Lite200开发指南

Lite200 Development Guide

作者：leesans (LiShanwen)

时间：2021年2月5日

版本：3.1

协议：GPL2.0

ElegantLaTeX Program

温柔正确的人总是难以生存，因为这世界既不温柔，也不正确。—— 比企谷八幡

特别声明

回首过去几年，有种莫名的失败感，

Ethan Deng

February 10, 2020

目录

第一章 Linux基础知识

1.1 Linux思想：一切皆文件

1.2 虚拟机使用

1.3 常用命令

1.4 FTP服务

1.5 shell

1.6 Makefile

1.7 vim使用

第二章 系统移植

内容提要

- 环境搭建
- u-boot移植
- linux移植
- rootfs移植
- 深入探究

本章分三个部分来介绍Linux操作系统的移植，分别是bootloader，kernel，rootfs。其中bootloader使用最多的是u-boot，本书将使用u-boot来移植。kernel是Linux内核部分，rootfs是根文件系统。如图2.1是这三者的结构图，bootloader是SoC芯片最开始执行的代码，这

图 2.1

个部分的主要工作是进行硬件初始化，例如时钟初始化，SRAM 初始化等，同时也为后面运行C语言建立环境(初始化堆栈)。kernel在存放在bootloader之后，对于SoC来说，代码都需要在RAM中运行，这里与MCU不一样的地方就是引入了MMU（内存管理单元）。对于MCU而言，由于其执行速度低，因此运行代码都在ROM中直接运行，而对于Flash而言，其读取速度远不及RAM的速度，因此对于运行速度非常快的SoC而言，所有的代码都需要在RAM中运行。但是这里有一个问题，RAM掉电数据将会丢失，故代码保存不可能放在RAM中，当前所有的嵌入式设备而言，代码保存都是放在ROM中，因此在SoC中运行代码需要将代码搬运到RAM中然后再执行。对于bootloader来说。其代码较少，很多开发者认为再将bootloader搬到RAM中实际意义不大，除非bootloader的体积很大。实际上在u-boot 中，全志公司提供的代码是直接在ROM中运行的，当u-boot运行完大部分后会将kernel搬运到指定的RAM位置，之后将启动内核，这个启动过程后面会详细说明。对于根文件系统（rootfs)而言，由于其执行过程需要对ROM进行读写操作，因此可以不用搬运到RAM中，但是实际过程中内核启动后会产生一个虚拟的文件系统，该文件系统是挂在根文件系统的关键所在，这里不详细讲解。整体来说，大致的过程为，嵌入式设备上电后将执行bootloader，对硬件进行硬件和堆栈初始化，然后搬运内核到RAM中并启动内核，紧接着挂载根文件系统。

⚠ 现在绝大多数SoC内部有一段固化的代码，该代码放在bootloader之前，也就是IROM，故实际上最开始执行的也并不是bootloader，而是芯片内部的IROM。IROM的主要目的是方便芯片的操作，例如通过USB对Flash芯片进行烧写程序。

2.1 环境搭建

2.1.1 交叉工具链安装

所有的嵌入式设备开发都会用到各种工具，由于每个嵌入式SoC的芯片架构不同，其开发工具也不尽相同。当前用的最多的是ARM，本书中使用的Lite200主芯片的内核为ARM9，其架构使用的是ARMv5架构。最主要的工具为交叉工具链，对于F1C200S，使用的交叉工具链必须高于6.0版本，本书编译u-boot和kernel使用7.2.1版本，连接：arm-linux-gnueabi-7.2 点击即可下载。

> 下载速度慢可以使用迅雷等下载软件进行下载。

下载完成后解压文件：

tar -vxjf gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz

然后在/usr/local目录下新建arm-linux-gcc目录

sudo mkdir /usr/local/arm-linux-gcc

进入解压目录下：

cd gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/

将该目录下的所有文件复制到新建的目录下

sudo cp -rd * /usr/local/arm-linux-gcc/

最后需要添加该工具链的环境变量使其可以在任何目录下执行，打开/etc/profile文件

sudo vim /etc/profile

在文件末尾添加以下内容

PATH=$PATH:/usr/local/arm-linux-gcc/bin

添加完毕，使路径生效

source /etc/profile

笔记 arm-linux-gnueabi与arm-linux-gnueabihf的区别在于前者可以编译无浮点运算单元的SoC芯片，而后者只能只能编译带浮点运算单元的芯片，由于全志F1C200S内部没有浮点运算单元，因此这里必须安装前者。对于全志V3s或者H3芯片，其内部有浮点运算单元，故应该安装后者。

> 上述也可以修改/etc/environment文件，在PATH值追加：
> :/usr/local/arm-linux-gcc/bin"

下面验证交叉工具链是否成功安装：

在任何目录中输入arm-linux-，然后连按两次Tab键，如果出现补全交叉工具链名称，则说明安装成功，如2.2

如果没有出现，则进行下面操作:

安装必要的动态链接库

sudo apt-get install lib32ncurses5 lib32z1

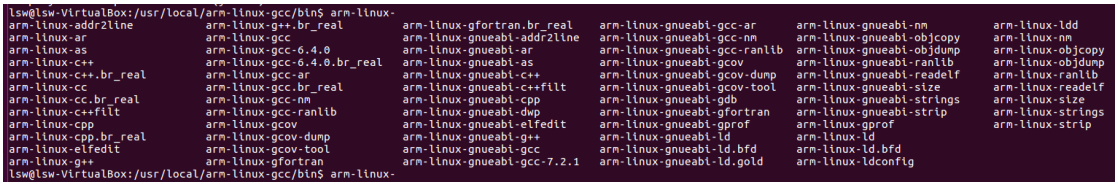

图 2.2

上面安装的动态链接库主要是因为ubuntu是64位的操作系统，而交叉工具链是32位的，因此我们需要安装 32位必须的一些动态链接库。

2.1.2 安装下载工具 (sunxi-tools)

⚠ 该部分只针对SPI_FLASH方式启动的用户而言，使用TF卡启动可以忽略此小结

因为后面会使用git工具，所以我们先要安装git工具，这个工具的目的就是从github上下载或者称之为克隆代码用的。使用如下命令安装git工具：

sudo apt-get install git

然后我们下载我们需要使用的下载工具（sunxi-tools):

git clone https://github.com/linux-sunxi/sunxi-tools.git -b f1c100s-spiflash

克隆下来的工具将保存在当前目录下，进入该目录后，执行:

make

然后安装即可：

make install

⚠ 如果出现错误，这是因为有些必要的依赖库没有安装，执行如下命令即可安装：

sudo apt-get install libusb-1.0-0-dev zlib1g-dev

若出现无法下载的情况，请先下载该库，然后手动安装即可，步骤如下：

wget http://packages.deepin.com/deepin/pool/main/libu/libusb-1.0/libusb-1.0-0-dev_1.0.21-1_amd64.deb

下载完毕后使用如下指令安装该库：

sudo dpkg -i libusb-1.0-0-dev_1.0.21-1_amd64.deb

安装完毕后需要验证是否安装成功，在终端输入： **sunxi-fel**

如果有如下输出则说明安装成功。

2.2 u-boot移植（TF卡启动）

一般来说u-boot是启动内核的关键所在，当前大部分使用的嵌入式设备都是u-boot，最新版本的u-boot几乎包含当前主流的SoC芯片，Lite200使用的芯片和licheePI nano相同，大部分硬件也是兼容的，为了快速移植该部分，这里采用licheePI nano的u-boot来进行移植。在终端输入如下命令克隆u-boot：

```
admir@admir-PC:~/licheePI/sunxi-tools$ sunxi-fel
sunxi-fel v1.4.1-104-g11a9d20

Usage: sunxi-fel [options] command arguments... [command...]
       -h, --help                    Print this usage summary and exit
       -v, --verbose                 Verbose logging
       -p, --progress                "write" transfers show a progress bar
       -l, --list                    Enumerate all (USB) FEL devices and exit
       -d, --dev bus:devnum          Use specific USB bus and device number
           --sid SID                 Select device by SID key (exact match)

       spl file                      Load and execute U-Boot SPL
              If file additionally contains a main U-Boot binary
              (u-boot-sunxi-with-spl.bin), this command also transfers that
              to memory (default address from image), but won't execute it.

       uboot file-with-spl           like "spl", but actually starts U-Boot
              U-Boot execution will take place when the fel utility exits.
              This allows combining "uboot" with further "write" commands
              (to transfer other files needed for the boot).
```

