack/out/bootfs/
```

以上是执行`fsbuild bootfs.ini split_xxxx.fex`时的错误信息，只是该错误不会使整个编译过程退出。 由于fsbuild是可执行文件，没有源代码，错误提示也语焉不详，所以暂时无法知道到底是什么原因导致二级目录没有被打包进去（如果以后查出原因再更新解决方法）。 所以现在先尝试手工打包bootfs。

```
dd if=/dev/zero of=bootloader.fex count=20480   //10MB 分区
mkdir mnt
sudo mount bootloader.fex mnt
sudo cp -r bootfs/* mnt/
sudo umount mnt
```

将上面手工打包的bootfs镜像烧录到sd卡36MB偏移（0x2400000）处：



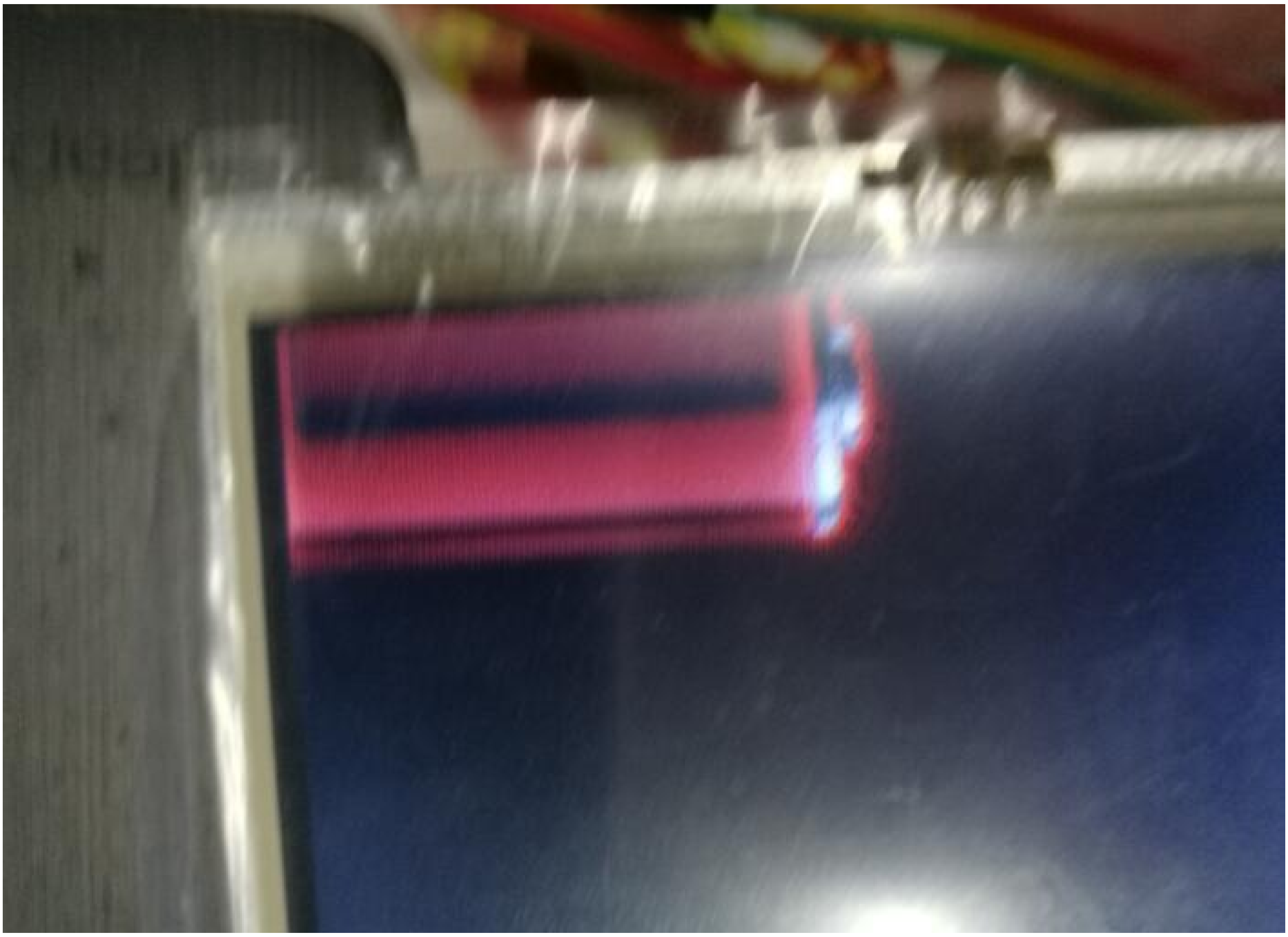
```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
023FFFD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
023FFFE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
023FFFF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
02400000  EB 3C 90 6D 6B 66 73 2E  66 61 74 00 02 04 01 00  ë< mkfs.fat
02400010  02 00 02 00 50 F8 14 00  20 00 40 00 00 00 00 00     Pø    @
02400020  00 00 00 00 80 00 29 C9  28 31 9C 4E 4F 20 4E 41      ▌ )É(1▌NO NA
02400030  4D 45 20 20 20 20 46 41  54 31 36 20 20 20 0E 1F  ME    FAT16
02400040  BE 5B 7C AC 22 C0 74 0B  56 B4 0E BB 07 00 CD 10  ¾[|¬"Àt V´ »  Í
02400050  5E EB F0 32 E4 CD 16 CD  19 EB FE 54 68 69 73 20  ^ëð2äÍ Í ëþThis
02400060  69 73 20 6E 6F 74 20 61  20 62 6F 6F 74 61 62 6C  is not a bootabl
02400070  65 20 64 69 73 6B 2E 20  20 50 6C 65 61 73 65 20  e disk.  Please
02400080  69 6E 73 65 72 74 20 61  20 62 6F 6F 74 61 62 6C  insert a bootabl
02400090  65 20 66 6C 6F 70 70 79  20 61 6E 64 0D 0A 70 72  e floppy and  pr
024000A0  65 73 73 20 61 6E 79 20  6B 65 79 20 74 6F 20 74  ess any key to t
024000B0  72 79 20 61 67 61 69 6E  20 2E 2E 2E 20 0D 0A 00  ry again ...
024000C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
024000D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

