

WOC O' Linux

要求分析：

1. 开发板选择：

考虑到开发难度和成本，本次demo我选择了LicheeZero开发板。

2. 如何去为一个开发版制作Linux系统：这边我就默认用Linux操作系统啦，当然甚至其他的RTOS也是可以的就比如官方声称这块板子有着很好的RTThread系统的支持

在linux系统软件架构可以分为4个层次（从低到高分别为）：

引导加载程序

引导加载程序（Bootloader）是固化在硬件Flash中的一段引导代码，用于完成硬件的一些基本配置，引导内核启动。

同时，Bootloader会在自身与内核分区之间存放一些可设置的参数（Boot parameters），比如IP地址，串口波特率，要传递给内核的命令行参数。

系统内核

系统内核（Kernel）是整个操作系统的最底层，它负责整个硬件的驱动，以及提供各种系统所需的核心功能，包括防火墙机制、是否支持LVM或Quota等文件系统等等，如果内核不认识某个最新的硬件，那么硬件也就无法被驱动，你也就无法使用该硬件。计算机真正工作的东西其实是硬件，例如数值运算要使用到CPU、数据储存要使用到硬盘、图形显示会用到显示适配器、音乐发声要有音效芯片、连接Internet可能需要网络卡等等。内核就是控制这些芯片如何工作。

文件系统

Linux文件系统（File System）中的文件是数据的集合，文件系统不仅包含着文件中的数据而且还有文件系统的结构，所有Linux用户和程序看到的文件、目录、软连接及文件保护信息等都存储在其中。

文件系统是操作系统用于明确存储设备（常见的是磁盘，也有基于NAND Flash的固态硬盘）或分区上的文件的方法和数据结构；即在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。

用户程序

用户应用程序（Application）为了完成某项或某几项特定任务而被开发运行于操作系统之上的计算机程序。

上面内容摘自<https://www.cnblogs.com/schips/p/13129047.html>

本次demo中，我使用的Bootloader是u-boot，系统内核为Linux Kernel 4.10，以上两个由板子官方提供，因为这两个都与硬件有关，必须适配，而官方做好了这个适配。文件系统用Buildroot搭建最最原始、啥都没有的一个文件系统。用户程序暂未编写，仅仅在内核层面配置好了OLED屏幕的驱动。

3.系统存储介质：

这块板子上留出了SPI接口的Flash芯片焊接处，我不打算使用，一是因为焊接不便，二是因为最大支持16MB，要接上更大的Flash芯片需要更改配置，比较麻烦，三是因为要执行对Flash芯片的烧写还需要专门的设备，比较麻烦。此外，板子上还留出了TF卡槽，通过TF卡存储系统比较方便可以通过读卡器编写十分方便；容量大，方便以后扩展功能；且驱动方便，官方已经做好配置。于是毫不犹豫选用TF卡。

材料准备：

硬件：

- 1.LicheeZero开发板(随板附赠一些排针)
- 2.12864 OLED(I2C接口，SSD1306芯片，这是)
- 3.电烙铁、焊锡丝(用于焊接LizheeZero上的排针)
- 4.USB转串口模块(由于板子是将串口作为终端，所以为了让Xshell可以通过usb口连接到终端，还需要这样一个模块)
- 4.杜邦线若干(用于连接各个硬件设备)
- 5.TF卡、读卡器(8GB已经非常非常非常足够了)
- 6.USB A to Micro USB连接线(用于连接lichee zero和电脑)

软件：

- 1.VMware(或者任何装有Linux主流发行版的虚拟机或物理机，推荐Ubuntu18.04或16.04。其他发行版会不会遇到问题暂时不知)
- 2.Xshell(在设备开启后需要通过Xshell打印信息)

开始！

在开始前首先要说明一下，我们所说的“定制一个系统”，其实质是我们使用开放的源码，做自己的配置，然后编译得到目标文件，放入存储介质。

而这个编译操作是在linux环境下做的，所以我们事先需要一个Linux系统，以下的操作如果不做说明都是在我的虚拟机Ubuntu18.04中做的。

由于我已经在我的电脑上做过一次，所以第二次编译的结果有可能因为我没有clean而与大家的略有不同。这一点大家要注意。

由于每个人的电脑装的软件不同，为了之后少出错，我先给大家安装一些必备的软件：

```
sudo apt-get install gcc
sudo apt-get install make
sudo apt-get install g++
sudo apt-get install gcc-arm-linux-gnueabihf
sudo apt-get install device-tree-compiler
sudo apt-get install python
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install libssl-dev
```

如果安装了以上软件之后仍然会在某一步报错，没关系，复制关键信息去浏览器搜索，上面有一些也是我经过搜索才去安装的，安装之后再重新进行一下操作就正常了。

1.编译u-boot:

u-boot下载地址：<https://github.com/Lichee-Pi/u-boot/archive/v3s-current.zip>

可以git clone，如果速度太慢可以在windows下下载(毕竟大家在Win下方法比较多)，然后传到linux。

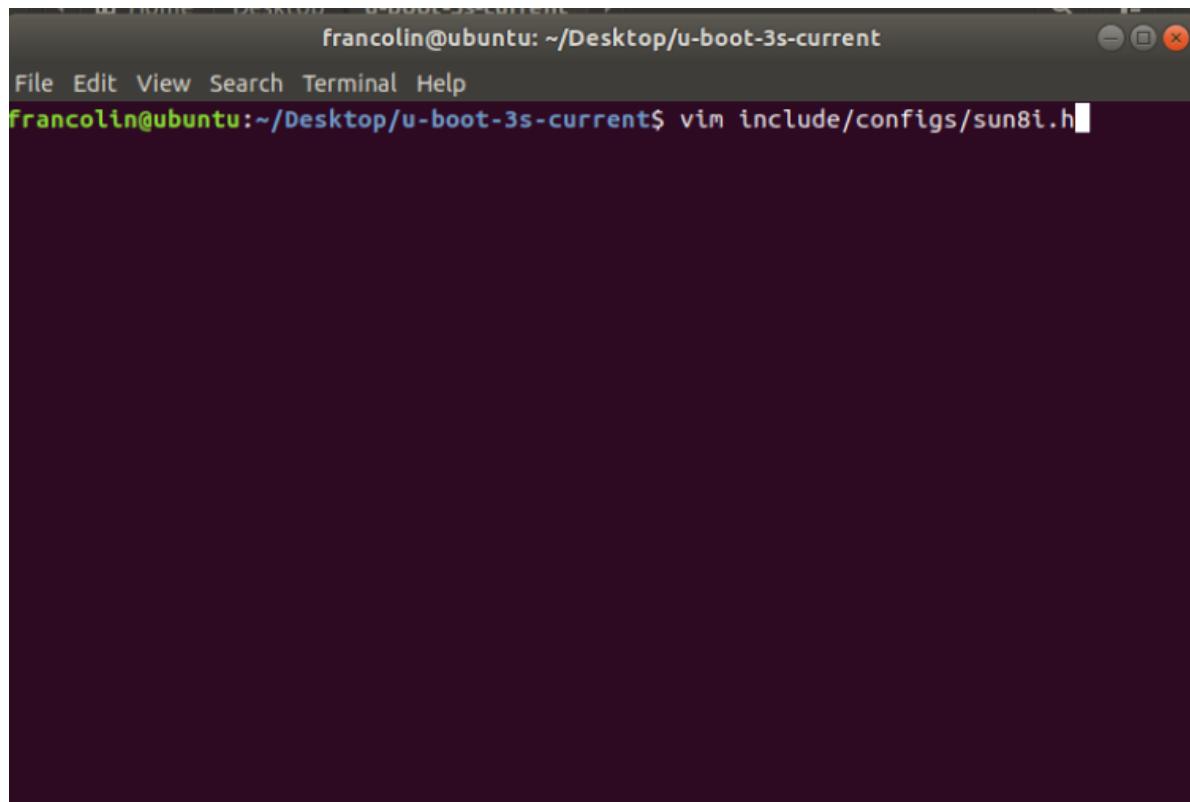
1):下载并解压上述文件得到一个文件夹并进入该目录

2):更改配置：

既然要从TF卡启动系统，当然做一些配置，而这个配置就需要在u-boot中做，至于为什么我想上面这个链接<https://www.cnblogs.com/schips/p/13129047.html>已经给出了比较好的解释了。注意：

uboot (universal bootloader) 是一种可以用于多种嵌入式CPU的BootLoader程序，换言之，uboot是bootloader的一个子集。**uboot的核心作用就是启动操作系统内核，uboot的本质就是一段裸机程序**

为了修改配置我们要修改sun8i.h这个文件



A screenshot of a terminal window titled "francolin@ubuntu: ~/Desktop/u-boot-3s-current". The window shows the command "francolin@ubuntu:~/Desktop/u-boot-3s-current\$ vim include/configs/sun8i.h" being typed. The terminal is dark-themed.

在文件里添加下面这段代码：（大家可以想一下这段代码是什么意思（小声

```
#define CONFIG_BOOTCOMMAND    "setenv bootm_boot_mode sec; " \
                            "load mmc 0:1 0x41000000 zImage; " \
                            "load mmc 0:1 0x41800000 sun8i-v3s-licheepi-zero- \
                            dock.dtb; " \
                            "bootz 0x41000000 - 0x41800000;" \
 \
#define CONFIG_BOOTARGS        "console=ttyS0,115200 panic=5 rootwait \
                           root=/dev/mmcblk0p2 earlyprintk rw vt.global_cursor_default=0"
```

如下图

```
francolin@ubuntu: ~/Desktop/u-boot-3s-current
File Edit View Search Terminal Help
#elif defined CONFIG_MACH_SUN8I_A83T
    #define CONFIG_SUNXI_USB_PHYS 3
#elif defined CONFIG_MACH_SUN8I_V3S
    #define CONFIG_SUNXI_USB_PHYS 1
#else
    #define CONFIG_SUNXI_USB_PHYS 2
#endif

/*
 * Include common sunxi configuration where most the settings are
 */
#include <configs/sunxi-common.h>

#define CONFIG_BOOTCOMMAND "setenv bootm_boot_mode sec; " \
                        "load mmc 0:1 0x41000000 zImage; " \
                        "load mmc 0:1 0x41800000 sun8i-v3s-lcheepi-zero-doc \
k.dtb; " \
                        "bootz 0x41000000 - 0x41800000;" \
                        "\n"
#define CONFIG_BOOTARGS "console=ttyS0,115200 panic=5 rootwait root=/dev/mm \
cblk0p2 earlyprintk rw vt.global_cursor_default=0"
#endif /* __CONFIG_H */
-- INSERT --                                         39,1          Bot
```

接着保存退出即可。

3):准备编译：

首先添加配置，成功如下图所示：

```
sudo ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make LicheePi_Zero_defconfig
```

```
francolin@ubuntu:~/Desktop/u-boot-3s-current$ vim include/configs/sun8i.h
francolin@ubuntu:~/Desktop/u-boot-3s-current$ sudo ARCH=arm CROSS_COMPILE=arm-li
nux-gnueabihf- make LicheePi_Zero_defconfig
:#                                     # configuration written to .config
:#                                     #
francolin@ubuntu:~/Desktop/u-boot-3s-current$
```

上面的是默认配置，但如果要使用通用4.3寸屏请输入：

```
ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make
LicheePi_Zero_480x272LCD_defconfig
```

如果你要使用通用5寸屏请输入：

```
ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make
LicheePi_Zero_800x480LCD_defconfig
```

接下来进行menuconfig：

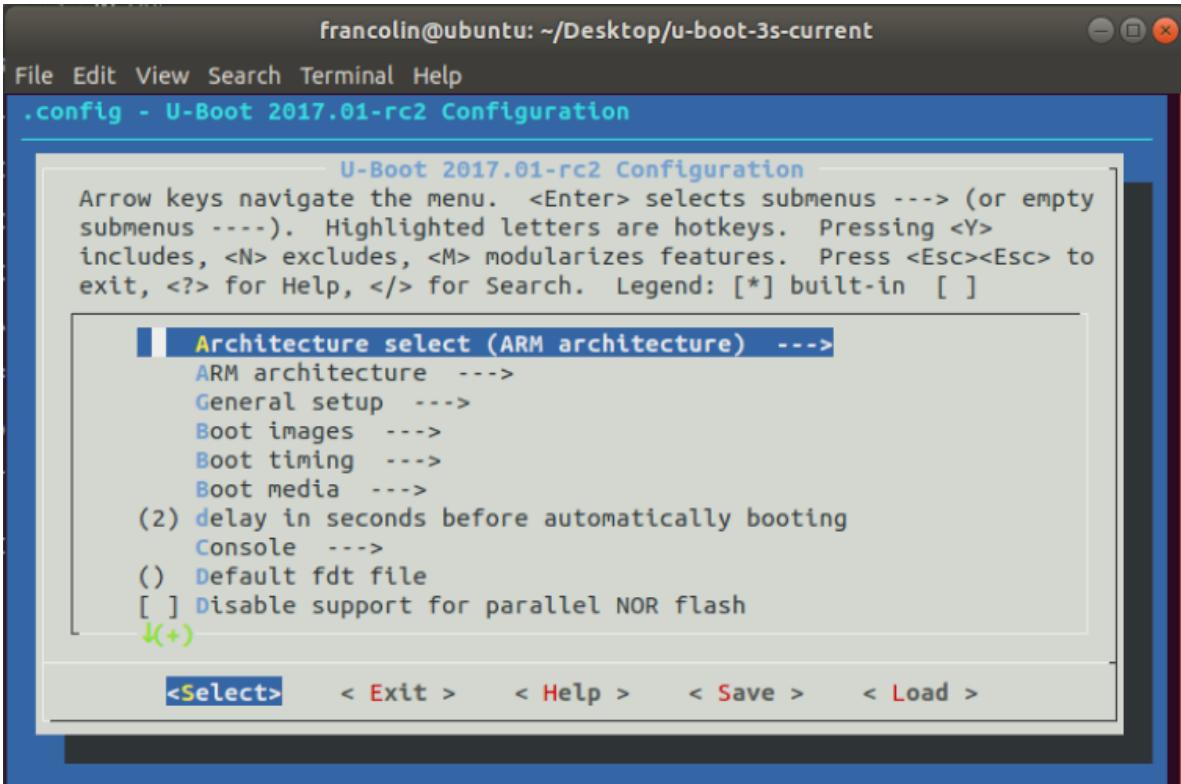
```
sudo ARCH=arm make menuconfig
```

```

francolin@ubuntu:~/Desktop/u-boot-3s-current$ vim include/configs/sun8i.h
francolin@ubuntu:~/Desktop/u-boot-3s-current$ sudo ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make LicheePi_Zero_defconfig
#
# configuration written to .config
#
francolin@ubuntu:~/Desktop/u-boot-3s-current$ sudo ARCH=arm make menuconfig