图 2.3

git clone https://github.com/Lichee-Pi/u-boot.git -b nano-v2018.01

克隆完毕文件会保存在当前目录下，进入该目录，

cd u-boot

在该文件夹下有很多分支，我们可以查看所有分支，使用如下命令：

git branch -a

现在我们使用的是nano开发板，所以将当前分支切换到nano分支，命令如下：

git checkout nano-v2018.01

u-boot默认的没有指定交叉工具链和架构，因此在编译之前需要指定交叉工具链和芯片架构，u-boot的交叉编译器在u-boot 的根目录下中的Makefile文件中定义了。打开Makefile文件：

vim Makefile

找到CROSS_COMPILE变量，将其改为如下：

ARCH=arm

CROSS_COMPILE=arm-linux-gnueabi-

如2.5所示。 config目录下是板级配置文件，由于每个板子的外设不同，因此编译之前必

```
242 ######################################################################
243
244 # set default to nothing for native builds
245 ARCH?=arm
246 CROSS_COMPILE ?=arm-linux-gnueabi-
247
248 KCONFIG_CONFIG  ?= .config
249 export KCONFIG_CONFIG
250
251 # SHELL used by kbuild
252 CONFIG_SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; \
253           else if [ -x /bin/bash ]; then echo /bin/bash; \
254           else echo sh; fi ; fi)
255
```

图 2.4

须要对u-boot进行配置。然而配置是一件比较繁琐的事情，特别是像u-boot这种比较复杂的项目而言，初学者几乎无法完成。幸运的是对于大部分开发板而言，config目录下有其配置好的默认配置文件。进入config目录中，然后执行ls查看当前所有的配置文件

cd config

ls

找到licheepi_nano_defconfig和licheepi_nano_spiflash_defconfig，前者表示为TF卡启动，后者表示从SPI 设备启动，这里有前者即可，如图2.5所示。 现在回到上级目录，然后执

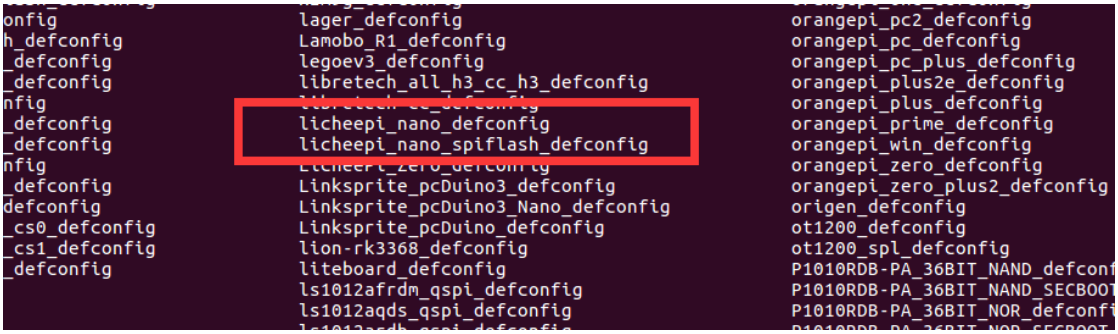

图 2.5

行make licheepi_nano_defconfig

cd ..

make make licheepi_nano_defconfig

配置完成后就可以进入图形界面进行配置了，执行make menuconfig命令：

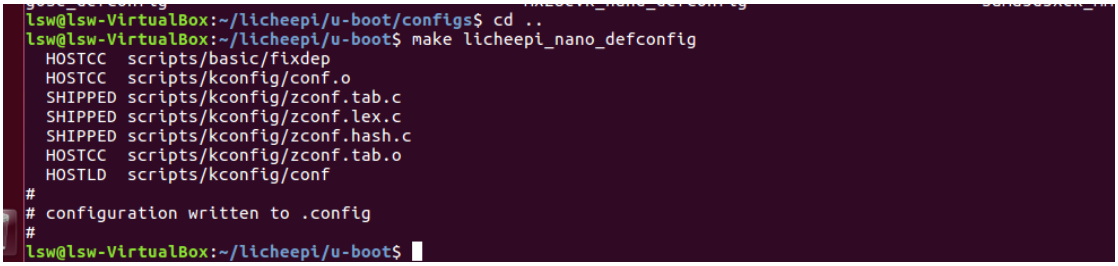

图 2.6

make menuconfig

此时出现图形配置选项，如图2.7 图形配置选项中有两个非常重要的参数配置–传参，下

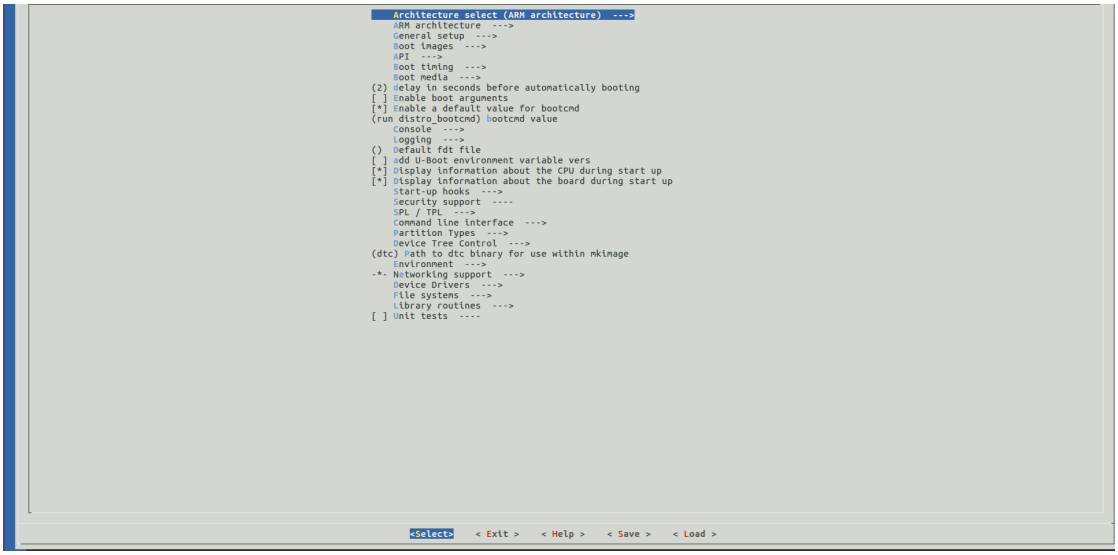

图 2.7

小结重点讲解该部分。

2.2.1 传参

如图2.1所示，其中parameter就是传参需要的参数地址。这里不得不提bootargs和bootcmd，这两个环境变量可以说是所有环境变量中最重要的变量，读者看完之后就会有体会。

2.2.1.1 bootcmd

在最开始提到过，内核一般不在flash中运行，这样就需要将内核搬运到内存中，这个过程需要u-boot来完成。对于mmc (TF卡)而言，在u-boot有专门的命令load mmc，该命令可以将mmc中的代码从flash搬运到指定的地址处。当环境变量bootdelay计数到0时，此时uboot就会开始执行bootcmd中的命令。

> 🔔 bootdelay这个环境变量是一个计数器，当u-boot主体运行完毕后，此时bootdelay该变量的值将会开始递减，递减时间为1s，当递减到0时，此时u-boot将会跳转到bootcmd处开始执行bootcmd命令。

load mmc命令的使用比较简单，这里以F1C200S例子来讲解

load mmc 0:1 0x80008000 zImage

load mmc有三个参数，第一个参数是mmc分区，第二个参数是目标地址，第三个参数是源文件。即上面了命令意思是将mmc的0：1 分区中的zImage复制到内存中的0x80008000地址处。

TF卡属于mmc存储器的一种

上面完成了zImage的搬运任务，但是当前大部分的驱动都是基于设备树写的，关于设备树的知识，后面将单独一章节讲解，这种基于设备树(dts)的驱动使得代码重复率降低，但是内核就需要对其设备进行解析，否则设备将无法确定是哪一个驱动。这里将引入dtb文件，所谓dtb文件就是dts文件通过dtc编译器编译成的目标文件即二进制文件。内核通过加载并解析并解析该文件从而获取驱动相关的配置信息。下面给出F1C200S中使用的

load mmc 0:1 0x80c08000 suniv-f1c100s-licheepi-nano.dtb

上面的命令意思是将mmc的0:1分区中的suniv-f1c100s-licheepi-nano.dtb文件加载到内存中的0x80c08000地址处。关于0x80008000和0x80c08000这两个地址是如何确定的后面会详细讲到。将zImage（也就是内核镜像文件）和dtb（设备树文件)搬运到了指定内存处，此时u-boot的任务完全结束了，剩下的工作就是启动内核了，这个就需要bootm或者bootz命令。这里以bootz命令为例，如下：

bootz 0x80008000 - 0x80c08000

上面的意思是告诉内核镜像的起始地址为0x80008000，加载的设备树地址为0x80c08000。

⚠

1. bootz命令的格式是：bootz空格0x80008000空格-空格0x80c08000,注意-左右有空格

2. bootm命令是对没有使用设备树内核的镜像启动命令，早期版本的内核没有引入设备树，因此对于早期的内核一般使用的是bootm，其命令格式为bootm 内核地址，比如bootm x0x30008000，意思是从0x30008000开始启动内核，启动内核的过程其实是将pc指针指向该地址，这样处理器就会从该地址处运行代码。