更新tf卡内容，板子插上TF卡，接上UART1的串口，上电，查看启动信息：

```
[       0.163] boot1 version : 1.7.0
[       0.163] pmu type = 3
[       0.164] bat vol = 2
[       0.196] axi:ahb:apb=3:2:2
[       0.196] set dcdc2=1400, clock=1008 successed
[       0.198] key
[       0.211] no key found
[       0.211] flash init start
[       0.229] flash init finish
[       0.229] begin init!
[       0.230] magic is softw311
[       0.232] fs init ok
[       0.235] fattype FAT16
[       0.237] fs mount ok
[       0.243] script finish
[       0.244] power finish
[       0.244] storage_type=1
[       0.256] BootMain start
[       0.256] 0
[       0.288] init to usb pc
[       0.310] power_start=0x00000002
[       0.311] power trigger
[       0.311] startup status = 0
[       2.387] no battery exist
[       2.387] key value = 1
[       2.387] recovery key high 6, low 4
[       2.391] unable to find fastboot_key key_max value
[       2.399] test for multi os boot with display
[       2.461] show pic finish
[       2.461] load kernel start
[       2.479] load kernel successed
[       2.479] star

U-Boot 2011.09-rc1 (Jul 29 2016 - 20:20:40) Allwinner Technology

CPU:   SUNXI Family
Board: A1X-EVB
DRAM:  512 MiB
MMC:   SUNXI SD/MMC: 0
In:    serial
Out:   serial
Err:   serial
--------fastboot partitions--------
-total partitions:5-
-name-        -start-       -size-
bootloader  : 1000000       1000000
env         : 2000000       1000000
boot        : 3000000       1000000
rootfs      : 4000000       10000000
UDISK       : 14000000      90c00000
-----------------------------------
no misc partition is found
Hit any key to stop autoboot:  0
read boot or recovery all
sunxi flash read :offset 3000000, 12824208 bytes OK

Starting kernel ...

[    0.000000] ram_console: buffer   (null), invalid size 0, datasize 4294967284
```

