**Spi flash启动**

<https://blog.csdn.net/weixin_41656917/article/details/114913259>

<https://wiki.sipeed.com/soft/Lichee/zh/Nano-Doc-Backup/index.html>

lichee nano官方linux config文件踩坑与填坑

<https://whycan.com/t_2703.html>

命令总汇：

sf probe 0 50000000;sf erase 0 0x10000;reset

sudo sunxi-fel -p spiflash-write 0 flashimg.bin

f1c100s/buildroot-2021.02.4/output/host/bin/arm-linux-gcc -o hi hi.c

ssh -p 6666 [root@127.0.0.1](mailto:root@127.0.0.1)

docker exec -it 0c01fd70c03e /bin/bash

rootfs：make

u-boot：make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

linux：（make ARCH=arm menuconfig）

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules\_install

1. 开发环境的搭建
   1. 安装交叉编译链

# 1. 下载 arm\_gcc 环境

wget <http://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabi/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz>

# 2. 解压到/usr/local/

sudo tar -vxJf gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi.tar.xz -C /usr/local/

# 3. 添加系统环境,在最后添加环境变量（在/etc/bash.bashrc末尾添加）

sudo nano /etc/bash.bashrc

PATH="$PATH:/usr/local/gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi/bin"

# 4. 使其配置生效

source /etc/bash.bashrc

# 5. 进行测试,如果失败，检查上面的环境是某设置正确。重新切换用户。在测试

arm-linux-gnueabi-gcc -v

# 6. 出现下面的消息既安装成功

COLLECT\_GCC=arm-linux-gnueabi-gcc

COLLECT\_LTO\_WRAPPER=/opt/gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi/bin/../libexec/gcc/arm-linux-gnueabi/7.2.1/lto-wrapper

...

* 1. 搭建SPI\_flash 烧录环境

（# 0. 如果无安装git 首先安装git）

sudo apt-get install git

# 1. 拉取 sunxi-tools 这个库

git clone -b f1c100s-spiflash <https://github.com/Icenowy/sunxi-tools.git>

# 2.进行安装

cd sunxi-tools

make && sudo make install

# 3. 错误处理：

# 3.1 如果出现 fatal error: libusb.h: No such file or directory 错误

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

# 3.2 fel.c:32:18: fatal error: zlib.h: 没有那个文件或目录

sudo apt-get install zlib1g-dev

# 4. 常用的命令有如下几个：

# 4.1 查看芯片是否开启了下载模式, 如果出现芯片信息既可以烧录

sudo sunxi-fel ver

# 4.2 烧录到 flash

sudo xunxi-fel -p spiflash-write [烧录的地址] [烧录的文件]

# 4.3 烧录到内存中(测试用)

sudo xunxi-fel -p write [烧录的内存地址] [烧录的文件]

# 5. 使芯片进入烧写模式（两种方式）

# 5.1 使用 U-boot删除flash中的引导数据

sf probe 0 50000000;sf erase 0 0x100000;reset

#切换到flash, 其中0为选择的flash，50000000为速度

#擦除flash

电子器材

低可信度描述已自动生成# 5.2 在开机的时候，拉低SPI flash的CS引脚，让芯片找不到flash芯片。

1. U-boot裁剪编译

新建 f1c100s 文件夹，进去文件夹。再进行如下操作。

mkdir f1c100s

cd f1c100s

* 1. 拉取u-boot

# 1. git 拉取 u-boot（nano-lcd800480分支）

git clone <https://github.com/Lichee-Pi/u-boot.git>

（git clone --depth=1 -b nano-lcd800480 <https://github.com/Lichee-Pi/u-boot.git>）

（git clone https://github.com/Lichee-Pi/u-boot.git -b nano-lcd800480）

# 2. 查看分支，并切换到对应分支

git branch -a (查看所有分支)

[git branch 查看本地分支]

git checkout nano-lcd800480

# 3. 进行配置

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- licheepi\_nano\_spiflash\_defconfig

* 1. 配置 u-boot

# 1. 进行可视化配置, 进入配置页面

make ARCH=arm menuconfig

# 如果出现错误，安装如下的包

sudo apt-get install libncurses5-dev libncursesw5-dev

# 2. 如果使用LCD，修改LCD选项



x:800,y:480,depth:18,pclk\_khz:33000,le:87,ri:40,up:31,lo:13,hs:1,vs:1,sync:3,vmode:0

文本

描述已自动生成

# 3. 修改boot arguments,传输到linux的参数

console=tty0 console=ttyS0,115200 panic=5 rootwait root=/dev/mtdblock3 rw rootfstype=jffs2