```

按下回车后会跳出图形配置界面，此次demo按默认即可，所以直接exit（键盘上下键控制上侧蓝色选中框上下移动，左右键控制下侧蓝色选中框左右移动，回车键选中）



当然也可以自己更改，更改完记得save完再exit。

4):开始编译：

```
sudo ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make
```

编译完成如下图：

```
francolin@ubuntu: ~/Desktop/u-boot-3s-current
File Edit View Search Terminal Help
CC      spl/drivers/serial/serial.o
CC      spl/drivers/serial/serial_ns16550.o
CC      spl/drivers/serial/ns16550.o
LD      spl/drivers/serial/built-in.o
LD      spl/drivers/built-in.o
LD      spl/dts/built-in.o
LD      spl/fs/built-in.o
LDS     spl/u-boot-spl.lds
LD      spl/u-boot-spl
OBJCOPY spl/u-boot-spl-nodtb.bin
COPY    spl/u-boot-spl.bin
MKSUNXI spl/sunxi-spl.bin
OBJCOPY u-boot-nodtb.bin
CAT    u-boot-dtb.bin
COPY    u-boot.bin
MKIMAGE u-boot.img
COPY    u-boot.dtb
BINMAN u-boot-sunxi-with-spl.bin
OBJCOPY u-boot.srec
SYM    u-boot.sym
MKIMAGE u-boot-dtb.img
./scripts/check-config.sh u-boot.cfg \
    ./scripts/config_whitelist.txt . 1>&2
francolin@ubuntu:~/Desktop/u-boot-3s-current$ S
```

注意生成了一个bin文件

```
francolin@ubuntu: ~/Desktop/u-boot-3s-current
File Edit View Search Terminal Help
OBJCOPY u-boot-nodtb.bin
CAT    u-boot-dtb.bin
COPY    u-boot.bin
MKIMAGE u-boot.img
COPY    u-boot.dtb
BINMAN u-boot-sunxi-with-spl.bin
OBJCOPY u-boot.srec
SYM    u-boot.sym
MKIMAGE u-boot-dtb.img
./scripts/check-config.sh u-boot.cfg \
    ./scripts/config_whitelist.txt . 1>&2
francolin@ubuntu:~/Desktop/u-boot-3s-current$ ls
api      examples   README           u-boot.dtb
arch     fs         scripts          u-boot-dtb.bin
board   include    snapshot.commit  u-boot-dtb.img
cmd      Kbuild    spl              u-boot.img
common  Kconfig   System.map       u-boot.lds
config.mk lib       test             u-boot.map
configs  Licenses  tools            u-boot-nodtb.bin
disk    MAINTAINERS u-boot          u-boot.srec
doc     Makefile   u-boot.bin       u-boot-sunxi-with-spl.bin
drivers net       u-boot.cfg       u-boot.sym
dts     post      u-boot.cfg.configs
francolin@ubuntu:~/Desktop/u-boot-3s-current$
```

把他复制出来备用，这就是我们最终要用的bootloader了。

2.编译LinuxKernel

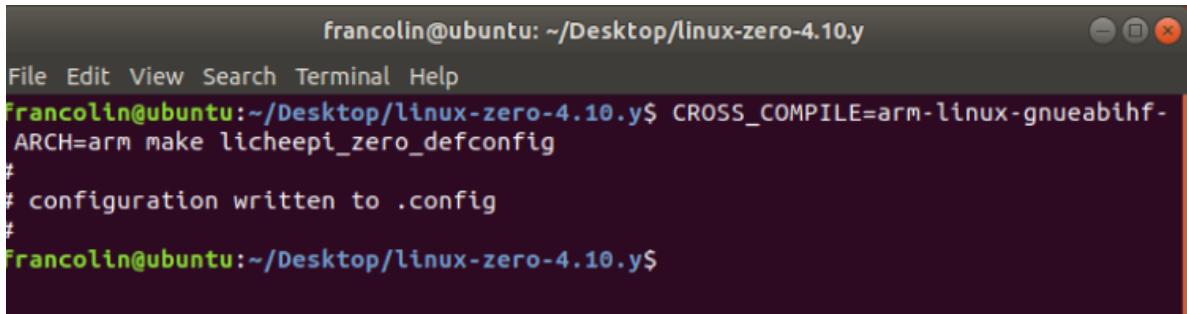
首先需要准备LinuxKernel的文件<https://github.com/Lichee-Pi/linux/archive/zero-4.10.y.zip>，推荐用4.10版本，上述链接的仓库里还存有其他4版本的内核源码，还有一个5.2的内核源码，我一开始试的是5.2的，结果后来发现设备树怎么也编译不过，不知道什么问题，只好转回4.10。

在这里我暂时先讲一下最基本的内核编译，之后再讲怎么把我们OLED屏幕放进来。

1):解压文件，进入目录

2):使用默认配置即可：

```
CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm make licheepi_zero_defconfig
```



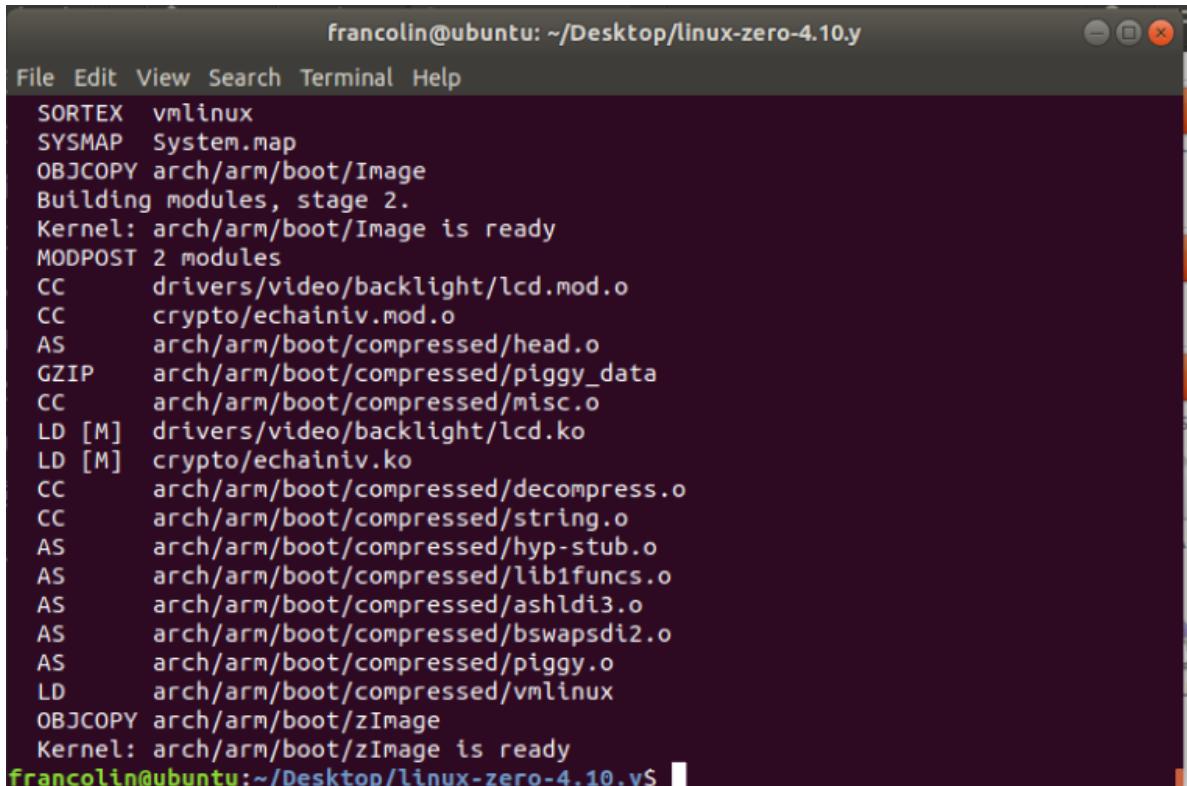
```
francolin@ubuntu: ~/Desktop/linux-zero-4.10.y
File Edit View Search Terminal Help
francolin@ubuntu:~/Desktop/linux-zero-4.10.y$ CROSS_COMPILE=arm-linux-gnueabihf-
ARCH=arm make licheepi_zero_defconfig
#
# configuration written to .config
#
francolin@ubuntu:~/Desktop/linux-zero-4.10.y$
```

3):开始编译内核：

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4
```

注意，内核编译会比较久。

编译完成如下：



```
francolin@ubuntu: ~/Desktop/linux-zero-4.10.y
File Edit View Search Terminal Help
SORTEX vmlinux
SYSMAP System.map
OBJCOPY arch/arm/boot/Image
Building modules, stage 2.
Kernel: arch/arm/boot/Image is ready
MODPOST 2 modules
CC      drivers/video/backlight/lcd.mod.o
CC      crypto/echainiv.mod.o
AS      arch/arm/boot/compressed/head.o
GZIP    arch/arm/boot/compressed/piggy_data
CC      arch/arm/boot/compressed/misc.o
LD [M]  drivers/video/backlight/lcd.ko
LD [M]  crypto/echainiv.ko
CC      arch/arm/boot/compressed/decompress.o
CC      arch/arm/boot/compressed/string.o
AS      arch/arm/boot/compressed/hyp-stub.o
AS      arch/arm/boot/compressed/lib1funcs.o
AS      arch/arm/boot/compressed/ashldi3.o
AS      arch/arm/boot/compressed/bswapsdi2.o
AS      arch/arm/boot/compressed/piggy.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
francolin@ubuntu:~/Desktop/linux-zero-4.10.y$
```

编译好的内核镜像在/arch/arm/boot下，文件名叫zImage，将其取出备用。

4):设备树的编译：

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
```

完成如下图：

```
francolin@ubuntu: ~/Desktop/linux-zero-4.10.y
File Edit View Search Terminal Help
DTC      arch/arm/boot/dts/sun8i-a23-q8-tablet.dtb
DTC      arch/arm/boot/dts/sun8i-a33-et-q8-v1.6.dtb
DTC      arch/arm/boot/dts/sun8i-a33-ga10h-v1.1.dtb
DTC      arch/arm/boot/dts/sun8i-a33-inet-d978-rev2.dtb
DTC      arch/arm/boot/dts/sun8i-a33-ippo-q8h-v1.2.dtb
DTC      arch/arm/boot/dts/sun8i-a33-olinuxino.dtb
DTC      arch/arm/boot/dts/sun8i-a33-q8-tablet.dtb
DTC      arch/arm/boot/dts/sun8i-a33-sinlinx-sina33.dtb
DTC      arch/arm/boot/dts/sun8i-a83t-allwinner-h8homlet-v2.dtb
DTC      arch/arm/boot/dts/sun8i-a83t-cubietruck-plus.dtb
DTC      arch/arm/boot/dts/sun8i-h3-bananapi-m2-plus.dtb
DTC      arch/arm/boot/dts/sun8i-h3-nanopi-m1.dtb
DTC      arch/arm/boot/dts/sun8i-h3-nanopi-neo.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-2.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-lite.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-one.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-pc.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-pc-plus.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-plus.dtb
DTC      arch/arm/boot/dts/sun8i-h3-orangepi-plus2e.dtb
DTC      arch/arm/boot/dts/sun8i-r16-parrot.dtb
DTC      arch/arm/boot/dts/sun8i-v3s-licheepi-zero.dtb
DTC      arch/arm/boot/dts/sun8i-v3s-licheepi-zero-dock.dtb
francolin@ubuntu:~/Desktop/linux-zero-4.10.y$
```