上面详细讲解了bootcmd环境变量，该环境变量若要执行多条命令，则每个命令之间用;隔开，例如F1C200S常用的bootcmd 为：

load mmc 0:1 0x80008000 zImage;load mmc 0:1 0x80c08000 suniv-f1c100s-licheepi-nano.dtb;bootz 0x80008000 - 0x80c08000;

在图形配置界面中（图2.7）,图2.8所示中默认开启了bootcmd，其中变量值为run distro_bootcmd，这个值的意思是执行一个名为distro_bootcmd的一个脚本文件。 对

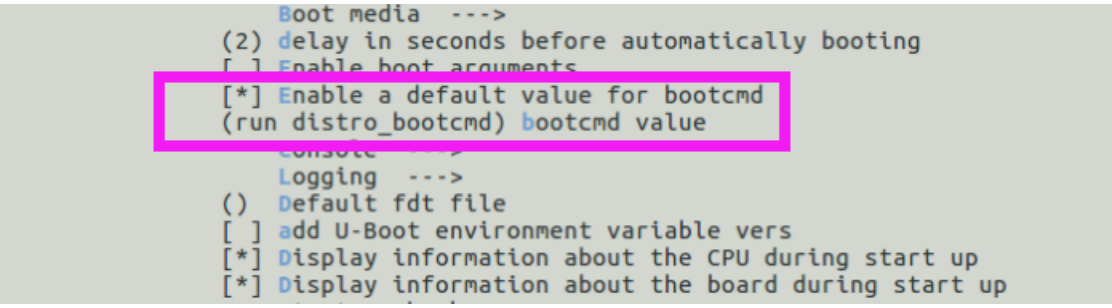

图 2.8

于一般的内核，可以直接上面的命令填写到bootcmd中，将光标移动到bootcmd value处，然后按回车键，进入编辑界面，如图2.9：

选择【Ok】后按回车键即可保存该变量值。如图2.10

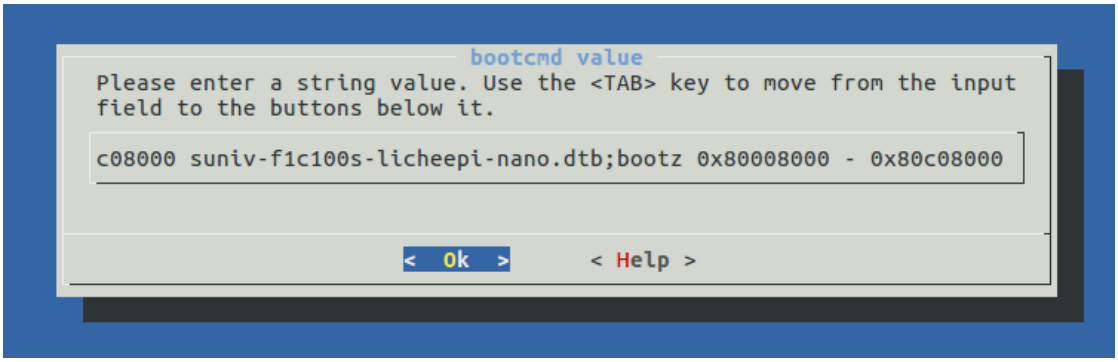

图 2.9

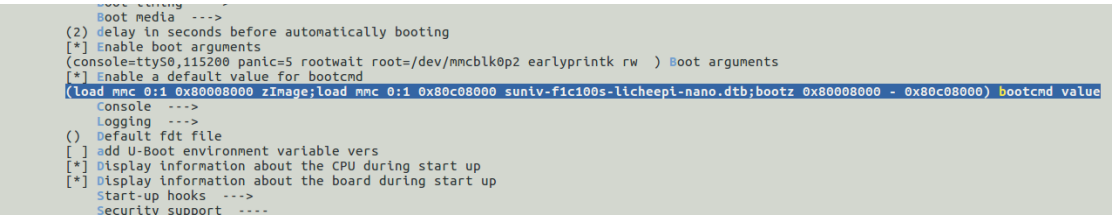

图 2.10

2.2.1.2 bootargs

bootargs也是u-boot环境变量中一个非常重要的变量，上面已经讲解了内核的启动可以通过bootcmd来完成，那接下来内核启动完毕后必须挂在根文件系统(rootfs)。但是内核并不知道根文件系统的具体位置，我们必须要告诉根文件的位置后内核才能将其挂载，这时就需要有bootargs变量。该变量的作用是告诉内核根文件系统的位置和属性以及必要的配置，这个就是图2.1中parameter的部分，u-boot会将bootargs的值（其实就是字符串）保存到事先规定好的地址处，当内核启动成功后就会获取该地址处的字符串值，然后对其进行解析并做相关的内核配置。这个过程看起来比较复杂，这里可以不用去了解细节，只需要知道bootargs是告诉内核根文件系统的位置属性以及一些配置参数。对于mmc 而言，一般采用分区来划分内存区域，下面以F1C200S常用的bootargs来讲解。

console=ttyS0,115200 panic=5 rootwait root=/dev/mmcblk0p2 earlyprintk rw

上面console=ttyS0,115200表示终端为ttyS0即串口0,波特率为115200。panic=5字面意思是恐慌，即linux内核恐慌，其实就是linux不知道怎么执行了，此时内核就需要做一些相关的处理，这里的5表示超时时间，当Linux卡住5秒后仍未成功就会执行Linux恐慌异常的一些操作。rootwait该参数是告诉内核挂在文件系统之前需要先加载相关驱动，这样做的目的是防止因mmc驱动还未加载就开始挂载驱动而导致文件系统挂载失败，所以一般bootargs中都要加上这个参数。root=/dev/mmcblk0p2 表示根文件系统的位置在mmc的0:2分区处，/dev是设备文件夹，内核在加载mmc中的时候就会在根文件系统中生成mmcblk0p2设备文件，该设备文件其实就是mmc的0:2分区，这样内核对文件系统的读写操作方式本质上就是读写/dev/mmcblk0p2该设备文件。earlyprintk参数是指在内核加载的过程中打印输出信息，这样内核在加载的时候终端就会输出相应的启动信息。rw表示文件系统的操作属性，此处rw表示可读可写。

设置bootargs参数时将光标移动到Enable boot， arguments处，点击键盘’Y’按键使其参数开启，如图2.11 然后将光标移动到下一个选项即Boot arguments (NEW)选项，然后按回