(console=tty0 console=ttyS0 设置linux终端及

115200 串口波特率

panic=5 当Linux卡住5秒后，会执行Linux恐慌异常的一些操作

rootwait 表示等待flash设备初始化完成以后再挂载，否则设备还没初始化完成就挂载根文件系统会出错的

root=/dev/mtdblock3 指定了flash第3分区为rootfs所在分区【配置linux时，配置MTD（Memory Technology Device）】

rw 表示根文件系统是可以读写的，不加 rw 的话可能无法在根文件系统中进行写操作

rootfstype= jffs2 指定了这个分区的格式是jffs2)

# 4. 修改bootcmd，启动linux命令

sf probe 0 50000000; sf read 0x80c00000 0x100000 0x4000; sf read 0x80008000 0x110000 0x400000; bootz 0x80008000 - 0x80C00000

(sf spi/qspi flash

sf probe 0 50000000 选择并初始化flash设备,速度为50000000

sf read 0x80c00000 0x100000 0x4000 读取linux内核

sf read 0x80008000 0x110000 0x400000 读取dtb文件

把flash偏移0x100000处, 长度为0x4000的数据, 写入到内存0x80c00000

bootz 0x80008000 - 0x80C00000 启动内存0x80008000 - 0x80C00000间的linux kernel)

图形用户界面, 文本, 应用程序

描述已自动生成

# 5. 保存退出

* 1. 修改 flash

# 1. 在代码编辑器中 打开 ./drivers/mtd/spi/spi\_flash\_ids.c

nano ./drivers/mtd/spi/spi\_flash\_ids.c

# 2. 在 170 行左右添加如下代码

{"xt25f128b", INFO(0x0b4018, 0x0, 64 \* 1024, 256, RD\_FULL | WR\_QPP | SECT\_4K) },

# 3. 保存退出

* 1. 编译、烧写验证 u-boot

# 编译代码

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

# 如果最后出现下面语句，编译成功

BINMAN u-boot-sunxi-with-spl.bin

# 错误处理:scripts/Makefile.build:425: recipe for target 'scripts/dtc/pylibfdt' failed

sudo apt-get install swig python-dev python3-dev

(烧写：

# 烧写到flash中

sudo sunxi-fel -p spiflash-write 0 u-boot-sunxi-with-spl.bin

# 烧写到 sram中，在测试的时候使用

sudo sunxi-fel uboot u-boot-sunxi-with-spl.bin)

1. Linux裁剪编译

进入 f1c100s 文件夹进行操作

* 1. 下载Linux

git clone --depth=1 -b master <https://gitee.com/LicheePiNano/Linux.git>

（4.15.0-rc8-licheepi-nano+）

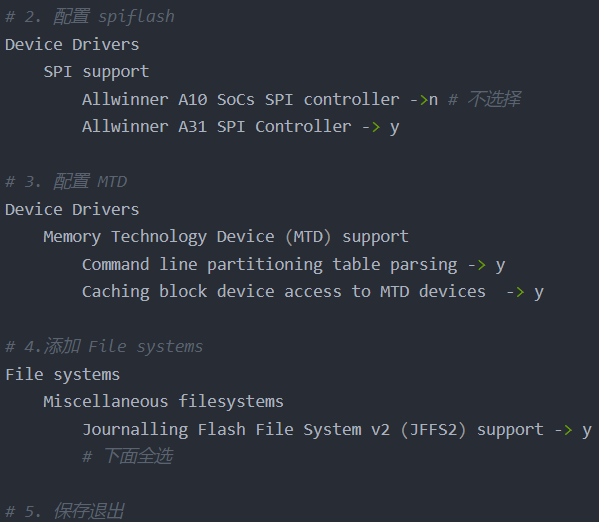
* 1. 配置Linux

# 使用./arch/arm/configs/f1c100s\_nano\_linux\_defconfig配置文件

make ARCH=arm f1c100s\_nano\_linux\_defconfig

# 图形化配置

make menuconfig ARCH=arm



Command line partitioning table parsing

#解析内核参数传过来的分区信息，如果用设备树应该可以不选

Caching block device access to MTD devices

#生成/dev/mtdblock\*设备，不选会在启动的时候卡在

watiting for device /dev/mtdblock\*

* 1. 配置flash 和设备树

# 配置 flash 驱动

打开 ./drivers/mtd/spi-nor/spi-nor.c 1186行。添加如下代码（对flash支持）

{ "xt25f128b", INFO(0x0b4018, 0, 64 \* 1024, 256, 0) },

# 配置设备树

（见<https://blog.csdn.net/weixin_41656917/article/details/114913259>）

* 1. 编译Linux

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules\_install

# 错误处理 fatal error: openssl/bio.h:

sudo apt-get install libssl-dev

(编译成功后，生成文件所在位置：

内核img文件：./arch/arm/boot/zImage

设备树dtb文件：./arch/arm/boot/dts/suniv-f1c100s-licheepi-nano.dtb

modules文件夹：./out/lib/modules)