编译好的设备树文件在/arch/arm/boot/dts文件夹下名为sun8i-v3s-licheepi-zero-dock.dtb，取出备用。注意不要取错了，还有一个名字很相近的是sun8i-v3s-licheepi-zero.dtb。

3.Buildroot编译

首先准备uboot文件<https://buildroot.org/downloads/buildroot-2017.08.1.tar.gz>

1):解压文件，进入目录

2):配置

```
make menuconfig
```

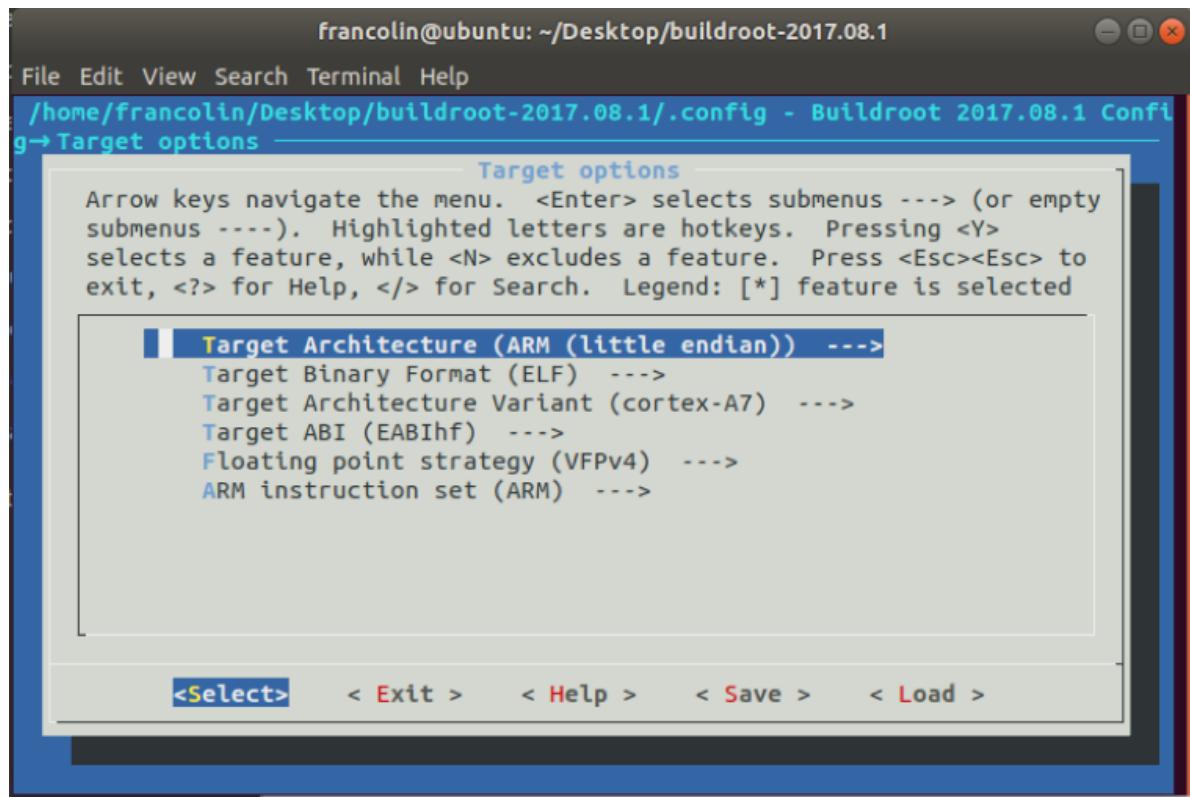
具体配置如下：

Target Options->Target Architecture 设置为 ARM(little endian)

设置完Target Architecture后Target Options页面的选项也会发生一些改变

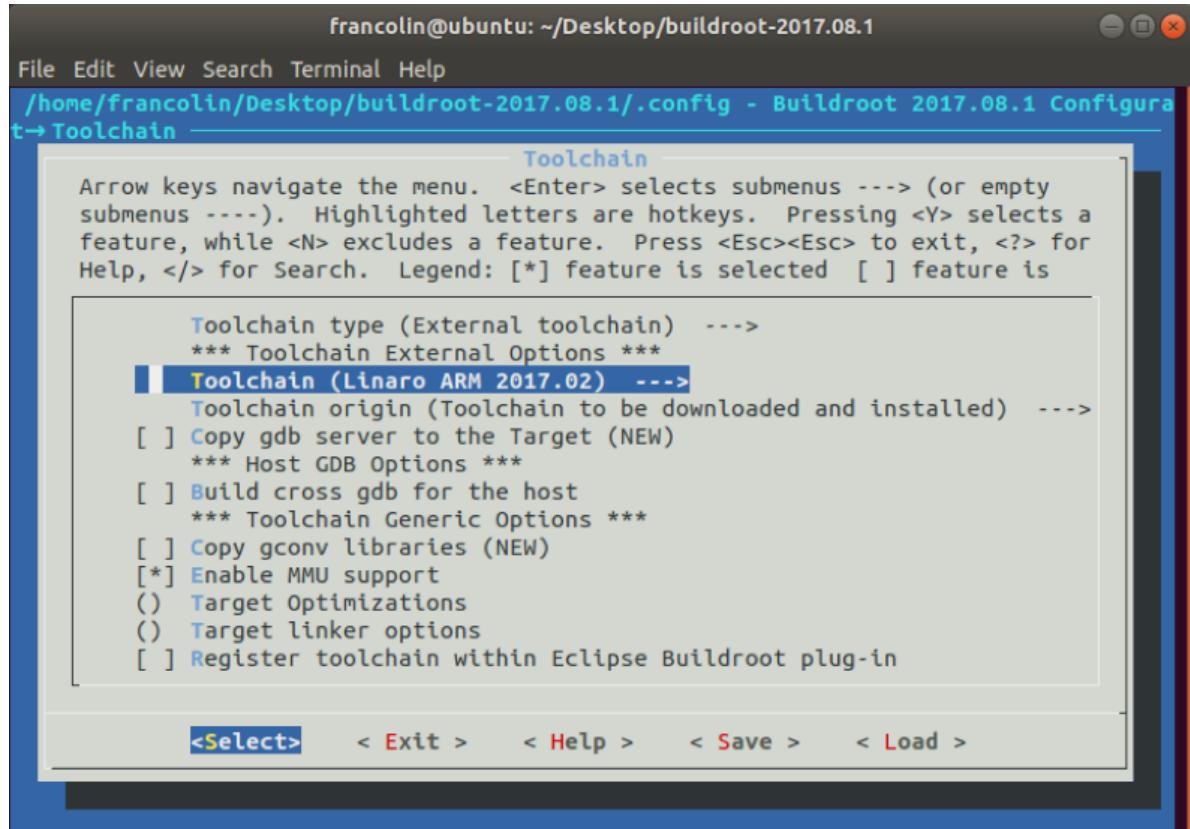
然后将Target Architecture Variant 设置为cortex-A7

Floating point strategy设置为VFPv4，具体如下图



exit返回上一级菜单

将Toolchain->Toolchain Type设置为External toolchain,具体如下图



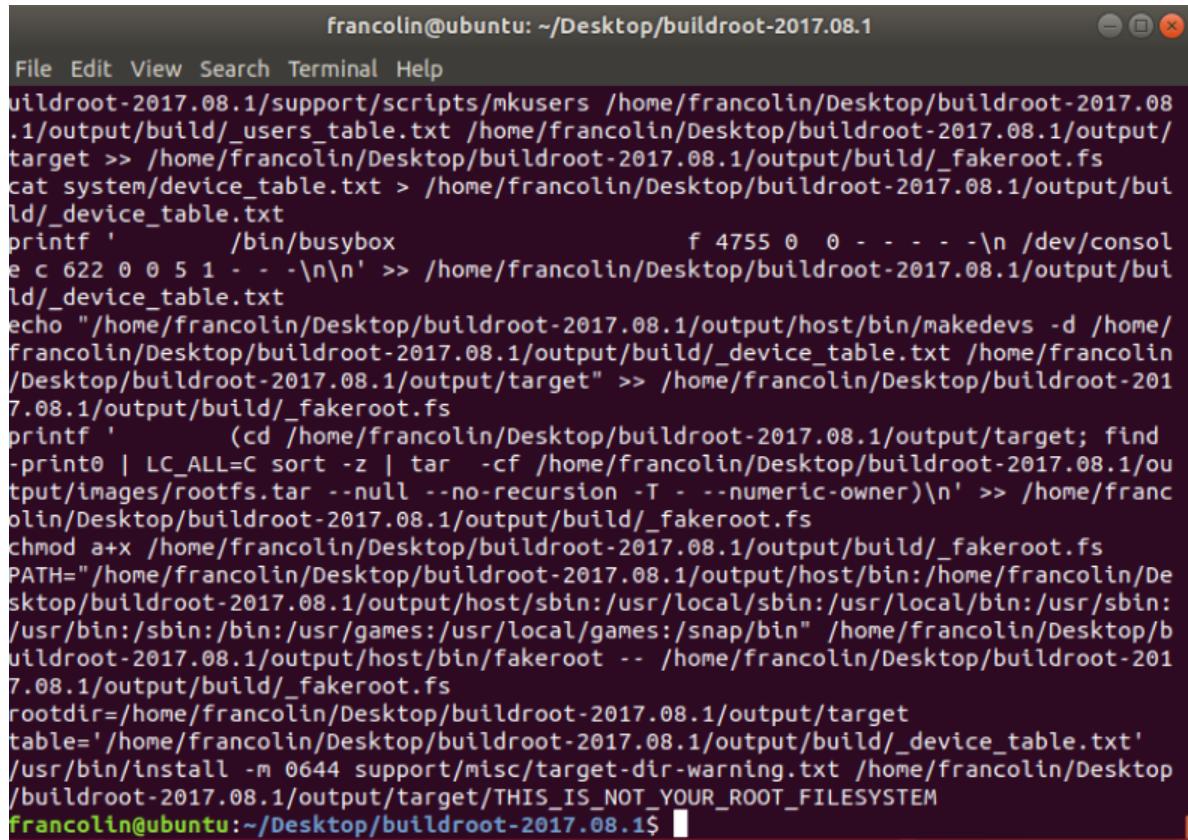
然后保存退出

3):编译

```
make
```

由于会从网上下载一些东西，此过程根据大家网速不同执行速度也不同。

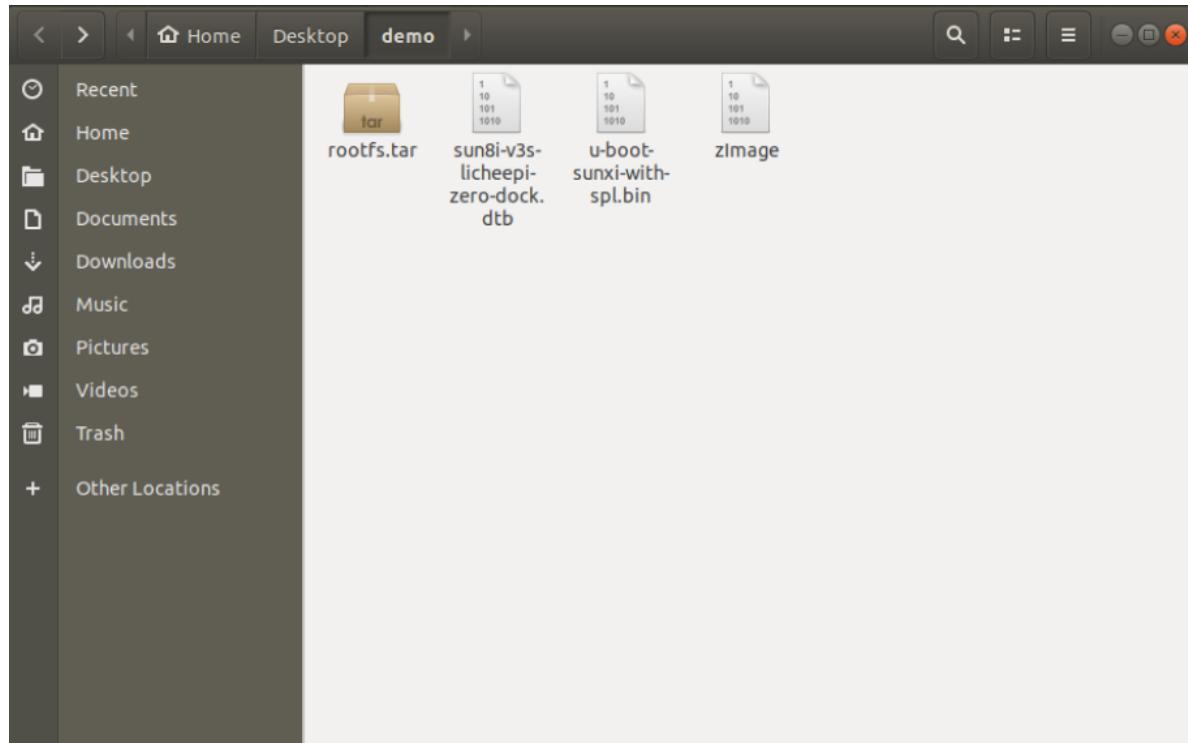
编译完成如下图：



```
francolin@ubuntu: ~/Desktop/buildroot-2017.08.1
File Edit View Search Terminal Help
buildroot-2017.08.1/support/scripts/mkusers /home/francolin/Desktop/buildroot-2017.08.1/output/build/_users_table.txt /home/francolin/Desktop/buildroot-2017.08.1/output/target >> /home/francolin/Desktop/buildroot-2017.08.1/output/build/_fakeroot.fs
cat system/device_table.txt > /home/francolin/Desktop/buildroot-2017.08.1/output/build/_device_table.txt
printf '/bin/busybox' f 4755 0 0 - - - -\n /dev/console c 622 0 0 5 1 - - -\n\n' >> /home/francolin/Desktop/buildroot-2017.08.1/output/build/_device_table.txt
echo "/home/francolin/Desktop/buildroot-2017.08.1/output/host/bin/makedevs -d /home/francolin/Desktop/buildroot-2017.08.1/output/build/_device_table.txt /home/francolin/Desktop/buildroot-2017.08.1/output/target" >> /home/francolin/Desktop/buildroot-2017.08.1/output/build/_fakeroot.fs
printf '(cd /home/francolin/Desktop/buildroot-2017.08.1/output/target; find -print0 | LC_ALL=C sort -z | tar -cf /home/francolin/Desktop/buildroot-2017.08.1/output/images/rootfs.tar --null --no-recursion -T - --numeric-owner)\n' >> /home/francolin/Desktop/buildroot-2017.08.1/output/build/_fakeroot.fs
chmod a+x /home/francolin/Desktop/buildroot-2017.08.1/output/build/_fakeroot.fs
PATH="/home/francolin/Desktop/buildroot-2017.08.1/output/host/bin:/home/francolin/Desktop/buildroot-2017.08.1/output/host/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin" /home/francolin/Desktop/buildroot-2017.08.1/output/host/bin/fakeroot -- /home/francolin/Desktop/buildroot-2017.08.1/output/build/_fakeroot.fs
rootdir=/home/francolin/Desktop/buildroot-2017.08.1/output/target
table='/home/francolin/Desktop/buildroot-2017.08.1/output/build/_device_table.txt'
/usr/bin/install -m 0644 support/misc/target-dir-warning.txt /home/francolin/Desktop/buildroot-2017.08.1/output/target/THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
francolin@ubuntu:~/Desktop/buildroot-2017.08.1$
```