由上可见，成功读取了系统信息，加载了Uboot
甚至Uboot也成功走完，进入了linux kernel的启动阶段 只是kernel启动开始时就出现了console初始化的错误，看上去像是参数传递错误。

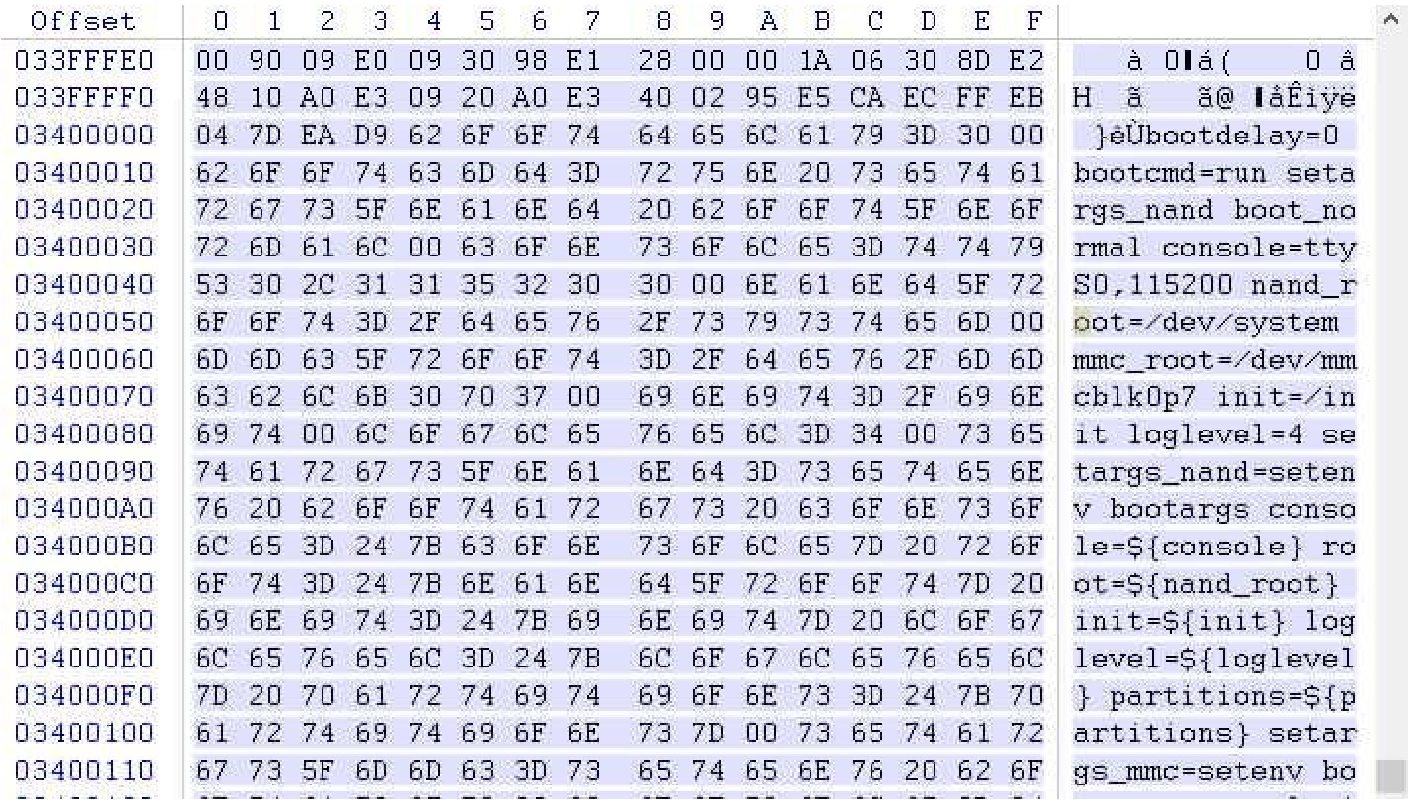
此时若接上屏幕，可以看到电池图标和andriod图标（屏幕参数尚未调整，显示不完全）：