1. 根文件系统编译

进入 f1c100s 文件夹操作

* 1. 下载并解压Buildroot代码包

wget <https://buildroot.org/downloads/buildroot-2021.02.4.tar.gz>

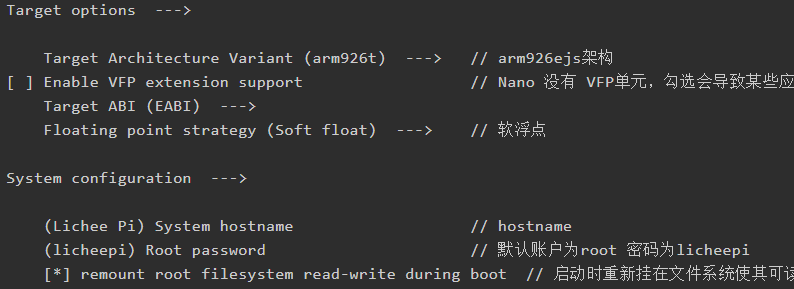
tar xvf buildroot-2021.02.4.tar.gz

cd buildroot-2021.02.4/

* 1. 配置

make menuconfig

一些配置的简单说明



另外：

文本

描述已自动生成

* 1. 编译

make

* 1. 串口终端来传输文件：lrzsz

文本

描述已自动生成

勾选lrzsz，编译

SecureCRT的下载、安装

* + - 1. <https://blog.csdn.net/qq_39052513/article/details/100272502>
      2. <https://zhuanlan.zhihu.com/p/343764437>

使用串口CRT工具连接终端，输入命令"lrz",回车，发送一个文件到目标板

使用mobaxterm

#sz：

执行 sz xxxx 命令：

在界面当中右键，选择Receive file using Z-modem

#rz：

在界面当中右键，选择Send file using Z-modem

chmod +x xxx

1. 制作烧录镜像
   1. 下载脚本nano\_flash\_dd.sh

<https://dl.sipeed.com/shareURL/LICHEE/Nano/SDK>

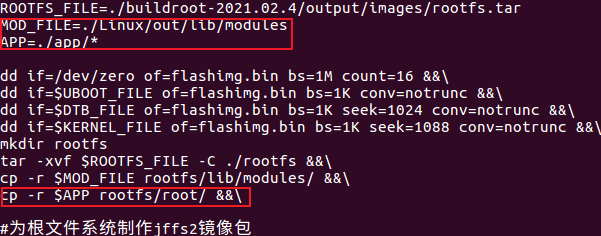
* 1. 修改脚本内容及权限

# 将（MOD\_FILE=./Linux/out/lib/modules/4.15.0-rc8-licheepi-nano+）改为：

MOD\_FILE=./Linux/out/lib/modules

（要将4.15.0-rc8-licheepi-nano+一起打包到rootfs）

# 在f1c100s添加文件夹document，在里面可以放应用文件等，可打包到rootfs



# 修改权限并执行脚本

sudo chmod +x nano\_flash\_dd.sh

./nano\_flash\_dd.sh

* 1. 烧录

sudo sunxi-fel -p spiflash-write 0 flashimg.bin

(sudo sunxi-fel -p spiflash-write 0 u-boot-sunxi-with-spl.bin

sudo sunxi-fel -p spiflash-write 0x0100000 suniv-f1c100s-licheepi-nano.dtb

sudo sunxi-fel -p spiflash-write 0x0110000 zImage

sudo sunxi-fel -p spiflash-write 0x0510000 jffs2.img)

# 错误处理 mkfs.jffs2: not found

sudo apt-get install mtd-utils

1. 应用程序

arm-linux-gnueabi-gcc是用来编译内核的；编译应用，使用buildroot输出的交叉编译工具链，与根文件系统一致，位于：buildroot-2021.02.4/output/host/bin/arm-linux-gcc

f1c100s/buildroot-2021.02.4/output/host/bin/arm-linux-gcc -o hi hi.c

# 错误处理：-sh: ./hello: not found

查看是否缺少运行库：

/home/moon/lichee/buildroot-2021.02.4/output/host/bin/arm-linux-readelf -d hello

hello.c

#include <stdio.h>

int main(int argc, char \*\*argv)

{

if (argc >= 2)

printf("Hello, %s!\n", argv[1]);

else

printf("Hello, world!\n");

return 0;

}

1. Wifi（ESP8089）
   1. 克隆代码

git clone https://github.com/Lichee-Pi/esp8089.git

* 1. 将代码移动(复制)到：Linux/drivers/net/wireless

cd /home/moon/f1c100s/Linux/drivers/net/wireless

mv (cp -r) esp8089/ .