会在/output/images下生成rootfs文件，取出备用。

至此所有的编译工作全部完成，我们现在应当有四个文件：



接下来我们要开始烧录TF卡的工作，把系统真正的烧进去。

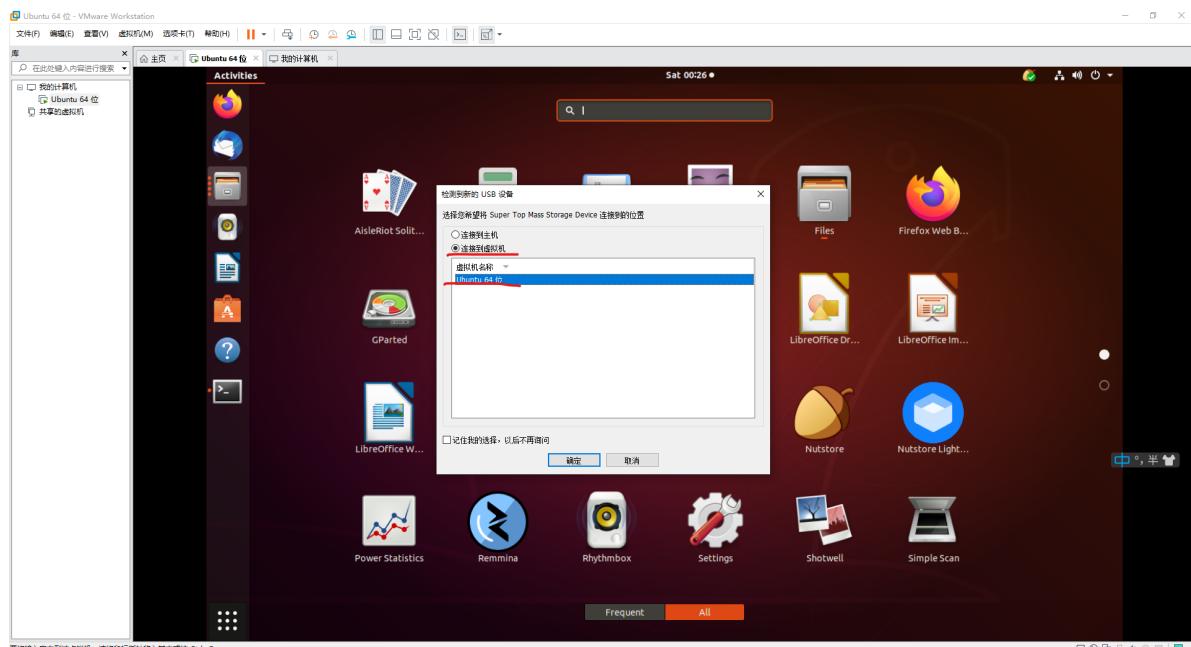
4.TF卡准备：

首先我们在linux系统上装一个软件Gparted

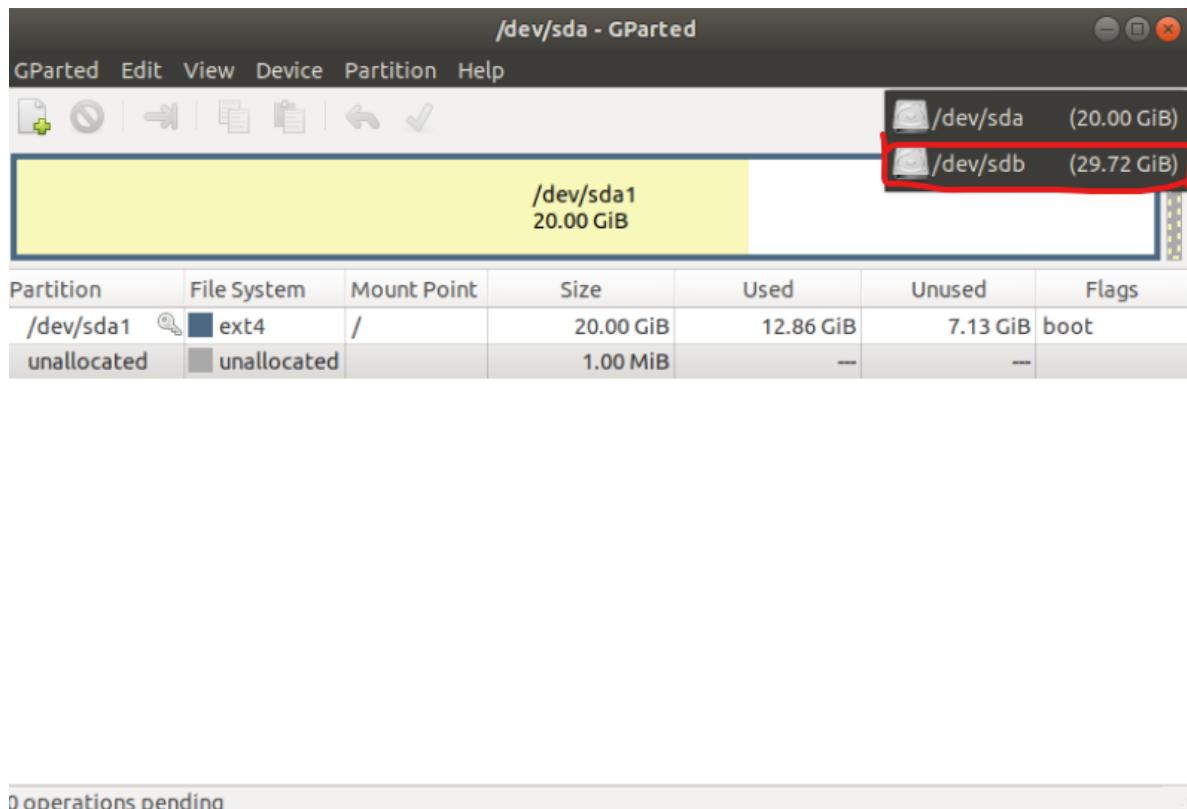
```
sudo apt-get install gparted
```



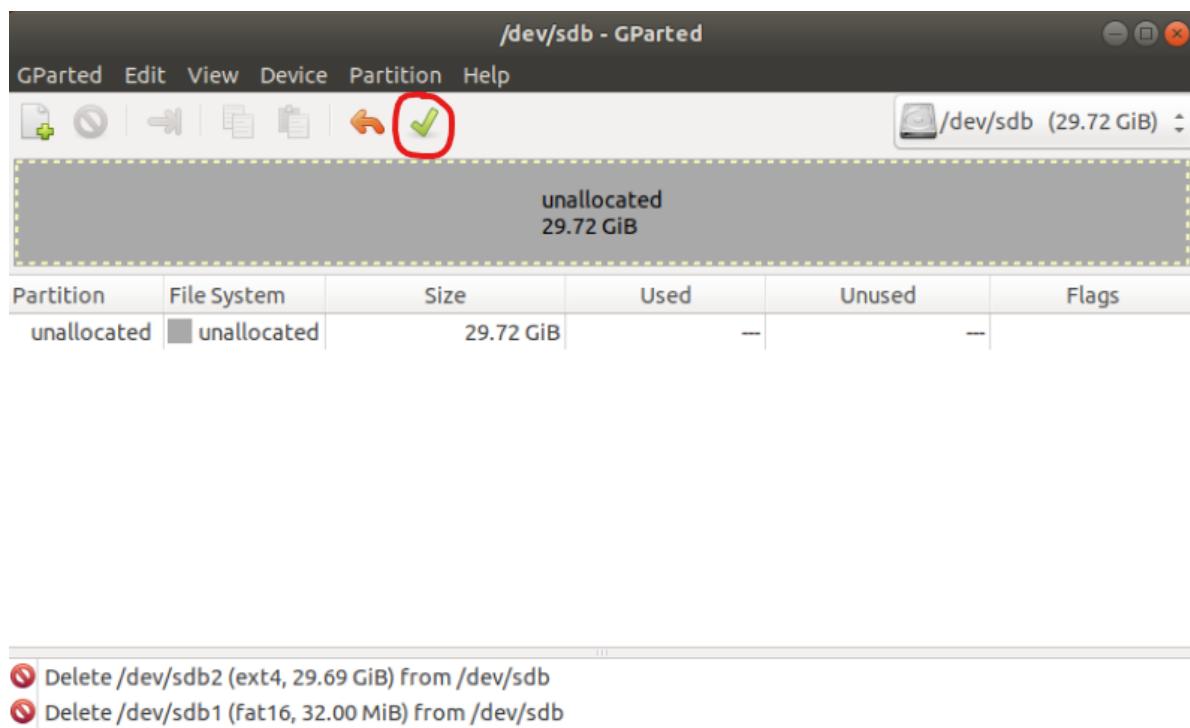
将TF卡插到读卡器上，并插入电脑，如果你是用的是虚拟机的话，那么虚拟机软件可能会询问你是将USB设备挂载到虚拟机上还是挂载到主机上，由于我们是在虚拟机上执行操作，所以应当选择挂载到虚拟机上。



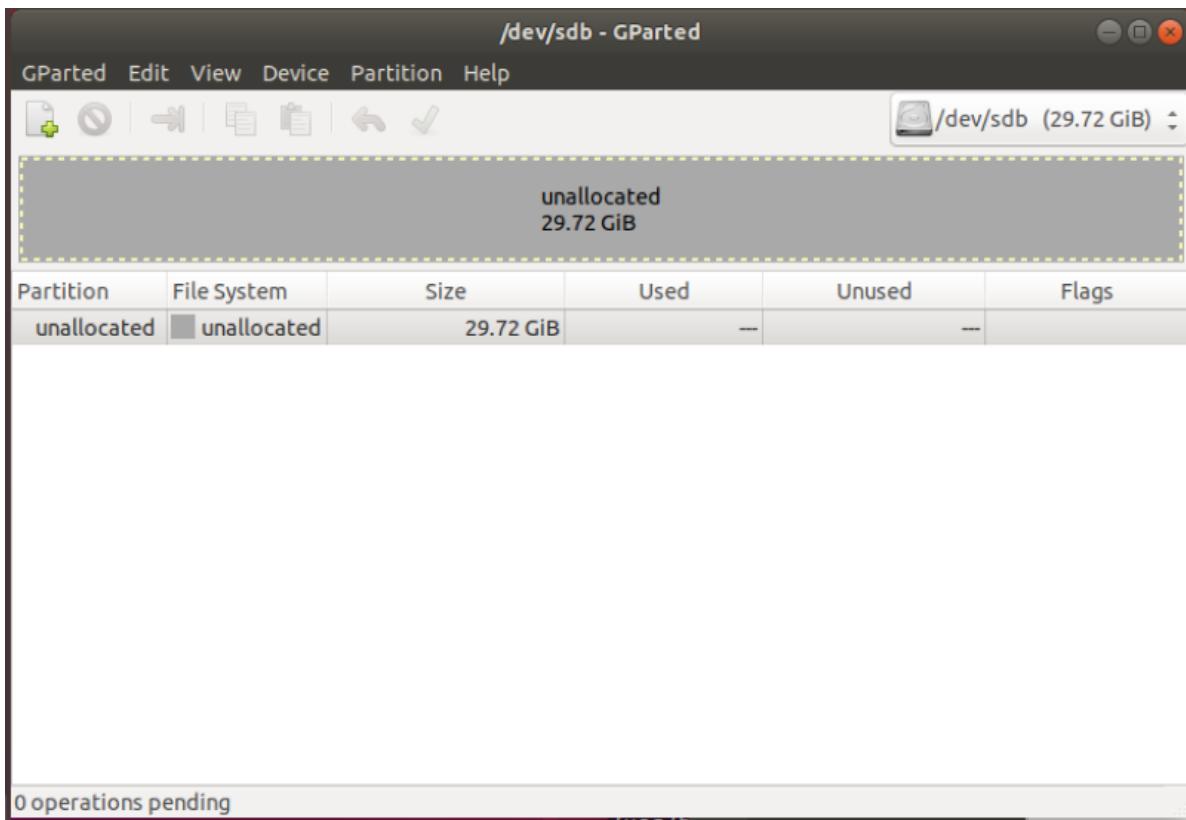
接下来我们打开Gparted软件，选择tf卡，如下图



你的盘可能事先有多个分区，我们要做的是先把他们先全部卸载(unmount)，然后全部删除(delete)（只要在每个分区选中后右击鼠标就能看到这些选项了）。执行完如下图，整个盘会变成unallocated，接着我们点绿色勾勾，保存当前操作。