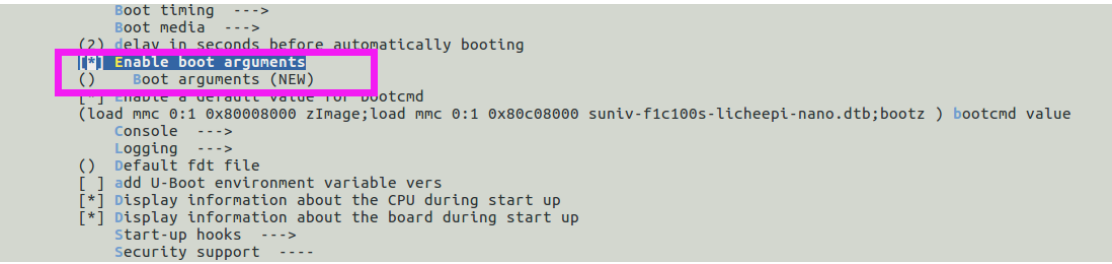

图 2.11

车键对该环境变量进行编辑，将上面的bootargs 字符串填写进入，然后按【Ok】键保存，如图2.12

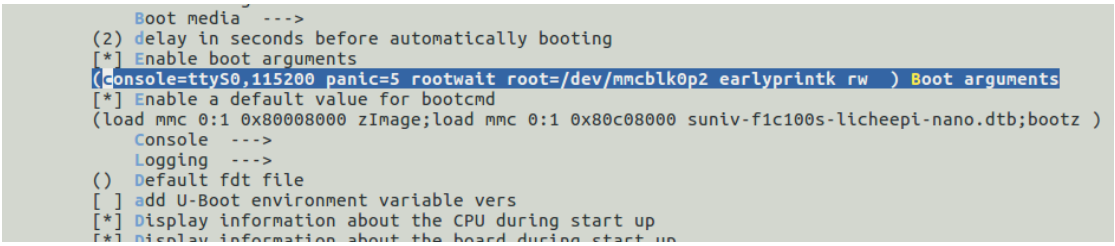

图 2.12

2.2.2 编译

先保存图形配置界面后推出界面，在终端执行make -j4即可对整个u-boot进行编译。

make -j4

如图2.13

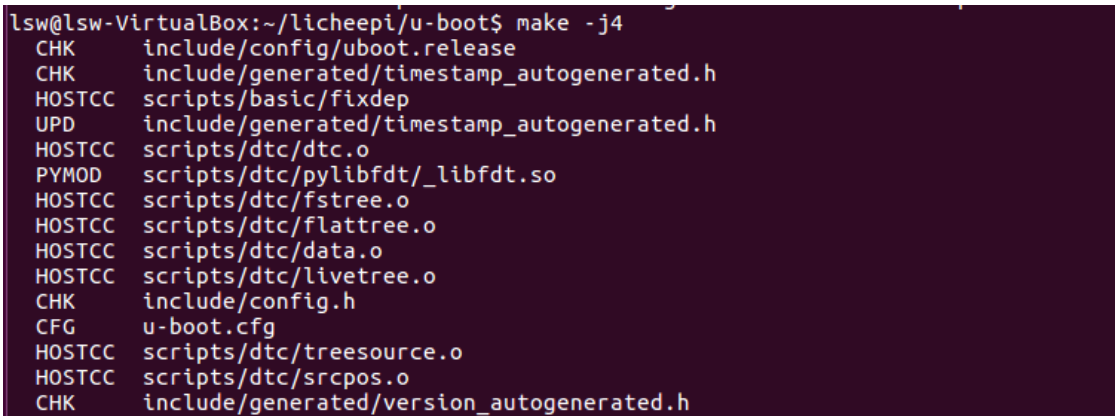

图 2.13

> make -j4后面的-j4表示4个核心进行编译，若电脑的处理器是2核心，请使用make -j2进行编译。

编译完成后会在当前目录生成u-boot-sunxi-with-spl.bin烧录文件。如图2.14，该文件就是我们最终要烧录的二进制文件。 在当前目录下会有一个隐藏的文件.config，该文件

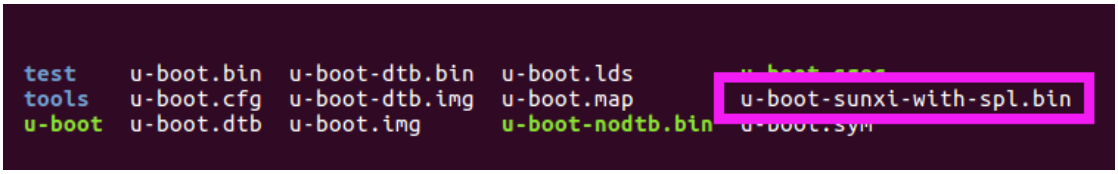

图 2.14

是u-boot编译后根据各个选项产生的配置文件，这个配置文件记录了所有配置选项的宏开关，编译的时候是根据最终的.config文件来进行编译的，当然编译前是需要有脚本解析.config文件然后进行相应的编译。

笔记 关于u-boot的配置过程这里简单说明下，其文件结构的和linux内核结构大致相同，对于当前而言，其配置方式有三种，分别是

1. make menuconfig
2. make xxx_defconfig
3. .config

这三种方式都可以对内核进行配置，三者之间的关系比作去饭店吃饭，make menuconfig可以看作菜单，.config和make xxx_defconfig是客人点的菜，但是其三个文件之间可能因为存在依赖关系,如果直接修改.config可能会导致配置失效。

只要将u-boot-sunxi-with-spl.bin烧录到tf卡的8k偏移处地址就可以了，烧录步骤如下：使用dd命令进行块搬移：

sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/sdb bs=1024 seek=8

if指的是输入文件，of指的是输出文件，这里的输出文件为主机电脑的/dev/sdb文件，也就是TF卡，这个可以用gparted 软件查看，该软件可以直接用命令安装即可：sudo apt-get install gparted，安装好后打开该软件，如图2.15在右上角可以看到两个硬盘sda，其中一个是主机的本地硬盘，另一个是TF，即sdb这里可以看到TF卡的卷标名为sdb，因此这里的of=/dev/sdb 烧录到8k偏移地址处是指绝对地址，这个绝对地址指的是TF卡的物理地址。这里为何是8k处而不是其他地址是由于F1C200S 内部的IROM中的一小段代码决定的。

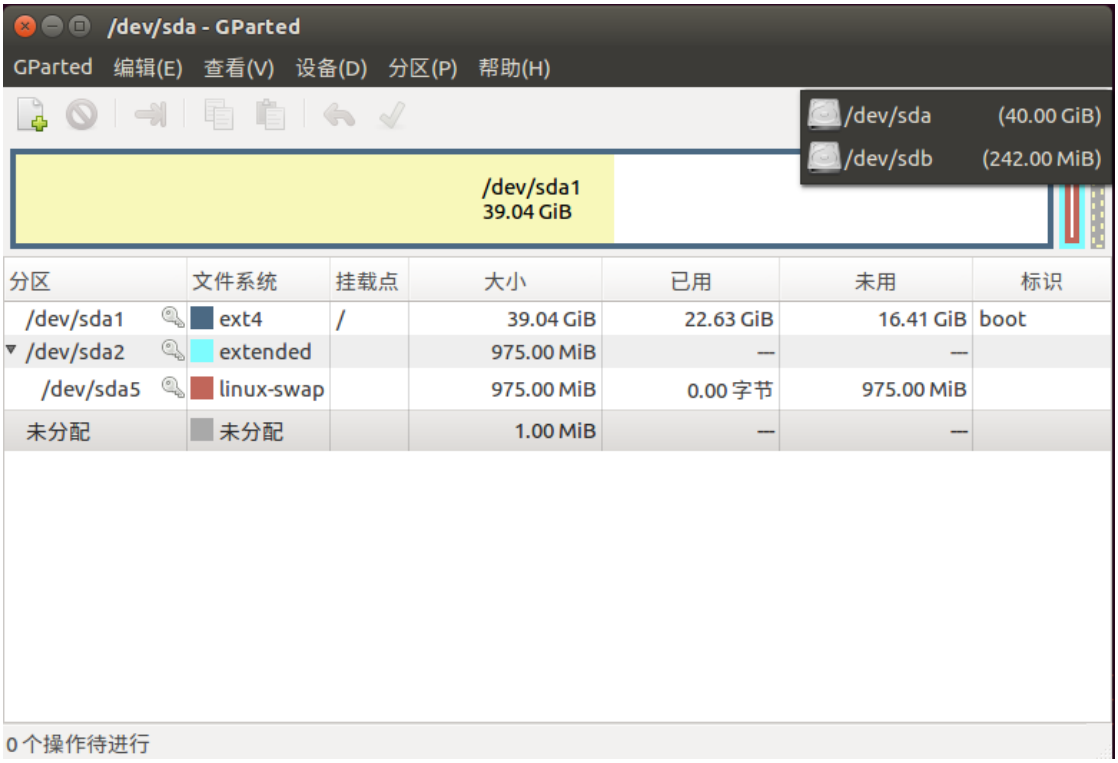

图 2.15

烧录完毕后如图2.16 现在这个TF卡内的u-boot可以启动内核了。将TF卡插入到开发

```
lsw@lsw-VirtualBox:~/licheepi/u-boot$ sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/sdb bs=1024 seek=8
[sudo] lsw 的密码:
记录了536+0 的读入
记录了536+0 的写出
548864 bytes (549 kB, 536 KiB) copied, 2.95585 s, 186 kB/s
lsw@lsw-VirtualBox:~/licheepi/u-boot$
```