* 1. 修改 esp8089/Makefile（存疑）

将（obj-$(CONFIG\_ESP8089) := esp8089.o）改为：

obj-m := esp8089.o

* 1. 将驱动选项添加至配置界面中

# 修改Kconfig：添加esp8089驱动配置信息

(将esp8089源码中的Kconfig添加到linux/drivers/net/wireless的Kconfig)

nano Kconfig

source "drivers/net/wireless/esp8089/Kconfig"

文本

描述已自动生成

* 1. 添加编译选项

# 修改Makefile：将添加的驱动关联到源码中（Linux/drivers/net/wireless）

nano Makefile

obj-$(CONFIG\_ESP8089) += esp8089/

文本

描述已自动生成

* 1. 内核配置

cd ../../.. （回到Linux/）

make ARCH=arm menuconfig

# 网络相关配置

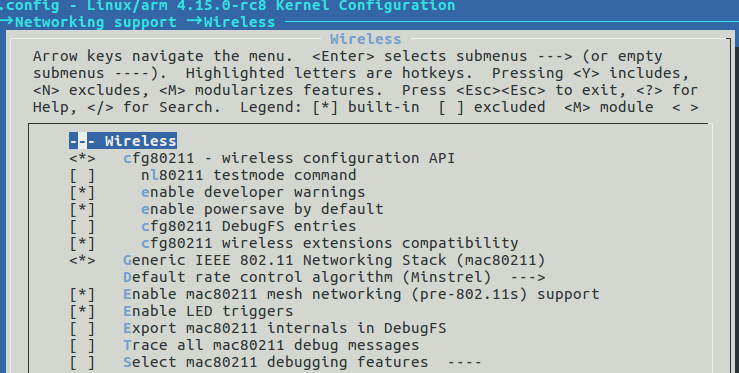
(Networking support -> Networking options)

文本

描述已自动生成

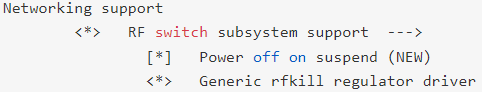
# Wireless配置

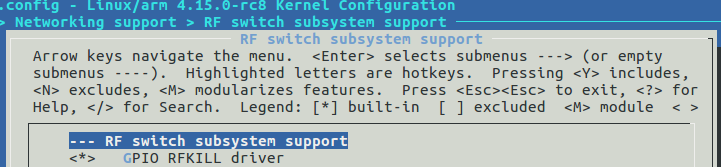
(Networking support -> Wireless)



开启rfkill：（存疑，不知是否必须）

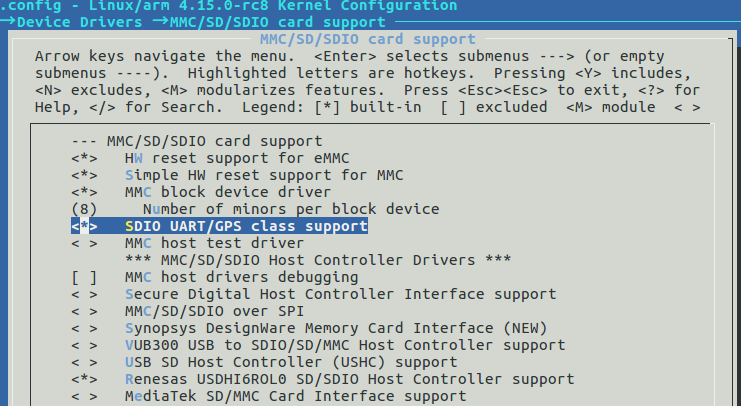
（不开启，出现警告：rfkill: Cannot open RFKILL control device）





# SDIO接口驱动

(Device Drivers -> MMC/SD/SDIO card support)



# WIFI驱动（esp8089）

(Device Drivers -> Network device support -> Wireless LAN)