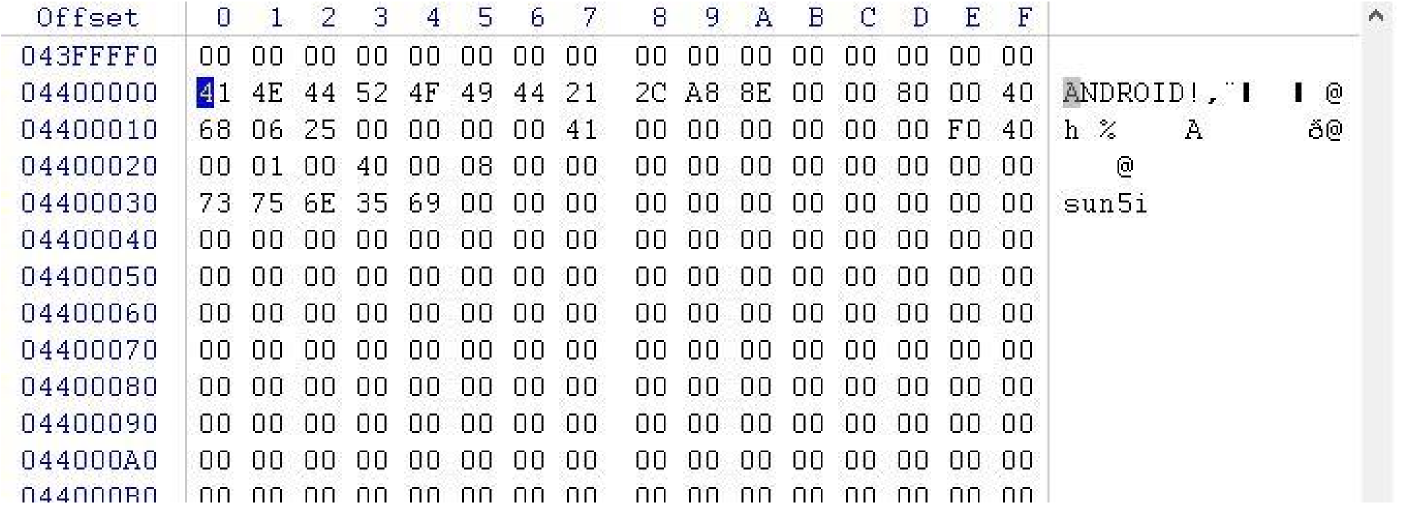
启动linux内核

由于前面的出错提示信息比较少，所以先修改linux的内核调试等级到最详细看看,在linux-3.0目

录下执行`make menuconfig ARCH=arm`进行配置。把内核调试等级调整到7后，重新烧录boot.fex到0x4400000偏移处 但是发现调试信息并没有增多，仔细查看发现uboot启动内核的env.cfg中也有调试等级的设置，还有一些其它板级设置，一并修改看看（tools\pack\chips\sun5i\configs\dragonboard\default\env.cfg），修改后会重新生成env.fex，需要烧录到0x34000000处



```
 Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
033FFFE0  00 90 09 E0 09 30 98 E1  28 00 00 1A 06 30 8D E2     à 0▌á(    0 â
033FFFF0  48 10 A0 E3 09 20 A0 E3  40 02 95 E5 CA EC FF EB  H  ã   ã@ ▌åÊìÿë
03400000  04 7D EA D9 62 6F 6F 74  64 65 6C 61 79 3D 30 00   }êÙbootdelay=0
03400010  62 6F 6F 74 63 6D 64 3D  72 75 6E 20 73 65 74 61  bootcmd=run seta
03400020  72 67 73 5F 6E 61 6E 64  20 62 6F 6F 74 5F 6E 6F  rgs_nand boot_no
03400030  72 6D 61 6C 00 63 6F 6E  73 6F 6C 65 3D 74 74 79  rmal console=tty
03400040  53 30 2C 31 31 35 32 30  30 00 6E 61 6E 64 5F 72  S0,115200 nand_r
03400050  6F 6F 74 3D 2F 64 65 76  2F 73 79 73 74 65 6D 00  oot=/dev/system
03400060  6D 6D 63 5F 72 6F 6F 74  3D 2F 64 65 76 2F 6D 6D  mmc_root=/dev/mm
03400070  63 62 6C 6B 30 70 37 00  69 6E 69 74 3D 2F 69 6E  cblk0p7 init=/in
03400080  69 74 00 6C 6F 67 6C 65  76 65 6C 3D 34 00 73 65  it loglevel=4 se
03400090  74 61 72 67 73 5F 6E 61  6E 64 3D 73 65 74 65 6E  targs_nand=seten
034000A0  76 20 62 6F 6F 74 61 72  67 73 20 63 6F 6E 73 6F  v bootargs conso
034000B0  6C 65 3D 24 7B 63 6F 6E  73 6F 6C 65 7D 20 72 6F  le=${console} ro
034000C0  6F 74 3D 24 7B 6E 61 6E  64 5F 72 6F 6F 74 7D 20  ot=${nand_root}
034000D0  69 6E 69 74 3D 24 7B 69  6E 69 74 7D 20 6C 6F 67  init=${init} log
034000E0  6C 65 76 65 6C 3D 24 7B  6C 6F 67 6C 65 76 65 6C  level=${loglevel
034000F0  7D 20 70 61 72 74 69 74  69 6F 6E 73 3D 24 7B 70  } partitions=${p
03400100  61 72 74 69 74 69 6F 6E  73 7D 00 73 65 74 61 72  artitions} setar
03400110  67 73 5F 6D 6D 63 3D 73  65 74 65 6E 76 20 62 6F  gs_mmc=setenv bo
```



```
 Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
043FFFF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400000  41 4E 44 52 4F 49 44 21  2C A8 8E 00 00 80 00 40  ANDROID!,¨▌   ▌ @
04400010  68 06 25 00 00 00 00 41  00 00 00 00 00 00 F0 40  h %    A       ð@
04400020  00 01 00 40 00 08 00 00  00 00 00 00 00 00 00 00     @
04400030  73 75 6E 35 69 00 00 00  00 00 00 00 00 00 00 00  sun5i
04400040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
04400090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
044000A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
044000B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

修改后可以查看到内核实际收到的启动参数