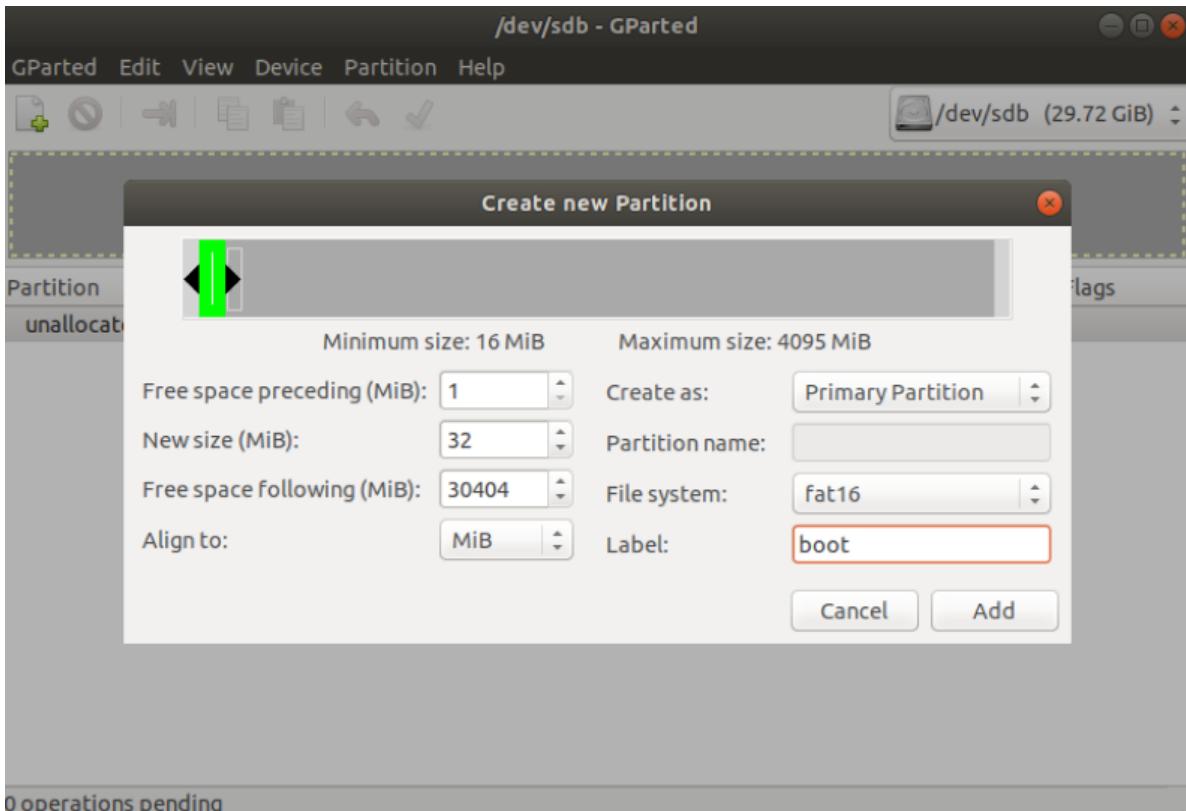


完成后如下：

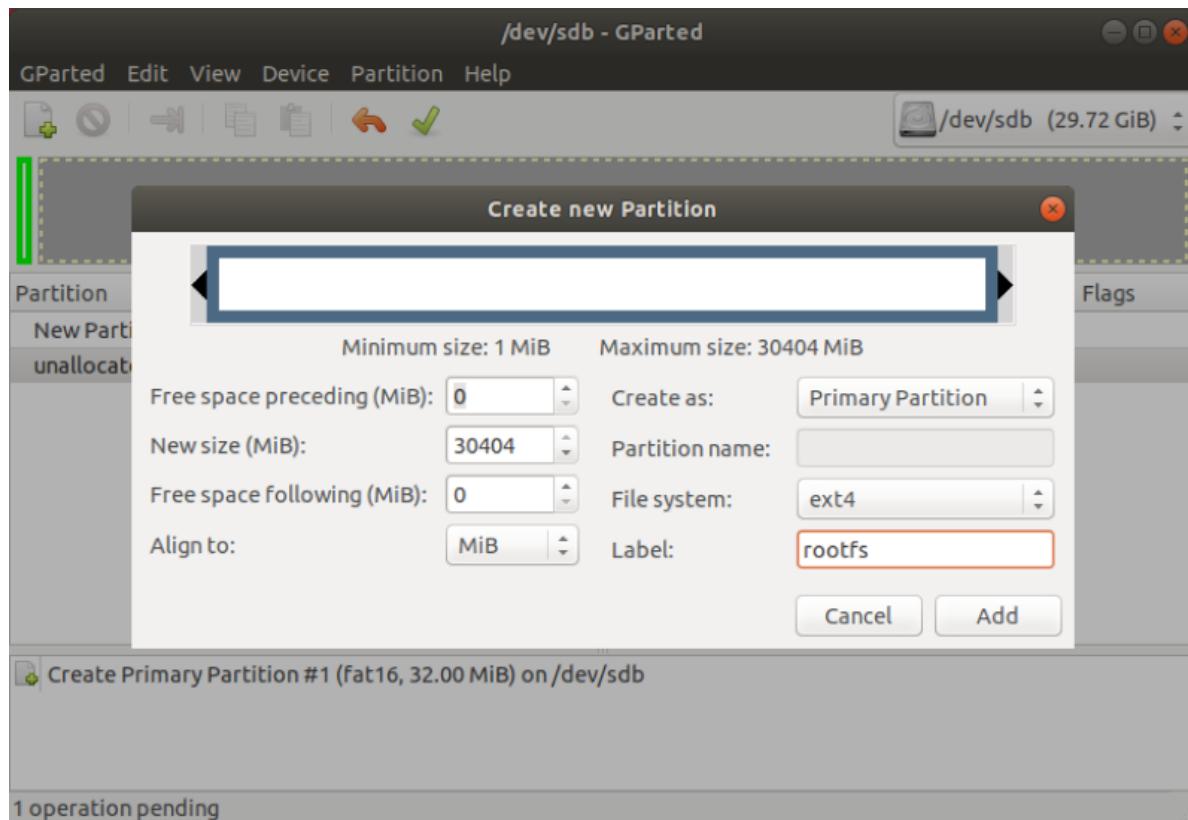


接下来我们重新为其分配：

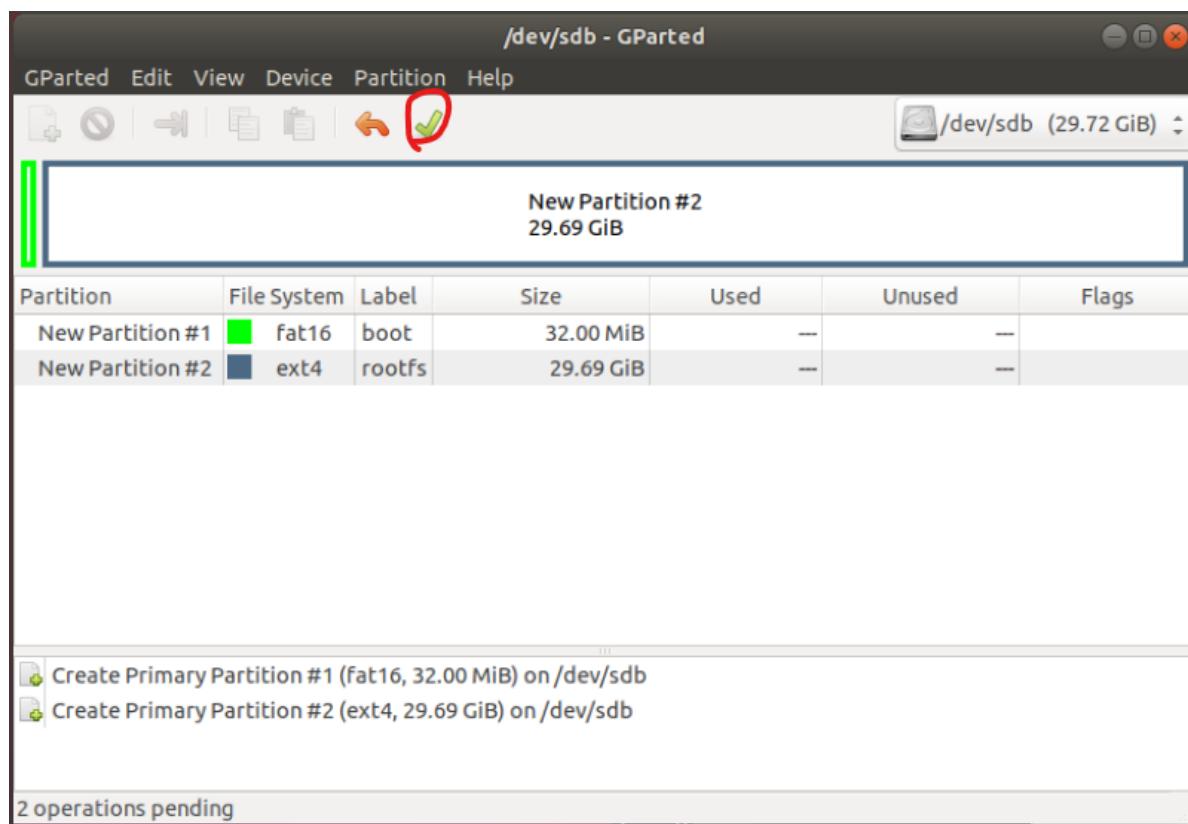
点击左上角有绿色加号的图标，分配一个32MB，FAT16格式，名称为boot的分区，如下图