文本

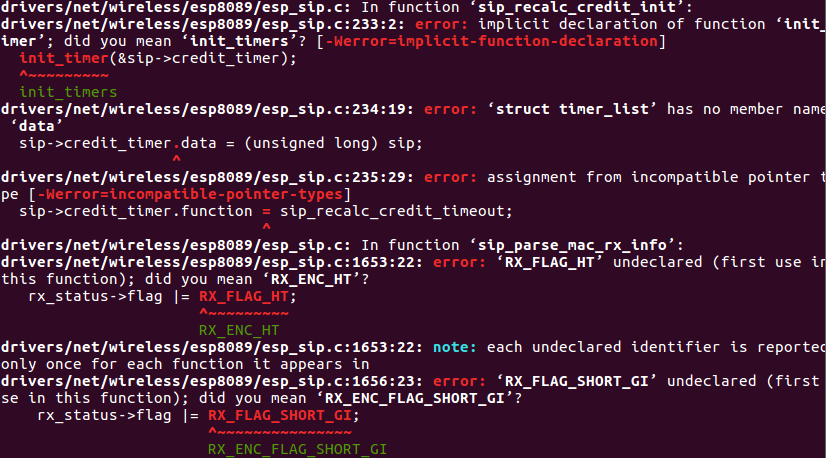
描述已自动生成

* 1. 编译

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules\_install

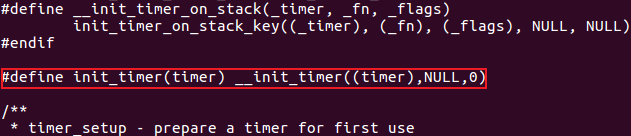


* 1. 修改源码

# 内核版本的timer.h与驱动不匹配

修改Linux/include/linux/timer.h，定义init\_timer这个宏

添加：#define init\_timer(timer) \_\_init\_timer((timer),NULL,0)



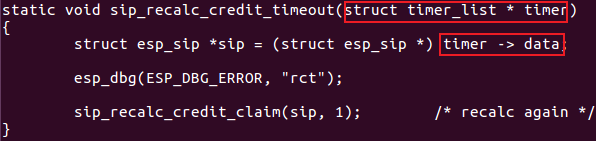
定义data这个成员。

修改Linux/include/linux/timer.h，添加：unsigned long data;

文本

描述已自动生成

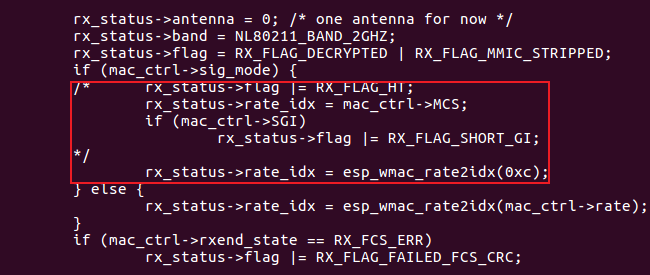
修改Linux/drivers/net/wireless/esp8089/esp\_sip.c的sip\_recalc\_credit\_timeout



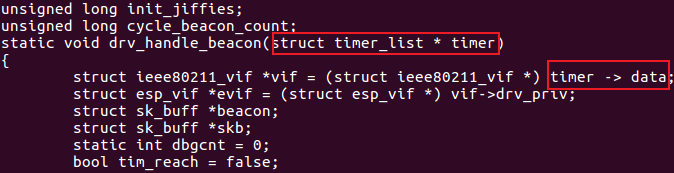
# 内核版本的mac80211.h与驱动不匹配

修改Linux/drivers/net/wireless/esp8089/esp\_sip.c的sip\_recalc\_credit\_timeout

rx\_status->rate\_idx = esp\_wmac\_rate2idx(0xc);



修改Linux/drivers/net/wireless/esp8089/esp\_mac80211.c

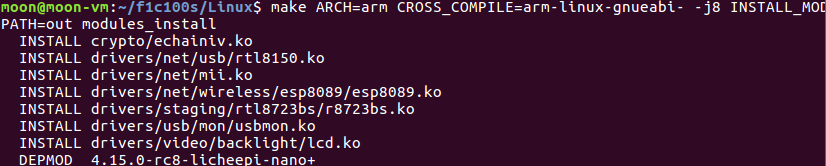


* 1. 重新编译

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules\_install



生成的esp8089.ko在Linux/drivers/net/wireless/esp8089

(Linux/out/lib/modules/4.15.0-rc8-licheepi-nano+/kernel/drivers/net/wireless/esp8089/esp8089.ko)

将esp8089.ko打包到rootfs (可以不用)

（cp drivers/net/wireless/esp8089/esp8089.ko /home/moon/f1c100s/app）

将fireware固件打包到rootfs

（cp Linux/drivers/net/wireless/esp8089/firmware/eagle\_fw\_\* app/）

* 1. Buildroot配置

（添加WIFI连接过程中需要的工具：

Wireless\_tools；wpa\_supplicant；openssl；iw）

cd buildroot-2021.02.4/

make clean

make menuconfig

-->Target packages

-->Networking applications

-->iw

-->wireless tools

-->wpa\_supplicant

-->Install wpa\_cli binary

-->Install wpa\_client shared library

-->Install wpa\_passphrase binary

编译：make

脚本打包，烧录进板

* 1. 开发板操作

（root 12346 进入系统）

# 将固件移动到/lib/firmware

(固件在esp8089源码的firmware文件夹下)

mkdir /lib/firmware && mv eagle\_fw\_\* /lib/firmware

# 加载模块

cd /lib/modules/4.15.0-rc8-licheepi-nano+/kernel/drivers/net/wireless/esp8089

modprobe esp8089.ko config=crystal\_26M\_en=1

# 配置网络配置文件

vi /etc/wpa\_supplicant.conf

ctrl\_interface=/var/run/wpa\_supplicant

ctrl\_interface\_group=0

ap\_scan=1

network={

ssid="moon"

scan\_ssid=1

key\_mgmt=WPA-EAP WPA-PSK IEEE8021X NONE

pairwise=TKIP CCMP

group=CCMP TKIP WEP104 WEP40

psk="asdfghjkl"

priority=5}

# 开启wlan0网口

ifconfig wlan0 up

# 连接网络

wpa\_supplicant -B -i wlan0 -c /etc/wpa\_supplicant.conf

# 分配IP和网关

udhcpc -i wlan0

# 测试

ping [www.baidu.com](http://www.baidu.com)