```
[    0.000000] Linux version 3.0.8+ (zp@ubuntu) (gcc version 4.5.1 (Sourcery G++
[    0.000000] Kernel command line: console=ttyS0,115200 root=/dev/mmcblk0p7 ini
```

其中partitions参数是在u-boot/Board/Allwinner/A1x-evb/A1x-evb.c里生成，是没有问题的。

继续查看下面，有问题的是：

```
[    1.434633] mmc0: new high speed SDHC card at address 1234
[    1.440523] mmcblk0: mmc0:1234 SA16G 14.5 GiB
[    1.447031]  mmcblk0: unknown partition table
```

这说明是分区表错误使得启动失败。 想想之前对tf卡的操作的确没有写入分区表，所以会挂载失败。 MBR是Master Boot Record的简称,又叫主引导记录.它是硬盘上最重要的一个数据结构。当用分区软件创建分区的时候,分区软件会自动创建MBR.MBR处于硬盘的第一个扇区.即0柱面,0磁头,1扇区.在总共512byte的主引导记录中，MBR的引导程序占了其中的前446个字节(偏移0H~偏移1BDH)，随后的64个字节(偏移1BEH~偏移1FDH)为DPT(Disk PartitionTable，硬盘分区表)，最后的两个字节"55 AA"(偏移1FEH~偏移1FFH)是分区有效结束标志。

分区表由4项组成，每项16个字节.共4×16 = 64个字节．每项描述一个分区的基本信息.每个字节的含义如下: 分区表项含义

|字节|含义| |--|--| |0|Activeflag.活动标志.若为0x80H,则表示该分区为活动分区.若为0x00H,则表示该分区为非活动分区| |1,2,3|该分区的起始磁头号,扇区号,柱面号磁头号 -- 1字节, 扇区号 -- 2字节低6位,柱面号-- 2字节高2位 + 3字节| |4|分区文件系统标志：分区未用: 0x00H. 扩展分区: 0x05H, 0x0FH. FAT16分区: 0x06H. FAT32分区: 0x0BH, 0x1BH, 0x0CH, 0x1CH. NTFS分区: 0x07H.| |5,6,7|该分区的结束磁头号,扇区号,柱面号，含义同上.| |8,9,10,11|逻辑起始扇区号。表示分区起点之前已用了的扇区数| |12,13,14,15|该分区所占用的扇区数|