图 2.16

板上，然后通过USB连接到电脑端，打开串口工具，可以看到按下开发板上的复位按钮，可以看到此时串口终端有信息输出，如图2.17，由于没有烧录内核，因此加载内核失败，停止在了u-boot命令终端处。 可以在终端输入pri命令打印环境变量的所有值，如

```
U-Boot SPL 2018.01-05679-g013ca45-dirty (Feb 16 2021 - 18:26:34)
DRAM: 64 MiB
Trying to boot from MMC1


U-Boot 2018.01-05679-g013ca45-dirty (Feb 16 2021 - 18:26:34 +0800) Allwinner Technology

CPU:   Allwinner F Series (SUNIV)
Model: Lichee Pi Nano
DRAM:  64 MiB
MMC:   SUNXI SD/MMC: 0
*** Warning - bad CRC, using default environment

In:    serial@1c25000
Out:   serial@1c25000
Err:   serial@1c25000
Net:   No ethernet found.
Hit any key to stop autoboot:  0
** Invalid partition 1 **
** Invalid partition 1 **
=>
```

图 2.17

图2.18所示，可以看到bootcmd和bootargs是正确的。

```
=> pri
arch=arm
baudrate=115200
board=sunxi
board_name=sunxi
bootargs=console=ttyS0,115200 panic=5 rootwait root=/dev/mmcblk0p2 earlyprintk rw
bootcmd=load mmc 0:1 0x80008000 zImage;load mmc 0:1 0x80c08000 suniv-f1c100s-licheepi-nano.dtb;bootz 0x80008000 - 0x80c08000
bootdelay=2
cpu=arm926ejs
fdtcontroladdr=83e56da8
fel_booted=1
fileaddr=80c08000
filesize=1572
soc=sunxi
stderr=serial@1c25000
stdin=serial@1c25000
stdout=serial@1c25000
```

图 2.18

也许到这里读者可能会认为u-boot内容结束了，其实不然，u-boot还有很多需要分析，由于有些部分过于复杂，对初学者不友好，因此u-boot的进阶内容将放到后面详细讲解。

2.3 内核移植

内核移植相对与u-boot移植复杂些，对于F1C200S而言，Linux官方源码已经对licheepi nano进行了支持，因此本次移植采用licheepi nano的配置文件，下面以linux5.7版本内核来讲解kernel移植步骤。进入linux内核官网(https://www.kernel.org/),点击https://www.kernel.org/pub/

进入下载界面，如图2.19，下载连接为 https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.7.1.tar.gz ,可以使用下载工具进行下载，下载后的源码包通过FTP传输到虚拟机。上传

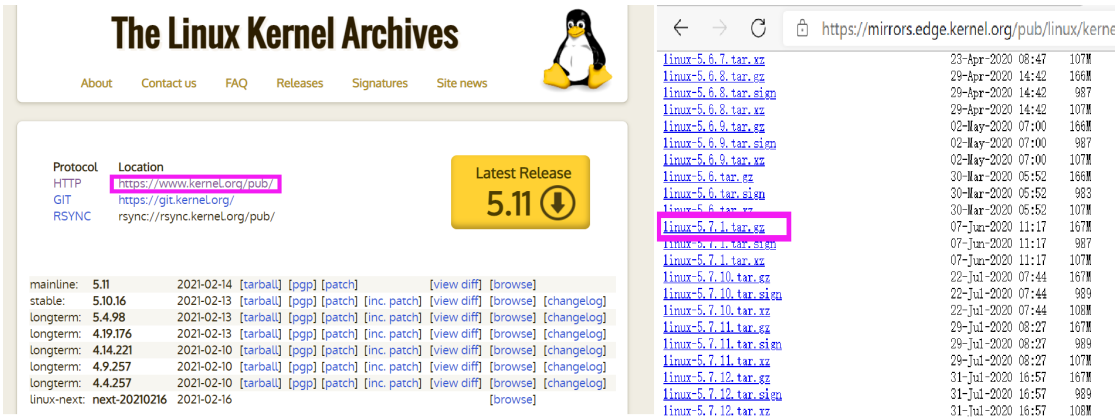

图 2.19

完毕后进入虚拟机解压源码，和u-boot步骤一样，在编译前必须对源码进行配置。进入该源码中的arch/arm/configs目录中，可以看到有很多开发板的配置文件，其中sunxi_defconfig是全志的配置文件，但是该配置文件非常不全，需要额外配置大量的选项，一般选项多大上千个，这里先使用licheepi_nano的配置文件。https://github.com/LiShanwenGit/MelonPI-MINI/tree/master/software 该目录下有一个linux-licheepi_nano_defconfig文件，这个文件是针对licheepi_nano的配置文件。这个配置文件与Lite200完全兼容，后面会详细分析该文件以及一些配置，这里为了快速启动内核就不做过多阐述。下载该文件，然后将其放到arch/arm/configs/目录下，如图2.20，然后回到主目录，和u-boot一样，在编译时必须指定交叉编译器。 打开主目录下的Makefile文件，然后找到CROSS_COMPILE变量，将

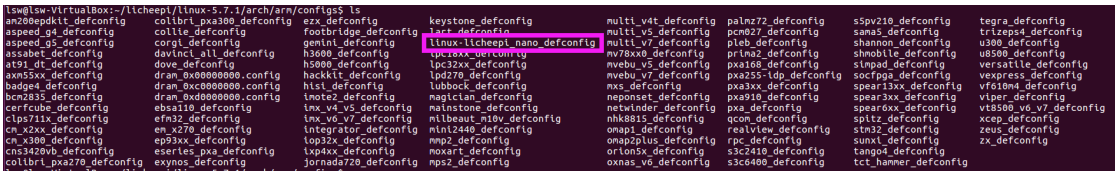

图 2.20

其修改为读者使用的交叉工具链前缀，同时指定架构即ARCH变量，这里使用arm-linux-gnueabi-为例，如图 保存退出，然后在终端处执行

make menuconfig

进入图形配置界面，如图2.22所示。 可以看到其配置模式和u-boot近似相同，也是通过上下键左右来操作和[Y][N]键来选择是否编译进内核。这里简单的先让linux内核跑起来，因此使用默认配置，不做任何修改。选择[Save]然后退出，在终端下输入

make -j4

如图2.23所示。 编译完毕后在就会生成zImage文件和dtb文件，zImage在arch/arm/boot目录下，dtb在arch/arm/boot/dts目录下。如图2.24。镜像编译完毕后就需要将其烧录到TF卡中，从上面的u-boot中bootcmd中可以看到需要将zImage 和dtb文件复制到TF卡的0:1分区中，这样u-boot在执行bootcmd中的load mmc命令时就可以找到zImage和dtb文件。下面来

```
358 # make CROSS_COMPILE=la64-linux-
359 # Alternatively CROSS_COMPILE can be set in the environment.
360 # Default value for CROSS_COMPILE is not to prefix executables
361 # Note: Some architectures assign CROSS_COMPILE in their arch/*/Makefile
362 ARCH            ?=arm
363 CROSS_COMPILE   ?=arm-linux-gnueabi-
364
365 # Architecture as present in compile.h
366 UTS_MACHINE     := $(ARCH)
367 SRCARCH         := $(ARCH)
368
369 # Additional ARCH settings for x86
370 ifeq ($(ARCH),i386)
371         SRCARCH := x86
372 endif
373 ifeq ($(ARCH),x86_64)
374         SRCARCH := x86
```