（编译驱动后一定要通过 make modules\_install命令生成驱动，然后把整个生成的驱动放到文件系统中，然后通过modprobe命令去加载，该命令可以读取modules.dep文件找到驱动的依赖关系）

ESP8089 AP模式：https://whycan.com/t\_2091.html

1. USB Gadget
2. ts fb
   1. 安装工具tslib（触摸屏适配层）

图形用户界面, 文本, 应用程序

描述已自动生成

文本

描述已自动生成

buildroot只生成根文件系统，ko是后来拷进去的。

sunxi-fel 是一边擦，一边写， 读 flash 有bug, 所以没有校验操作。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成文本

描述已自动生成

一直停在Starting kernel ...

https://whycan.cn/t\_2402.html

uname -r 显示当前系统的版本号

4.15.0-rc8-licheepi-nano+

进入uboot，输入：version（显示uboot版本）

U-Boot 2018.01-05679-g013ca45-dirty

ACM(虚拟串口)和ECM(网络共享)

**Docker镜像**

1. 安装荔枝派docker开发镜像
   1. 导入lichee-nano编译环境镜像
      1. 使用nano.tar.gz（<https://pan.baidu.com/s/1aYcGfzyz-g4CbxGSsVREGQ>）

gunzip nano.tar.gz

docker import nano.tar

* + 1. 通过docker官方仓库拉取lichee-nano编译环境镜像

docker pull zepan/licheepi-nano:1.0.0

1. 相关指令

#镜像命令

docker images #查看所有本地主机上的镜像 可以使用docker image ls代替

docker rm id #删除镜像docker image rm（删除镜像前，删除依赖的容器）

#容器命令

docker run 镜像id #新建容器并启动

（docker run --name licheepi -it 14e95922d8ec /bin/bash）

（docker run --name licheepi -dp 6666:22 14e95922d8ec /usr/sbin/sshd -D）

docker ps (-a) #列出所有运行的容器 docker container list

docker rm id #删除指定容器

docker rm -f $(docker ps -aq) #删除所有的容器

进入容器

docker exec -it id /bin/bash

docker exec -it b1aab5161608 /bin/bash

ssh -p 6666 [root@127.0.0.1](mailto:root@127.0.0.1) （若开起了ssh）

#退出容器

exit #容器直接退出，并关闭

ctrl +P +Q #容器不停止退出 ---注意：这个很有用的操作

#启动和停止容器

docker start id #启动容器

docker restart id #重启容器

docker stop id #停止当前正在运行的容器

#其他常用命令

docker cp id: 容器路径 主机路径 #从容器内拷贝到主机上

#查看docker目录大小

sudo du -h --max-depth=1 /var/lib/docker

**Xboot**

<https://whycan.com/t_449.html> （4楼）

<https://wiki.sipeed.com/soft/Lichee/zh/Nano-Doc-Backup/get_started/first_eat.html> （二）

<https://github.com/xboot/xboot/blob/master/README.md> （交叉编译链）

1. 下载交叉编译链

交叉编译链，arm-none-eabi：

<https://pan.baidu.com/s/1dDtssIt#list/path=%2F>

下载arm-eabi-工具下载包

gcc-arm-10.2-2020.11-x86\_64-arm-none-eabi.tar.xz

解压

tar -vxJf gcc-arm-10.2-2020.11-x86\_64-arm-none-eabi.tar.xz

1. 编译，烧录

make CROSS\_COMPILE=/path/bin/arm-eabi- PLATFORM=arm32-f1c100s

（多半有问题）

xboot.bin位置：xboot/output/xboot.bin

烧写到RAM中并运行

sudo sunxi-fel -p write 0x80000000 output/xboot.bin; sudo sunxi-fel exec 0x80000000

（sudo sunxi-fel exec 地址 //调用指定地址的函数）

烧录到spi flash

sudo sunxi-fel -p spiflash-write 0 output/xboot.bin; sudo sunxi-fel exec 0

**SD启动**

# 错误处理: Unable to mount 64 GB Volume （ubuntu无法识别大容量SD卡）

sudo apt-get install exfat-utils

1. 开发环境的搭建
   1. 安装交叉编译链

# 1. 下载 arm\_gcc 环境

wget <http://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabi/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz>