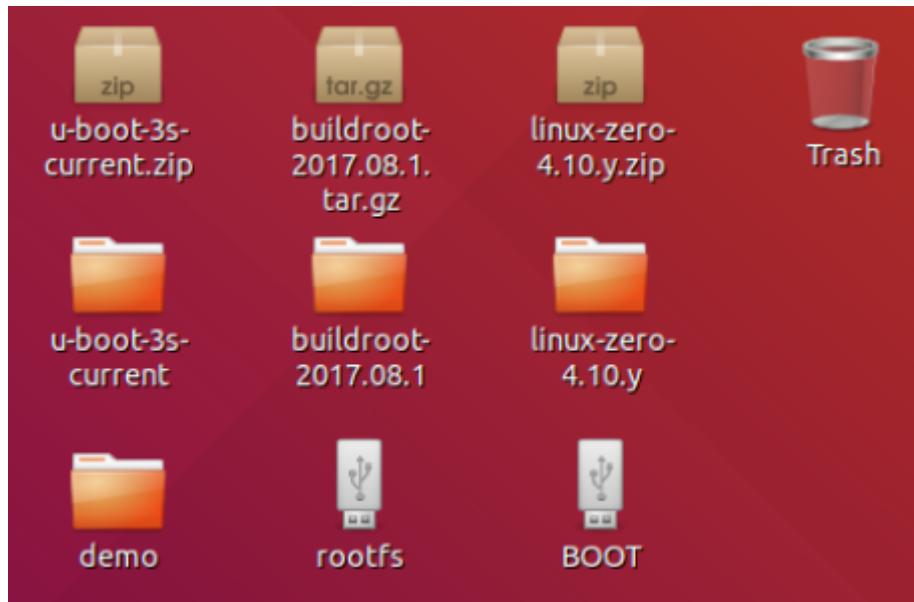
Add后把剩下的空间都分配为一个ext4，名称为rootfs的分区，如下图：



Add后点击小绿√保存



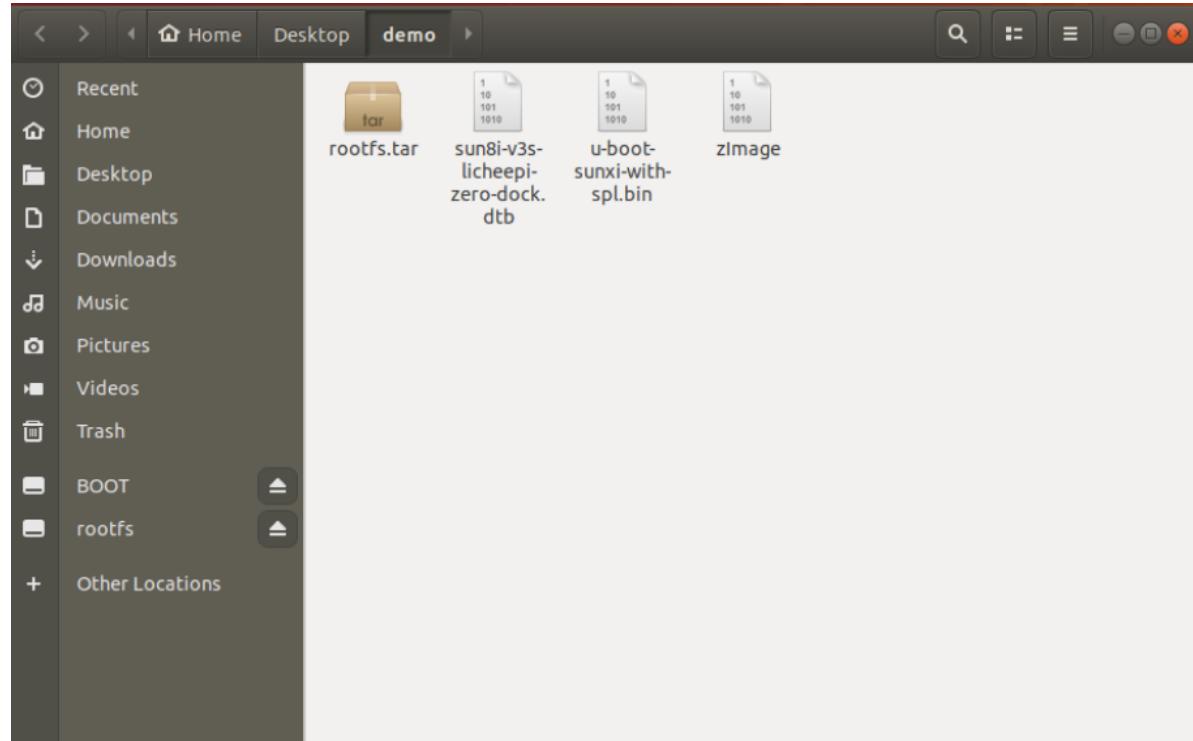
等待期操作完后，将读卡器拔出电脑，再插回去，可以看到已经变成两个我们所命名盘挂载到电脑了。



两个盘名字顾名思义，boot是放启动文件的，rootfs是放根文件系统的，boot为fat16，rootfs为ext4，这也是为什么大家之前在玩树莓派等等各种卡片Linux电脑时，把镜像烧到sd卡里后再插上电脑，发现sd卡只有几百甚至几十MB了，就是因为windows资源管理器无法访问ext4格式的存储设备。

5.TF卡烧写：

下面的操作，我的工作目录结构如下图



1.烧写u-boot

首先将编译uboot产生的bin文件烧写到TF卡8k偏移处(必须8K偏移, brom规定的)

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

成功如下：

```
francolin@ubuntu: ~/Desktop/demo
File Edit View Search Terminal Help
francolin@ubuntu:~/Desktop/demo$ sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/sdb bs=1024 seek=8
[sudo] password for francolin:
383+1 records in
383+1 records out
393143 bytes (393 kB, 384 KiB) copied, 0.299128 s, 1.3 MB/s
francolin@ubuntu:~/Desktop/demo$ S
```

这个/dev/sdb 指的是TF卡，你的可能是sdb也有可能是sdc，主要还是用命令 fdisk -l 来查看

```
Disk /dev/sdb: 29.7 GiB, 31914983424 bytes, 62333952 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xda2dbbf

Device      Boot Start    End   Sectors  Size Id Type
/dev/sdb1        2048    67583    65536   32M  6 FAT16
/dev/sdb2       67584 62332927 62265344 29.7G 83 Linux
francolin@ubuntu:~/Desktop/demo$ S
```

可见我的是sdb。

2.拷贝zImage,dtb

将zImage和.dtb文件拷贝到BOOT分区中：

```
sudo cp zImage sun8i-v3s-limepi-zero-dock.dtb /media/francolin/BOOT/
```

这个命令中的目标地址写/media/francolin/BOOT/是因为U盘会被自动挂载在media下的用户目录下，**我的用户名为francolin，大家在使用的时候只要替换自己的用户名即可**(当然图形化操作也是可以的)。

3.解压rootfs

把rootfs.tar这个文件直接解压到rootfs这个分区下

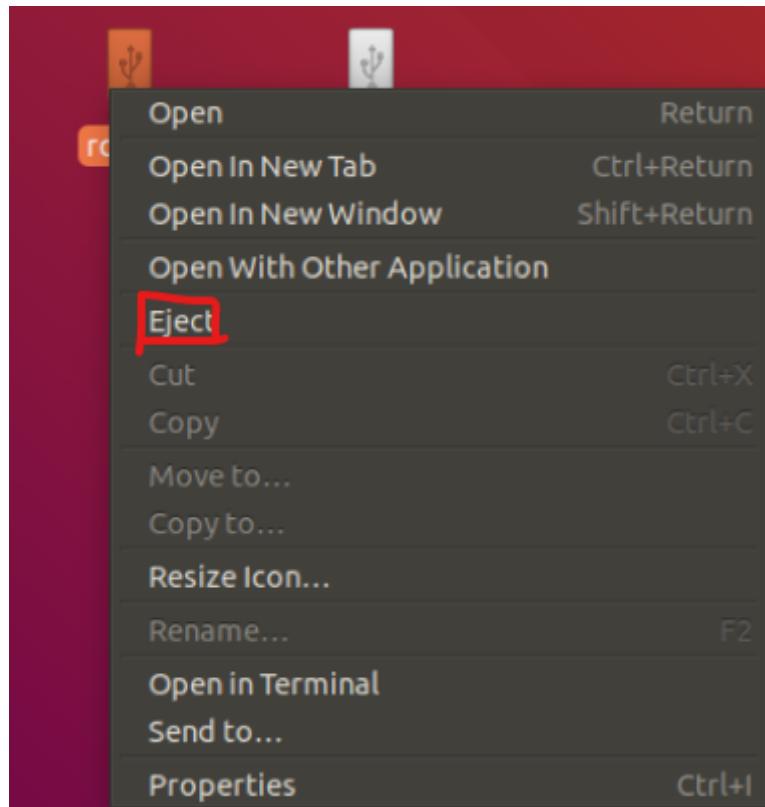
```
sudo tar xvf rootfs.tar -c /media/francolin/rootfs/
```