图 2.21

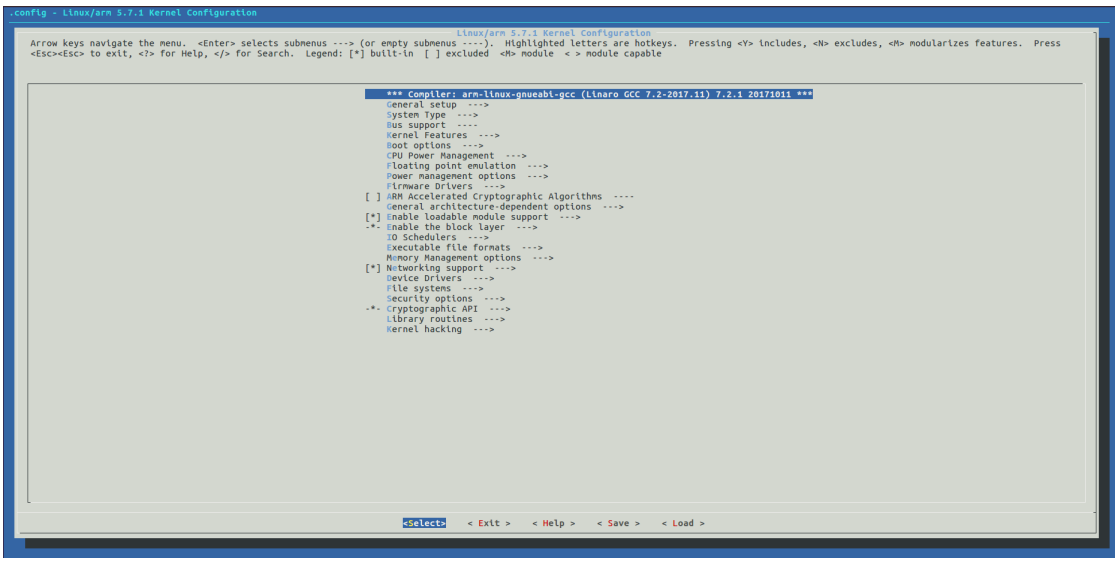

图 2.22

```
lsw@lsw-VirtualBox:~/licheepi/linux-5.7.1$ make -j4
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-common.h
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-oabi.h
  SYSHDR  arch/arm/include/generated/uapi/asm/unistd-eabi.h
  WRAP    arch/arm/include/generated/uapi/asm/kvm_para.h
  WRAP    arch/arm/include/generated/uapi/asm/bitsperlong.h
  WRAP    arch/arm/include/generated/uapi/asm/bpf_perf_event.h
  WRAP    arch/arm/include/generated/uapi/asm/errno.h
  WRAP    arch/arm/include/generated/uapi/asm/ioctl.h
  WRAP    arch/arm/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/arm/include/generated/uapi/asm/msgbuf.h
  WRAP    arch/arm/include/generated/uapi/asm/param.h
  WRAP    arch/arm/include/generated/uapi/asm/poll.h
  WRAP    arch/arm/include/generated/uapi/asm/resource.h
  WRAP    arch/arm/include/generated/uapi/asm/sembuf.h
  WRAP    arch/arm/include/generated/uapi/asm/shmbuf.h
  WRAP    arch/arm/include/generated/uapi/asm/siginfo.h
  WRAP    arch/arm/include/generated/uapi/asm/socket.h
  WRAP    arch/arm/include/generated/uapi/asm/sockios.h
  WRAP    arch/arm/include/generated/uapi/asm/termbits.h
```

图 2.23

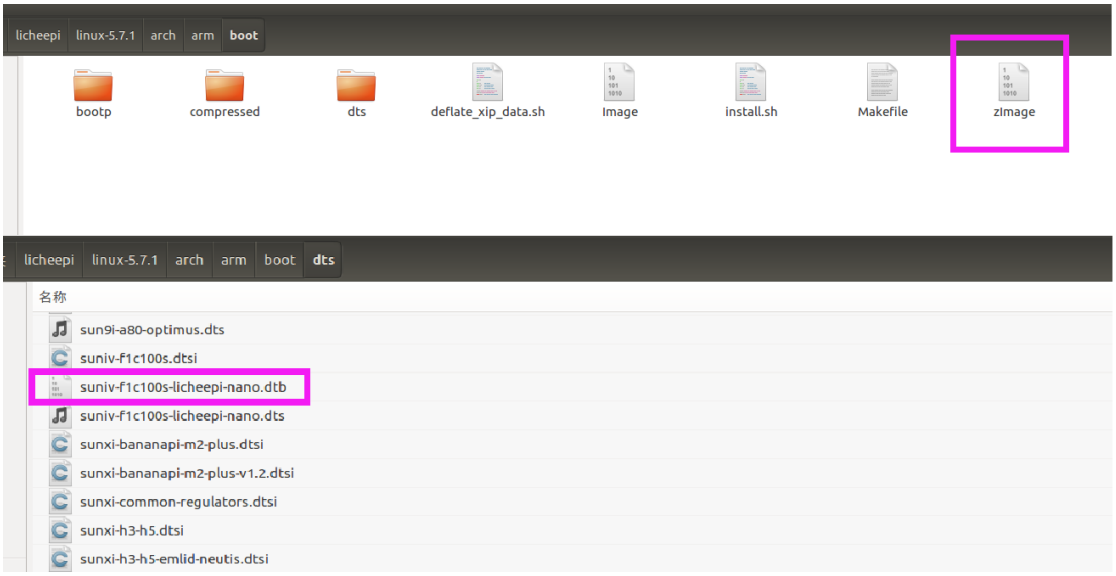

图 2.24

对TF卡进行分区。分区工具最常用的是gparted软件，该软件可以直接在终端中安装即可。

sudo apt-get install gparted

然后插入TF卡到电脑的USB上，打开该软件，可以看到此时有两个硬盘，一个是sda另一个是sdb，其中sdb就是TF卡。如 图2.25所示，选中sdb，可以看到有一个未分配的空间，

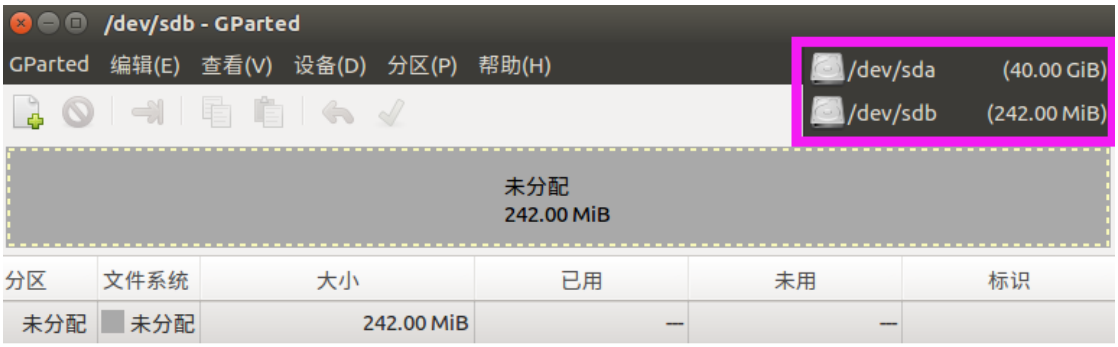

图 2.25

一般对于嵌入式系统而言需要将其分两个区，一个是存放zImage和dtb文件，另一个区存放根文件系统。对于第一个分区，一般格式为fat16格式，对于第二个区，一般为ext4格式。下面开始分区操作。

> ⚠ 这里第一分区格式只能是fat16格式，原因在于在u-boot中对mmc的操作命令load只支持fat16格式，不支持ext2、3或者4格式，因此第一分区只能是fat16，对于第二分区，一般分区为ext4格式，原因在于一般Linux操作系统默认挂载的文件系统格式是ext类型的。

选中未分配空间并右击鼠标，点击[新建]，然后填写相关属性，如图2.26，然后点击[添加],所示。 ，以相同的方式新建第二分区，如图2.27所示，最后点击[对勾]完成分