# 2. 解压到/usr/local/

sudo tar -vxJf gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi.tar.xz -C /usr/local/

# 3. 添加系统环境,在最后添加环境变量（在/etc/bash.bashrc末尾添加）

sudo nano /etc/bash.bashrc

PATH="$PATH:/usr/local/gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi/bin"

# 4. 使其配置生效

source /etc/bash.bashrc

# 5. 进行测试,如果失败，检查上面的环境是某设置正确。重新切换用户。在测试

arm-linux-gnueabi-gcc -v

# 6. 出现下面的消息既安装成功

COLLECT\_GCC=arm-linux-gnueabi-gcc

COLLECT\_LTO\_WRAPPER=/opt/gcc-linaro-7.2.1-2017.11-x86\_64\_arm-linux-gnueabi/bin/../libexec/gcc/arm-linux-gnueabi/7.2.1/lto-wrapper

...

* 1. 搭建SPI\_flash 烧录环境

（# 0. 如果无安装git 首先安装git）

sudo apt-get install git

# 1. 拉取 sunxi-tools 这个库

git clone -b f1c100s-spiflash <https://github.com/Icenowy/sunxi-tools.git>

# 2.进行安装

cd sunxi-tools

make && sudo make install

# 3. 错误处理：

# 3.1 如果出现 fatal error: libusb.h: No such file or directory 错误

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

# 3.2 fel.c:32:18: fatal error: zlib.h: 没有那个文件或目录

sudo apt-get install zlib1g-dev

# 4. 常用的命令有如下几个：

# 4.1 查看芯片是否开启了下载模式, 如果出现芯片信息既可以烧录

sudo sunxi-fel ver

# 4.2 烧录到 flash

sudo xunxi-fel -p spiflash-write [烧录的地址] [烧录的文件]

# 4.3 烧录到内存中(测试用)

sudo xunxi-fel -p write [烧录的内存地址] [烧录的文件]

# 5. 使芯片进入烧写模式（两种方式）

# 5.1 使用 U-boot删除flash中的引导数据

sf probe 0 50000000;sf erase 0 0x100000;reset

#切换到flash, 其中0为选择的flash，50000000为速度

#擦除flash

电子器材

低可信度描述已自动生成# 5.2 在开机的时候，拉低SPI flash的CS引脚，让芯片找不到flash芯片。

1. U-boot裁剪编译

新建 f1c100s 文件夹，进去文件夹。再进行如下操作。

mkdir f1c100s

cd f1c100s

* 1. 拉取u-boot

# 1. git 拉取 u-boot（nano-lcd800480分支）

git clone <https://github.com/Lichee-Pi/u-boot.git>

（git clone --depth=1 -b nano-lcd800480 <https://github.com/Lichee-Pi/u-boot.git>）

# 2. 查看分支，并切换到对应分支

git branch -a (查看所有分支)

[git branch 查看本地分支]

git checkout nano-lcd800480

# 3. 进行配置

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- licheepi\_nano\_spiflash\_defconfig

* 1. 配置 u-boot

# 1. 进行可视化配置, 进入配置页面

make ARCH=arm menuconfig

# 如果出现错误，安装如下的包

sudo apt-get install libncurses5-dev libncursesw5-dev

# 2. 如果使用LCD，修改LCD选项

x:800,y:480,depth:18,pclk\_khz:33000,le:87,ri:40,up:31,lo:13,hs:1,vs:1,sync:3,vmode:0

文本

描述已自动生成

* 1. 修改 flash

# 1. 在代码编辑器中 打开 ./drivers/mtd/spi/spi\_flash\_ids.c

nano ./drivers/mtd/spi/spi\_flash\_ids.c

# 2. 在 170 行左右添加如下代码

{"xt25f128b", INFO(0x0b4018, 0x0, 64 \* 1024, 256, RD\_FULL | WR\_QPP | SECT\_4K) },

# 3. 保存退出

* 1. 编译、烧写验证 u-boot

# 编译代码

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

# 如果最后出现下面语句，编译成功

BINMAN u-boot-sunxi-with-spl.bin

# 错误处理:scripts/Makefile.build:425: recipe for target 'scripts/dtc/pylibfdt' failed

sudo apt-get install swig python-dev python3-dev

(烧写：

# 烧写到flash中

sudo sunxi-fel -p spiflash-write 0 u-boot-sunxi-with-spl.bin

# 烧写到 sram中，在测试的时候使用

sudo sunxi-fel uboot u-boot-sunxi-with-spl.bin)