一样的，需要替换你自己的用户名。

4.弹出U盘

这一步非常非常非常非常容易忽略，我在刚做的时候，总是不在系统里弹出，而直接拔u盘，导致的问题就是各种各样且非常非常难以定位问题源头，太坑了。

只要在boot或者rootfs任意图标上右击——>eject即可，待桌面上这两个分区消失后，方可拔下u盘。

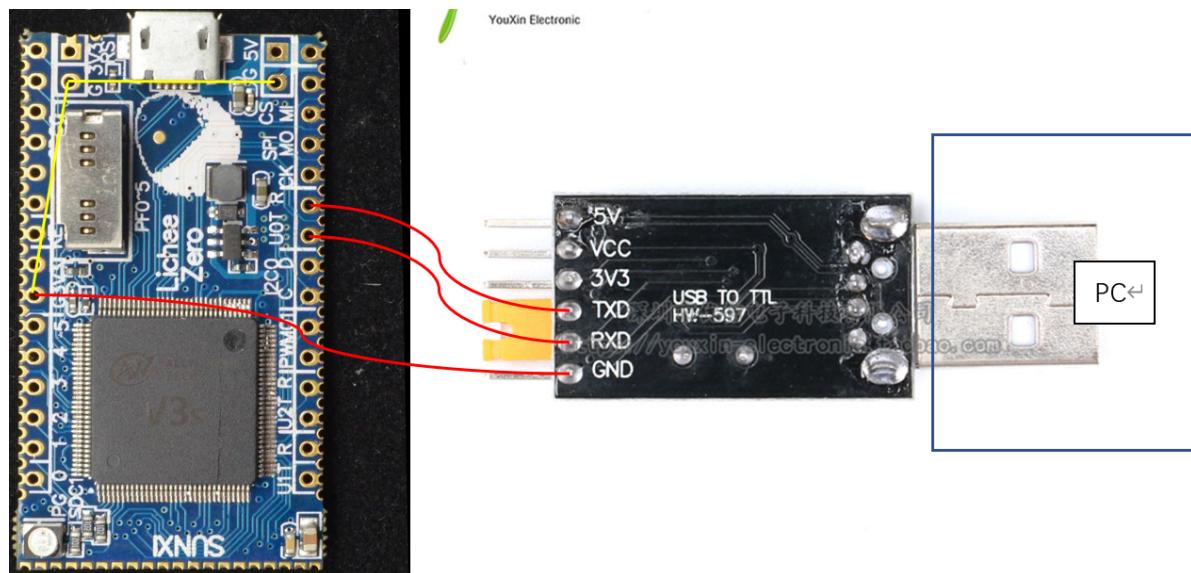


将TF卡插到板子的卡槽里，至此，如果正常的话，系统已经制作完成。

5. 测试：

接下来，大家最好将板子焊好排针，这样才好调试。

焊好后，首先用杜邦线连接两设备如下图：

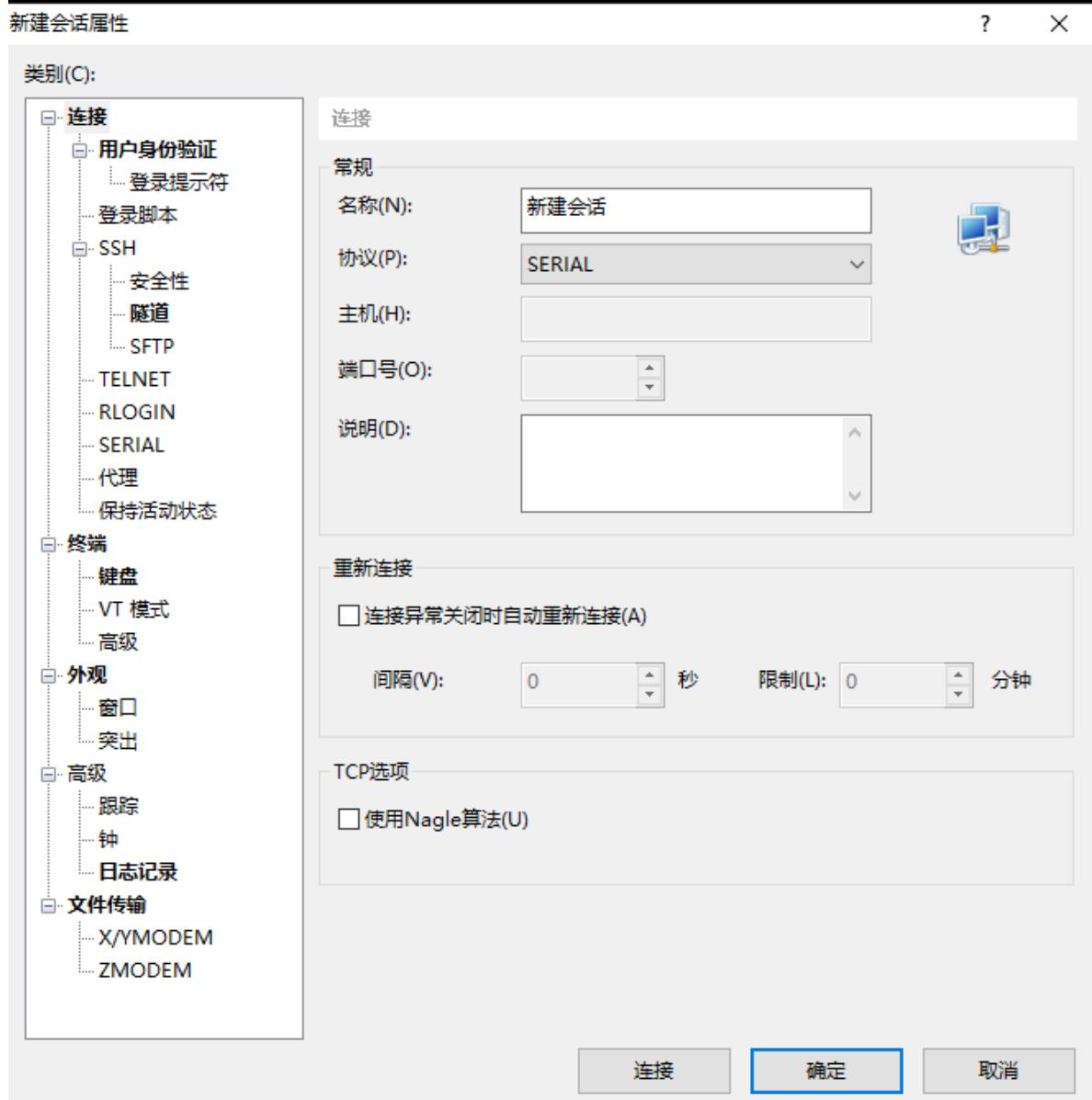


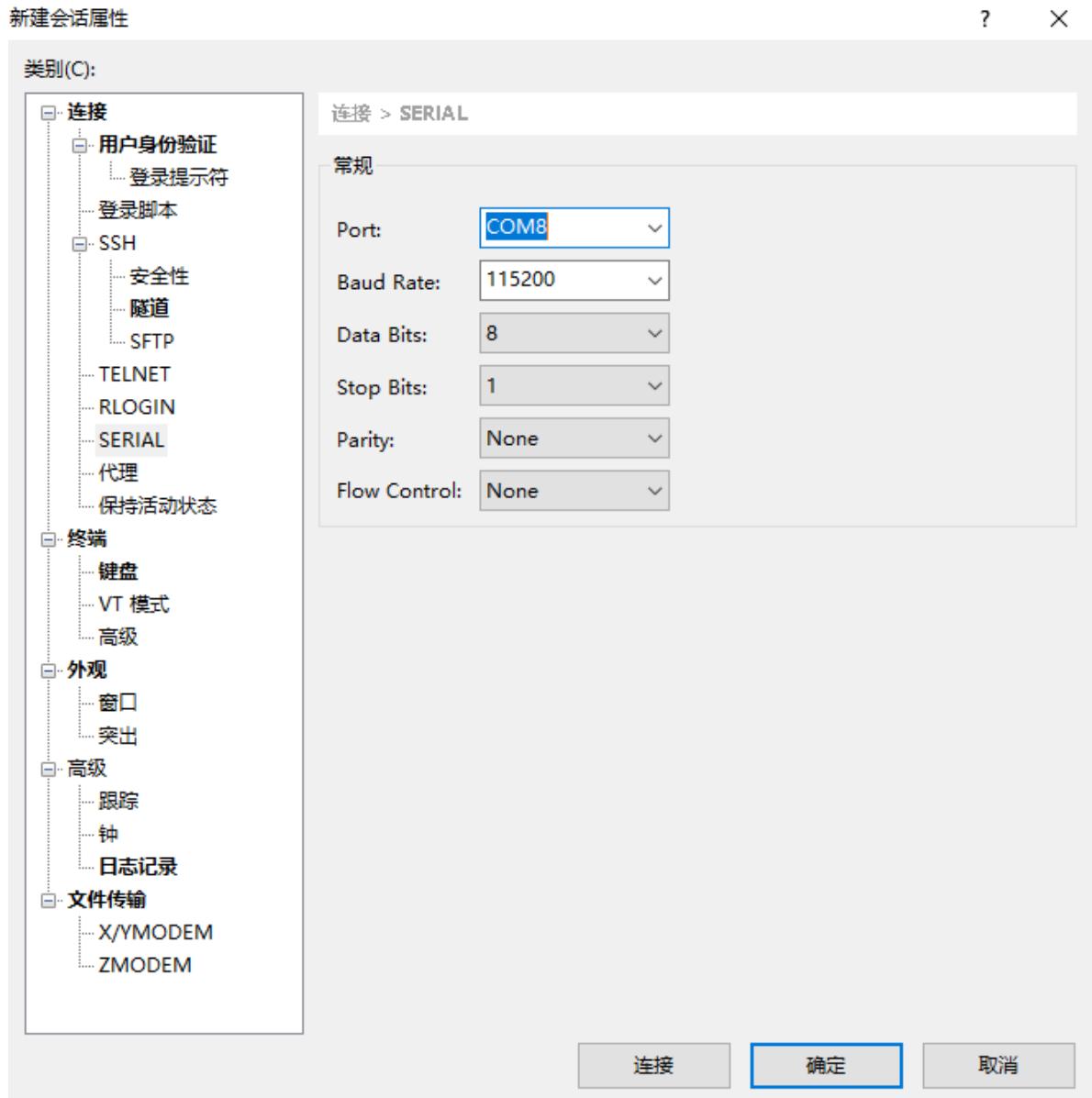
如上图Zero

USB转串口模块的TXD(发送)要接Zero的R(注意是Uart0的R端，板上U0表示Uart0，正是默认的输出终端)如果对这部分不懂的话可以去搜一下串口通信)，RXD(接收)要接Zero的T，另外，两设备还需要共地，这一点很重要，图中Zero上两条黄线相连的三个点都是地，只要接其中任意一个即可。

如图连接好后，插上电脑，打开Xshell

新建会话，协议选Serial：





```

In:  serial@01c28000
Out: serial@01c28000
Err: serial@01c28000
Net: No ethernet found.
starting USB...
No controllers found
Hit any key to stop autoboot: 0
reading zImage
3816728 bytes read in 197 ms (18.5 MiB/s)
reading uInit-v3s-licheepi-zero-dock.dtb
9330 bytes read in 25 ms (364.3 KiB/s)
## Flattened Device Tree blob at 41800000
Booting using the fdt blob at 0x41800000
Loading Device Tree to 42dFa000, end 42dff471 ... OK
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.10.15-licheepi-zero (francolin@ubuntu) (gcc version 7.5.0 (Ubuntu/Linaro 7.5.0-3ubuntu1-18.04) ) #1 SMP Wed Jan 20 00:50:08 PST 2021
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
[    0.000000] CPU: div instructions available; patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] OF: fdt:Machine model: Lichee Pi Zero with Dock
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] percpu: Embedded 14 pages/cpu @c3f63000 s24716 r8192 d24436 u57344
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
[    0.000000] Kernel command line: console=ttyS0,115200 panic=5 rootwait root=/dev/mmcblk0p2 earlyprintk rw vt.global_cursor_default=0
[    0.000000] PID hash table entries: 256 (order: -2, 1024 bytes)
[    0.000000] Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
[    0.000000] Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
[    0.000000] Memory: 55152K/65536K available (6144K kernel code, 199K rdata, 1388K rodata, 1024K init, 259K bss, 10384K reserved, 0K cma-reserved, 0K highmem)
[    0.000000] Virtual kernel memory layout:
[    0.000000]   vector : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]   fixmap : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]   vmalloc : 0x48000000 - 0xffff0000   ( 944 MB)
[    0.000000]   lowmem : 0xc0000000 - 0x40000000   ( 64 MB)
[    0.000000]   pkmapping : 0xdfe00000 - 0x00000000   (   2 kB)
[    0.000000]   modules : 0xdf000000 - 0xdfe00000   ( 14 kB)
[    0.000000]   .text : 0xc0000000 - 0xc0700000   (7136 kB)
[    0.000000]   .init : 0xc0900000 - 0xc0a00000   (1024 kB)
[    0.000000]   .data : 0xc0a00000 - 0xc0a31e40   ( 200 kB)
[    0.000000]   .bss : 0xc0a33000 - 0xc0a73f00   ( 260 kB)
[    0.000000] SLUB: Walign=64, Order=-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] Hierarchical RCU implementation.
[    0.000000] Build-time adjustment of leaf fanout to 32.
[    0.000000] RCU: Restricting CPUs from NR_CPUS=8 to nr_cpu_ids=1.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=1
[    0.000000] NR_IRQS:16 nr_irqs:16
[    0.000000] arm_arch_timer: Architected cp15 timer(s) running at 24.00MHz (virt).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffff max_cycles: 0x588fe9dc0, max_idle_ns: 440795202592 ns
[    0.000007] sched_clock: 56 bits at 24MHz, resolution 41ns
[    0.000138] clocksource: timer: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 79635851949 ns
[    0.000356] Console: colour dummy device 80x30
[    0.000795] console [tty1] enabled
[    0.000835] Calibrating delay loop (skipped), value calculated using timer frequency.. 48

```

```

Starting logging: OK
Initializing random number generator... done.
Starting network: OK

Welcome to Buildroot
buildroot login: root
# echo Hello World!
Hello World!
#

```

登陆成功，echo成功，恭喜你！系统制作成功！！

但是当我们试图像在PC上那样apt-get install时.....

咳咳，这根本不可能完成，一，我们没有给它配任何网络驱动，二，buildroot生成的根文件系统太单薄了，没有包管理。

于是乎，我决定继续折腾，至少有个包管理！

进阶之安装Debian文件系统

1.准备工作：

需要安装这样两个东西

```

sudo apt install qemu-user-static -y
sudo apt install debootstrap -y

```

2.mkdir rootfs

3.在当前目录debootstrap

```

debootstrap --foreign --verbose --arch=armhf stretch rootfs
http://ftp2.cn.debian.org/debian

```

看着它开始下载那些我们耳熟能详的软件，就知道已经离成功不远了！

完成后发现rootfs文件夹里被塞满了文件

4.

```
cd rootfs
sudo mount --bind /dev dev/
sudo mount --bind /sys sys/
sudo mount --bind /proc proc/
sudo mount --bind /dev/pts dev/pts/
cd ..
sudo cp /usr/bin/qemu-arm-static rootfs/usr/bin/
sudo chmod +x rootfs/usr/bin/qemu-arm-static
```

5.解压

```
sudo LC_ALL=C LANGUAGE=C LANG=C chroot rootfs /debootstrap/debootstrap --second-stage --verbose
```

这一步由于网络原因，可能在下图所示这一小步报错，可以多试几次/隔一段时间来试/换个网络/科学上网。

成功如下图：

```
francolin@ubuntu: ~/Desktop
File Edit View Search Terminal Help
: Configuring libiptc0:armhf...
: Configuring iproute2...
: Configuring isc-dhcp-client...
: Configuring libhogweed4:armhf...
: Configuring iptables...
: Configuring libtext-wrapi18n-perl...
: Configuring libp11-kit0:armhf...
: Configuring apt...
: Configuring pinentry-curses...
: Configuring libnewt0.52:armhf...
: Configuring gnupg-agent...
: Configuring ifupdown...
: Configuring apt-utils...
: Configuring debconf-i18n...
: Configuring whiptail...
: Configuring gnupg...
: Configuring libgnutls30:armhf...
: Configuring wget...
: Configuring tasksel...
: Configuring tasksel-data...
: Configuring libc-bin...
: Configuring systemd...
: Base system installed successfully.
francolin@ubuntu:~/Desktop$
```

由于考虑到zero暂时没有联网，所以下载软件不方便，但是我们可以在它的文件系统里预装几个我们想要的啊！

```
sudo LC_ALL=C LANGUAGE=C LANG=C chroot rootfs
```

敲完这一行后会发现用户变成root了，如下图

```
francolin@ubuntu:~/Desktop$ sudo LC_ALL=C LANGUAGE=C LANG=C chroot rootfs
[sudo] password for francolin:
root@ubuntu:/# S
```

其实现在可以认为，当前我们所面对的系统，**它的文件系统已经换成我刚解压完的、准备放到zero里的根文件系统，我们现在再执行apt-get等任何命令就相当于对那个系统进行操作了**

为了方便，我就装了个gcc，装了个vim，设置了root账户的登录密码（否则会无法登入系统）。

执行完你想要的操作后可以exit退出，返回自己的文件系统。

```
exit
```

```
root@ubuntu:/# exit  
exit  
francolin@ubuntu:~/Desktop$
```

```
sudo rm rootfs/usr/bin/qemu-arm-static  
sudo umount rootfs/dev dev/  
sudo umount rootfs/sys sys/  
sudo umount rootfs/proc proc/  
sudo umount rootfs/dev/pts dev/pts/
```

接下来是把这个文件系统压缩，方便保存

```
cd rootfs  
sudo tar cvzf ../debian9.9.rootfs.gz .
```

至此debian文件系统已经制作完成。至于如何使用，只要把原来tf卡rootfs下的内容删除，把上面压缩包的内容拷贝进去即可。**拔卡前记得eject！**

```
sudo rm -r /media/francolin/rootfs/*  
sudo tar xvf debian9.9.rootfs.gz -C /media/francolin/rootfs/
```

重新插上TF卡，重新上电：

```
[ OK ] Found device /dev/ttys0.  
[ OK ] Started Permit User Sessions.  
[ OK ] Started System Logging Service.  
[ OK ] Started getty on tty2-tty6 if dbus and logind are not available.  
[ OK ] Started Getty on tty6.  
[ OK ] Started Getty on tty5.  
[ OK ] Started Getty on tty4.  
[ OK ] Started Getty on tty3.  
[ OK ] Started Getty on tty2.  
[ OK ] Started Getty on ttys1.  
[ OK ] Started Serial Getty on ttys0.  
[ OK ] Reached target Login Prompts.  
[ OK ] Reached target Multi-User System.  
[ OK ] Reached target Graphical Interface.  
      Starting Update UTMP about System Runlevel Changes...  
[ OK ] Reached target Sound Card.  
[ OK ] Started Update UTMP about System Runlevel Changes.  
  
Debian GNU/Linux 9 ubuntu ttys0  
ubuntu login: █
```