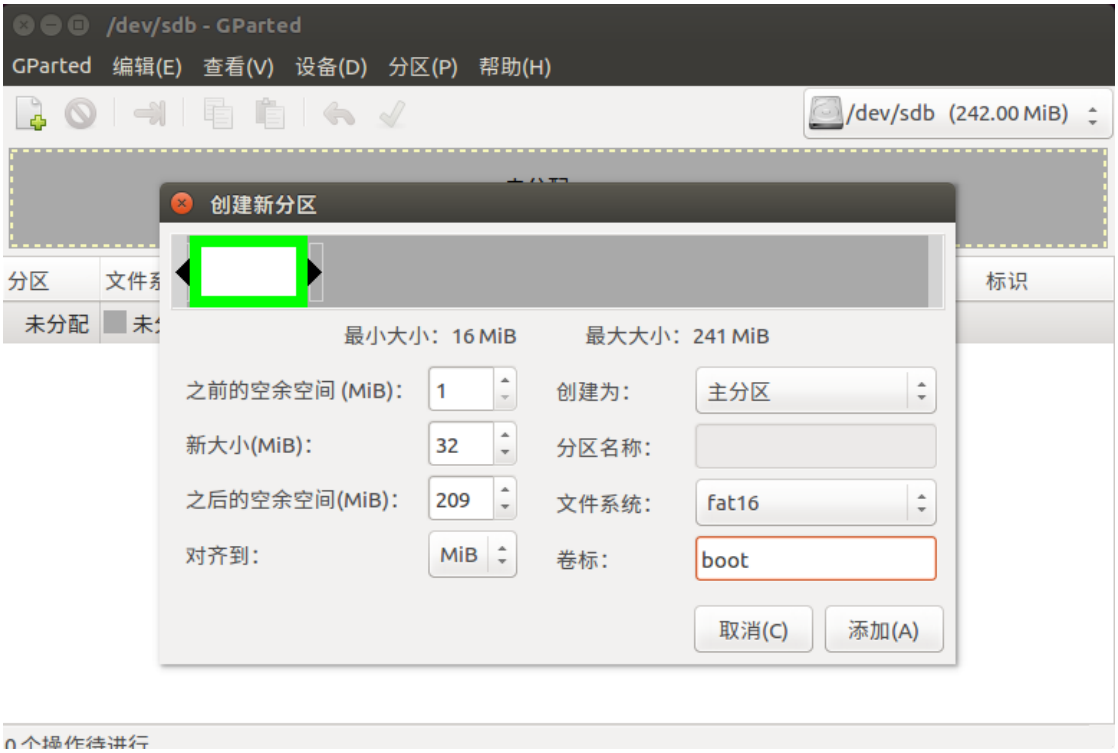

图 2.26

区操作，如图2.28 ，最终分区如图2.29 分区完毕后剩下的就是将zImage和dtb文件复制

图 2.27

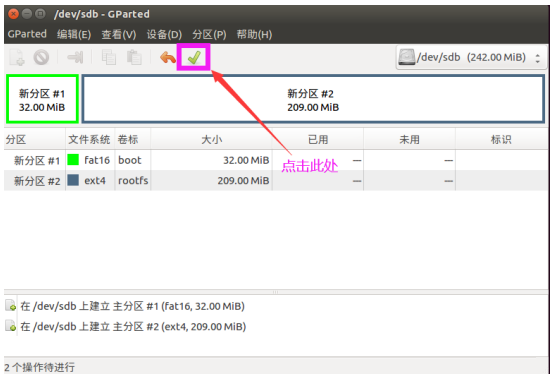

图 2.28

到TF卡的BOOT分区中。此时复制可以直接通过图形操作即可，也可以通过命令复制。将复制好内核的TF卡插到Lite200上接上USB和串口即可看到终端有信息输出，如图2.30 所示，可以看到内核已经成功启动，不过最终内核卡住了，其原因在于没有根文件系统，下节开始移植根文件系统。

⚠ 上面仅仅是简单的启动了Linux内核，实际还有很多文件需要分析，但这里作为简单的了解大致的过程，故不做深入讲解，后面分析驱动的时候将会详细讲解内核的文件结构和设备树相关的驱动以及如何添加和修改驱动。

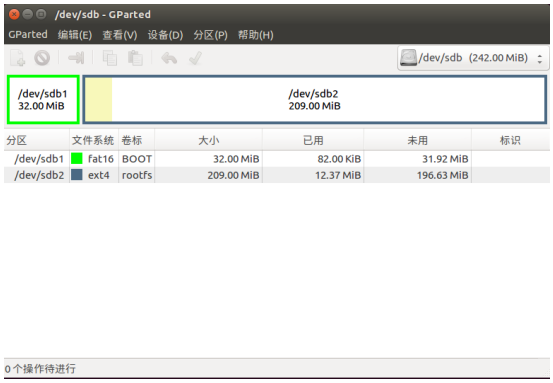

图 2.29

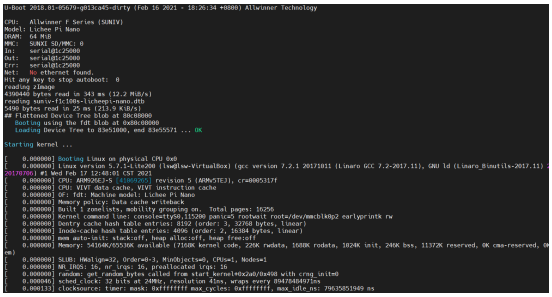

图 2.30

2.4 根文件系统移植

根文件系统是内核启动后挂载的第一个文件系统，如果没有根文件系统，内核将无法开启shell以及其他进程，下面来开始移植根文件系统。

笔记 实际上内核启动后会先挂载一个虚拟的文件系统，这个虚拟文件系统是在内存中运行的，其主要运行核心进程，虚拟文件系统挂载之后才挂载硬盘（TF卡或者emmc）上的根文件系统。

制作根文件系统的工具最有名的莫过于busyBox，该工具体积非常小，非常适合制作根文件系统。但是笔者尝试过使用busyBox 制作，然而体积较大，这主要原因在于文件系统需要有动态库和静态库，而对于7.2版本的交叉编译器而言，其动态和静态链接库实在太大，因此本文将使用另一个非常强悍的工具–Buildroot，该工具集成了非常多的其他应用，制作过程相对简单，不会像busyBox那样出现硬件架构不兼容情况。

由于根文件系统制作比较简单，这里就完全从头开始。进入buildroot官网（https://buildroot.org/downloa
下面以buildroot2018.2.11版本作为移植示例。将下载的源码包上传到虚拟机上，然后解压进入该源码目录中。进入源码目录后在终端输入

make clean

⚠ 在开始编译之前必须执行make clean以清楚一些预设配置，即使是第一次编译也是一样。

然后执行

make menuconfig

此时会进入图形配置界面，如图2.31。进入第一个Target options选项，配置如图2.32 第一个选项为架构选择，这里选择ARM架构小端模式，第二个为输出的二进制文件格式，这里选择EFL格式，第三个为架构体系，这里选择arm926t，因为F1C100S的架构就是这个架构，第四个为矢量浮点处理器，这里不勾选，因为对于F1C100S 而言，其内部没有浮点运算单元，只能进行软浮点运算，也就是模拟浮点预运算。第五个为应用程序二进制接口，这里选择EABI，原因是该格式支持软件浮点和硬件实现浮点功能混用。第六个为浮点运算规则，这里使用软件浮点，第七个选择指令集，这里选择ARM指令集，因

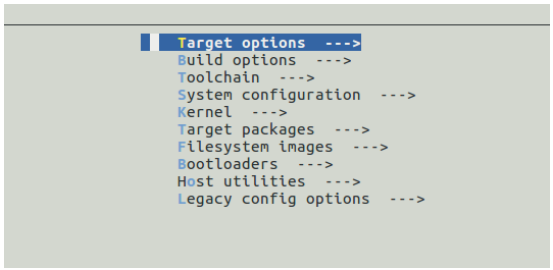

图 2.31　　图 2.32

为thumb主要针对Cortex M系列而言的，对于运行操作系统的A系列以及ARM9和ARM11而言，使用的都是32位的ARM指令集。

保存后，回到上一级配置界面，然后进入第二个Build options选项，配置如图2.33。保存后，回到上一级配置界面，然后进入第三个Toolchain选项，配置如图2.34。 这里

图 2.33

是选择交叉工具链，对于初学者而言，最好选择buildroot指定的默认交叉工具链，因为这里涉及到C库问题，如果使用自己安装的交叉工具链，编译很可能会报错，因为使用的C库不匹配。黄色框中的选项尽可能勾选，因为后面移植QT5 的时候需要用到C++相关库，如果这里没有勾选，QT5选项将无法勾选。

保存后回到上一级配置界面，然后进入第四个System configuration选项，配置如图2.35。第一个红色框中表示启动根文件系统后输出的信息（横幅），这里默认，当然你也可以修改此值，比如改为Welcome to Lite200，第二个红色框中表示开启登录密码，可以看到默认密码为空，这里就默认了，读者也可以根据自己情况修改即可。