1. TF卡分区

# 放进TF卡的内容有：

第一分区：

boot.scr

zImage

suniv-f1c100s-licheepi-nano.dtb

第二分区：

根文件系统内容

# 在TF卡上构建系统之前，需要将TF卡进行分区与格式化：

sudo fdisk -l # 首先查看电脑上已插入的TF卡的设备号

sudo umount /dev/sdXx # 若自动挂载了TF设备，请先卸载

sudo fdisk /dev/sdX # 进行分区操作

# d删除各个分区（若已存分区即按）

# n新建分区，第一分区暂且申请为32M，剩下的空间都给第二分区

# w 保存写入并退出

# p 查看已有分区

sudo mkfs.vfat /dev/sdX1 # 将第一分区格式化成FAT

sudo mkfs.ext4 /dev/sdX2 # 将第一分区格式化成EXT4

1. 一键烧录

Linux环境---TF卡启动：

插上tf卡后，命令行执行 sudo fdisk -l 查看tf卡盘号；

执行 sudo dd if=/path/to/your-dd-image of=/your/tf-card && sync 即可完成烧录

（sudo dd if=Nano\_tf\_800480.dd of=/dev/sdb && sync）

1. Linux裁剪编译

进入 f1c100s 文件夹进行操作

* 1. 下载Linux

git clone https://github.com/Lichee-Pi/linux.git --depth=1 -b nano-4.14-exp

* 1. 配置Linux

# 下载配置文件

wget https://whycan.cn/files/members/3/lichee\_nano\_linux\_config.zip

mv lichee\_nano\_linux\_config.zip .config

# 图形化配置

make menuconfig ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi-

* 1. 编译Linux

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8

（make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j8 INSTALL\_MOD\_PATH=out modules\_install ）

# 错误处理 fatal error: openssl/bio.h:

sudo apt-get install libssl-dev

(编译成功后，生成文件所在位置：

内核img文件：./arch/arm/boot/zImage

设备树dtb文件：./arch/arm/boot/dts/suniv-f1c100s-licheepi-nano.dtb

modules文件夹：./out/lib/modules)

1. 根文件系统编译

进入 f1c100s 文件夹操作

* 1. 下载并解压Buildroot代码包

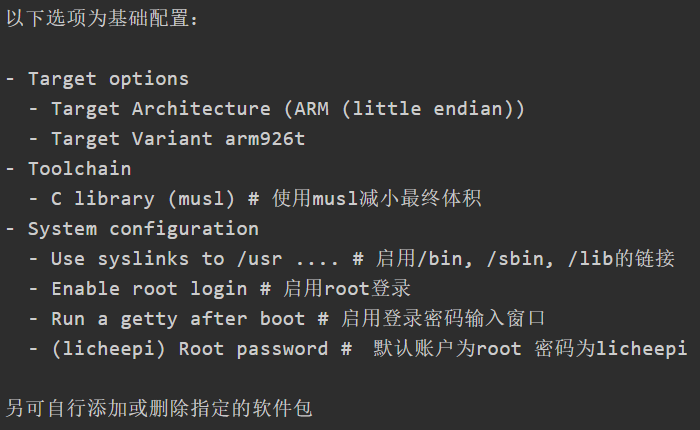
wget <https://buildroot.org/downloads/buildroot-2021.02.4.tar.gz>

tar xvf buildroot-2021.02.4.tar.gz

cd buildroot-2021.02.4/

* 1. 配置

make menuconfig



一些配置的简单说明

文本

描述已自动生成

另外：

文本

描述已自动生成

* 1. 编译

make

* 1. 串口终端来传输文件：lrzsz

文本

描述已自动生成

勾选lrzsz，编译

SecureCRT的下载、安装

* + - 1. <https://blog.csdn.net/qq_39052513/article/details/100272502>
      2. <https://zhuanlan.zhihu.com/p/343764437>

使用串口CRT工具连接终端，输入命令"lrz",回车，发送一个文件到目标板

使用mobaxterm

#sz：

执行 sz xxxx 命令：

在界面当中右键，选择Receive file using Z-modem

#rz：

在界面当中右键，选择Send file using Z-modem

chmod +x xxx

1. 制作烧录镜像
2. 应用程序

arm-linux-gnueabi-gcc是用来编译内核的；编译应用，使用buildroot输出的交叉编译工具链，与根文件系统一致，位于：buildroot-2021.02.4/output/host/bin/arm-linux-gcc

f1c100s/buildroot-2021.02.4/output/host/bin/arm-linux-gcc -o hi hi.c

# 错误处理：-sh: ./hello: not found

查看是否缺少运行库：

/home/moon/lichee/buildroot-2021.02.4/output/host/bin/arm-linux-readelf -d hello

hello.c

#include <stdio.h>

int main(int argc, char \*\*argv)

{

if (argc >= 2)

printf("Hello, %s!\n", argv[1]);

else

printf("Hello, world!\n");

return 0;

}