成功！

进阶之OLED驱动

Zero为我们把这么多引脚引出，包含了很多的信号口。直接上并口的大屏幕难度太大，那就来做一个I2C接口的超小128*64的oled吧！

I2C是一种串行总线，详情可咨询搜索引擎。

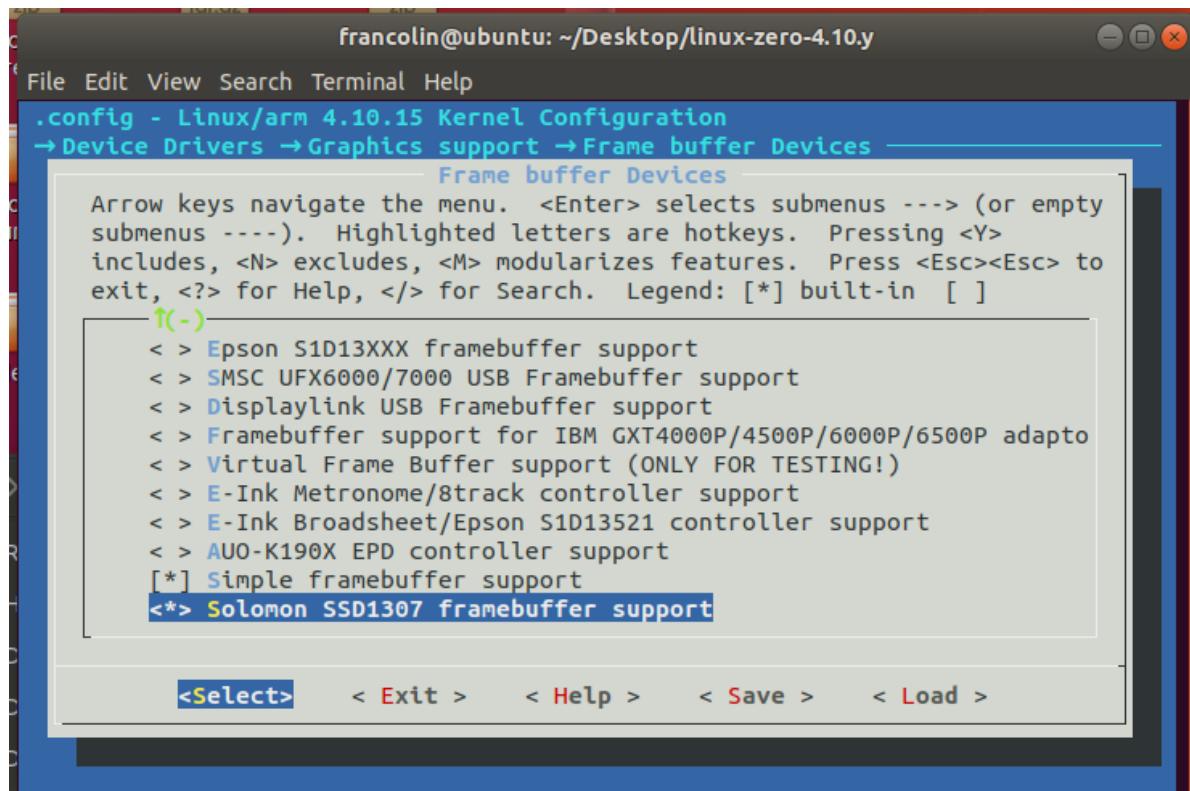
Linux驱动设计是一门很高深的学问，它既需要开发者有较好的硬件知识，还需要开发者非常理解linux操作系统。驱动的开发可以分为：设备注册，驱动注册，驱动实现。感兴趣的同学可以看《Linux设备驱动开发详解：基于最新的Linux 4.0内核》或者《Linux Device Drivers》，前者可以了解概念，后者被奉为经典。

Oled使用的控制芯片是ssd1306，我们下载的linux源码中包含Ssd1306的i2c驱动，驱动加载后会注册成功linux framebuffer，驱动文件路径是：/drivers/video/fbdev/ssd1307fb.c，这里我将驱动直接编译到内核里。

为了将驱动包含进来，不同于上面在内核配置时直接使用默认配置，我们需要配置包含SSD1306驱动在内核源码目录下：

```
make ARCH=arm menuconfig
```

修改Driver Devices->Graphics support->Frame buffer Devices,勾选Solomon SSD1307 framebuffer support(SSD1307兼容SSD1306)



保存后退出；

接下来我们要做的是修改设备树. (什么是设备树? <https://blog.csdn.net/u014650722/article/details/79076352>)

我们需要先进入linux内核目录，然后

```
vim arch/arm/boot/dts/sun8i-v3s-licheepi-zero.dts
```

在下图位置,添加下面这段代码：

```
ssd1306fb: ssd1306fb@3c {
    compatible = "solomon,ssd1306fb-i2c";
    reg = <0x3c>;
    solomon,width = <128>;
    solomon,height = <64>;
    reset-gpios = <&pio 1 0 GPIO_ACTIVE_HIGH>;
    solomon,page-offset = <0>;
    solomon,com-invdir;
};
```

```
francolin@ubuntu: ~/Desktop/linux-zero-4.10.y
File Edit View Search Terminal Help

&i2c0 {
    status = "okay";

    ns2009: ns2009@48 {
        compatible = "nsiway,ns2009";
        reg = <0x48>;
    };

    ssd1306fb: ssd1306fb@3c {
        compatible = "solomon,ssd1306fb-i2c";
        reg = <0x3c>;
        solomon,width = <128>;
        solomon,height = <64>;
        reset-gpios = <&pio 1 0 GPIO_ACTIVE_HIGH>;
        solomon,page-offset = <0>;
        solomon,com-invdir;■
    };
};

&uart0 {
    pinctrl-0 = <&uart0_pins_a>;
    pinctrl-names = "default";
-- INSERT --

```

其中width, height代表屏幕像素数, reg代表I2C地址（买到的oled基本都是这个地址）, compatible之所以这么写是因为SSD1307的驱动源码里有这样一段：

```
static const struct of_device_id ssd1307fb_of_match[] = {
{
    .compatible = "solomon,ssd1305fb-i2c",
    .data = (void *)&ssd1307fb_ssd1305DeviceInfo,
},
{
    .compatible = "solomon,ssd1306fb-i2c",
    .data = (void *)&ssd1307fb_ssd1306DeviceInfo,
},
{
    .compatible = "solomon,ssd1307fb-i2c",
    .data = (void *)&ssd1307fb_ssd1307DeviceInfo,
},
{
    .compatible = "solomon,ssd1309fb-i2c",
    .data = (void *)&ssd1307fb_ssd1309DeviceInfo,
},
},
```

这样，驱动就和这个设备对上号了。

接着，重新编译内核、设备树

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4  
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
```

插上U盘，将生成的zImage和.dtb文件重新拷贝至TF卡BOOT分区。**eject后拔出**。

接下来用杜邦线连接Zero和oled

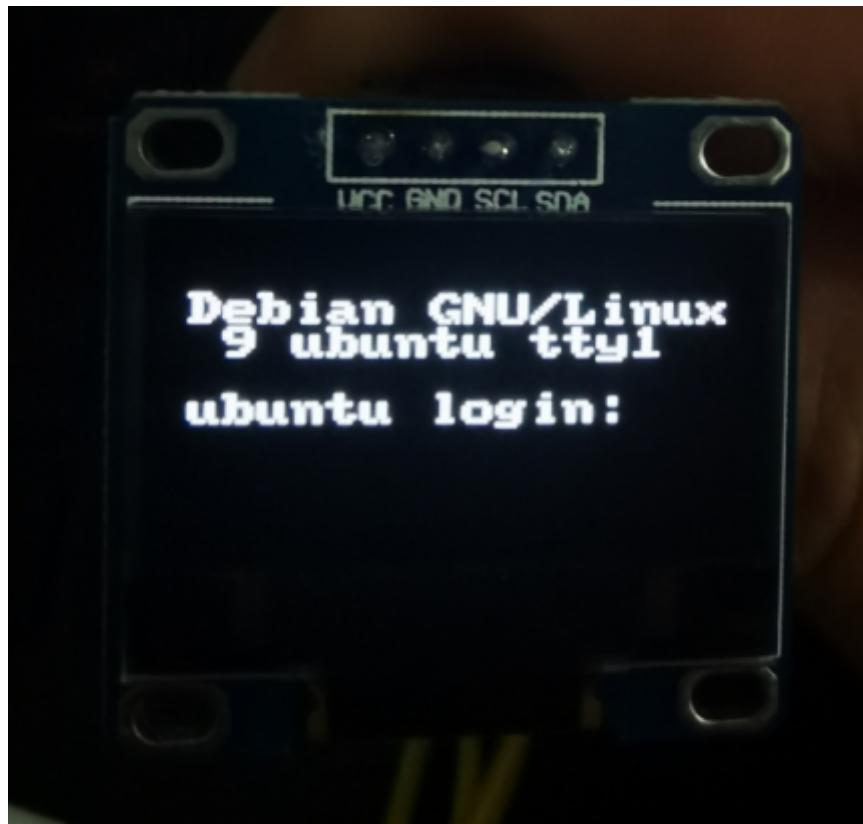
VCC->3V3-----(电源正)

GND->GND-----(GROUND)

SCL->SCK-----(CLOCK)

SDA->SDA-----(DATA)

将tf卡插上，重新上电：

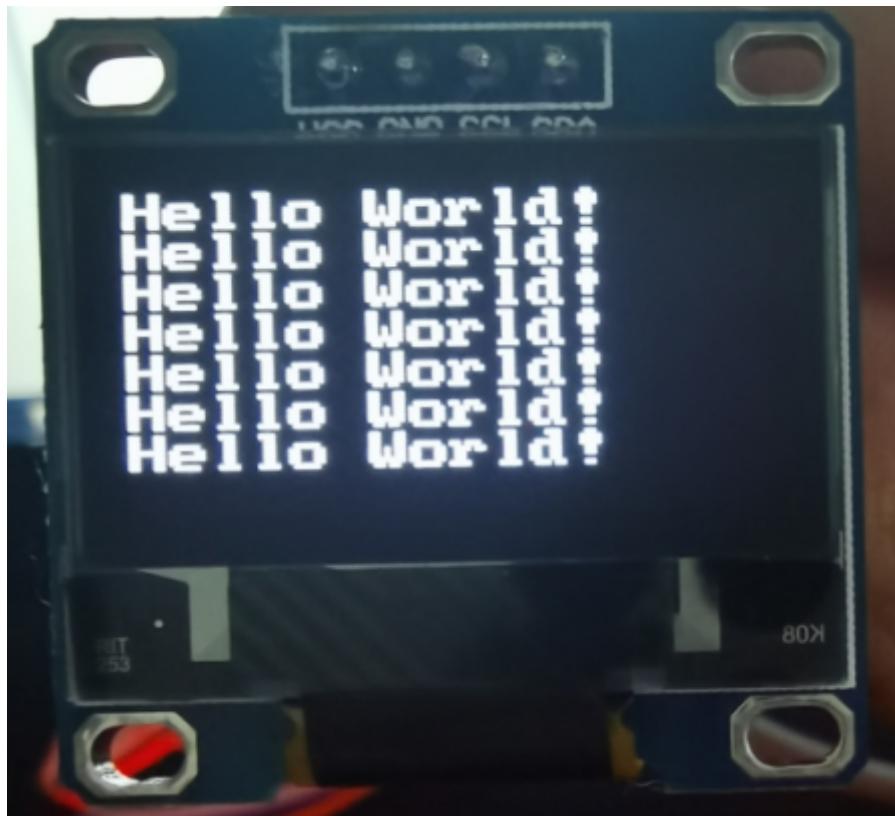


发现系统已经把这个显示屏当做tty1终端的输出了

我在Xshell里输入了很多

```
echo Hello World! > /dev/tty1
```

于是屏幕变成了：



屏幕驱动成功！

使用刚刚安装debian文件系统时顺便安装的vim, gcc编写并编译一个小c程序

```
#include <stdio.h>
int main(){
    printf("Hello Francolin!\r\n");
    return(0);
}
```

```
root@ubuntu:/home# gcc -c main.c
root@ubuntu:/home# gcc main.o -o out
root@ubuntu:/home# ./out > /dev/tty1
```

效果如下：



其他.....

以上是我给大家做的一个demo，其他的功能还要靠大家实现，主要目的是为了让大家更了解Linux操作系统，进行嵌入式设备Linux的简易开发。大家可以参考我的过程，也可以另辟蹊径，任何开发板都是可以的，任何功能都是可以的，如果有任何疑问可以qq上与我私聊，另外，**希望大家在过程中也能用文档记录自己的过程，这一方面是有利于自己，一方面也便于在结束的时候评价大家的成果。**

这块板子的实力远远没有被开发，就比如加上官方给的wifi驱动，板子可以联网，使用openwrt系统，你甚至可以做个路由器出来.....小巧的体型又让他可以有一些很特殊有趣的用处.....

可能有用的网址

<https://www.kancloud.cn/lichee/lpi0/317714>

<http://zero.lichee.pro/index.html>

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

<https://zhuanlan.zhihu.com/p/53822863>