保存后回到上一级配置界面，可以看到后面还有部分选项没有配置，由于剩下的选项也可以不用配置，因此这里为了简便，直接保持推出，然后执行

make

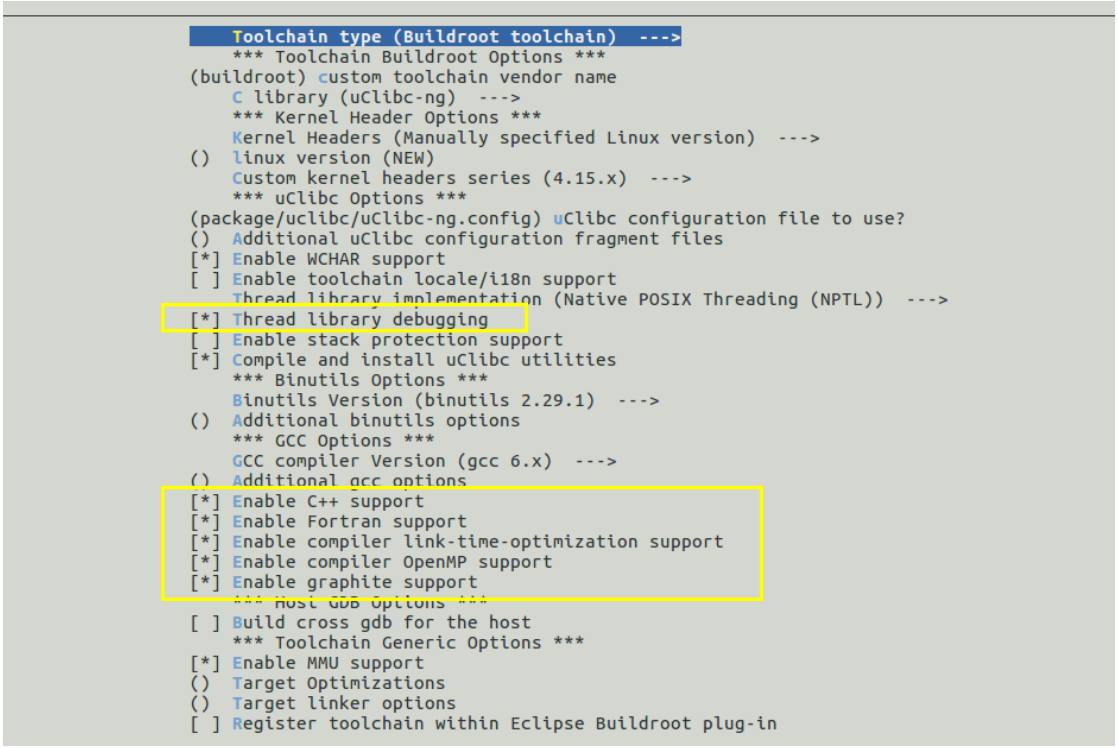
Toolchain type (Buildroot toolchain) --->
*** Toolchain Buildroot Options ***
(buildroot) custom toolchain vendor name
C library (uClibc-ng) --->
*** Kernel Header Options ***
Kernel Headers (Manually specified Linux version) --->
() linux version (NEW)
Custom kernel headers series (4.15.x) --->
*** uClibc Options ***
(package/uclibc/uClibc-ng.config) uClibc configuration file to use?
() Additional uClibc configuration fragment files
[*] Enable WCHAR support
[ ] Enable toolchain locale/i18n support
Thread library implementation (Native POSIX Threading (NPTL)) --->
[*] Thread library debugging
[ ] Enable stack protection support
[*] Compile and install uClibc utilities
*** Binutils Options ***
Binutils Version (binutils 2.29.1) --->
() Additional binutils options
*** GCC Options ***
GCC compiler Version (gcc 6.x) --->
() Additional gcc options
[*] Enable C++ support
[*] Enable Fortran support
[*] Enable compiler link-time-optimization support
[*] Enable compiler OpenMP support
[*] Enable graphite support
*** Host GDB Options ***
[ ] Build cross gdb for the host
*** Toolchain Generic Options ***
[*] Enable MMU support
() Target Optimizations
() Target linker options
[ ] Register toolchain within Eclipse Buildroot plug-in

图 2.34

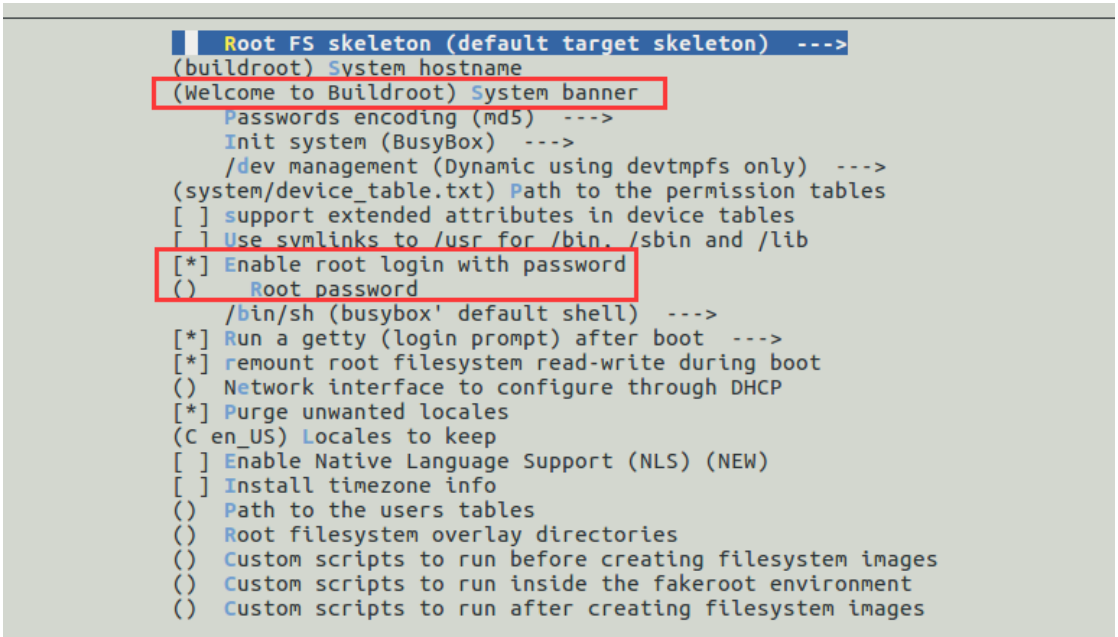
Root FS skeleton (default target skeleton) --->
(buildroot) System hostname
(Welcome to Buildroot) System banner
Passwords encoding (md5) --->
Init system (BusyBox) --->
/dev management (Dynamic using devtmpfs only) --->
(system/device_table.txt) Path to the permission tables
[ ] support extended attributes in device tables
[ ] Use symlinks to /usr for /bin, /sbin and /lib
[*] Enable root login with password
() Root password
/bin/sh (busybox' default shell) --->
[*] Run a getty (login prompt) after boot --->
[*] remount root filesystem read-write during boot
() Network interface to configure through DHCP
[*] Purge unwanted locales
(C en_US) Locales to keep
[ ] Enable Native Language Support (NLS) (NEW)
[ ] Install timezone info
() Path to the users tables
() Root filesystem overlay directories
() Custom scripts to run before creating filesystem images
() Custom scripts to run inside the fakeroot environment
() Custom scripts to run after creating filesystem images

图 2.35

⚠ 这里最好不要用多核编译，一般来说，首次编译过程非常慢，因为要下载很多必要的文件和交叉工具链。

编译完毕后可以在output/images目录下有一个rootfs.tar，该文件就是最终生成的根文件系统镜像，现在只需要将该镜像解压到TF卡的第二分区即可。插入TF卡到电脑端，进入out/images目录，然后输入

sudo tar -xvf rootfs.tar -C /media/lsw/rootfs/

此时可以看到TF卡的rootfs分区中有文件系统了，现在将TF卡弹出，插入到Lite200上，连接好串口，打开串口助手或者其他串口终端软件，可以看到根文件系统成功挂载，同时进入shell交互，如图2.36。 用户名默认为root，无密码，进入root账号后，在终端输出

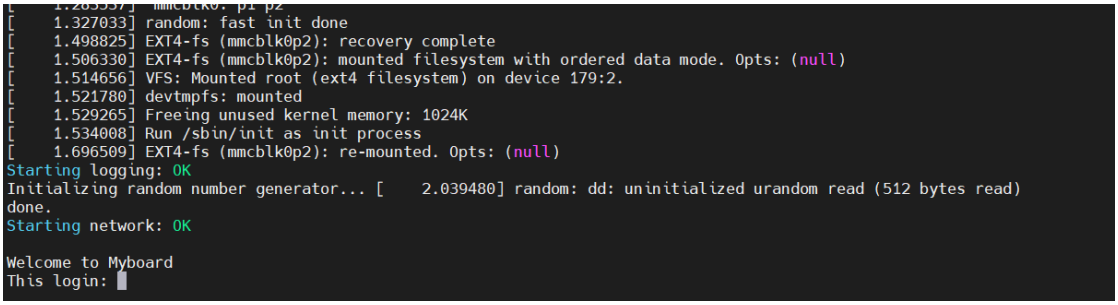

图 2.36

cd /

ls

可以看到文件系统中的linux的根目录情况，到此根文件系统的移植完成。

第 三 章 设备树(DTS)

3.1 设备树由来

3.2 文件结构

3.3 设备树与驱动关系

第 四 章 常用驱动移植

4.1 屏幕驱动移植

4.2 USB驱动移植

4.3 音频驱动移植

第 五 章 常用应用移植

5.1 QT5移植

5.2 图片浏览器移植

5.3 NES模拟器移植

5.4 视频播放器mplayer移植

第 六 章 驱动开发

6.1 linux驱动框架

6.2 字符设备驱动

6.3 块设备驱动

6.4 网络设备驱动

第 七 章 深入理解C语言

7.1 C指针

7.2 C语言与汇编的关系

7.3 C语言的编译过程

7.4 C语言的运行过程