# 软件开发指南
# T113X 核心板

成都亿佰特电子科技有限公司
Chengdu Ebyte Electronic Technology Co.,Ltd.

# 目录

# 免责申明和版权公告

本文中的信息，如有变更，恕不另行通知。 文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反 言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

文中所得测试数据均为亿佰特实验室测试所得，实际结果可能略有差异。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

最终解释权归成都亿佰特电子科技有限公司所有。

注 意 ：

由于产品版本升级或其他原因，本手册内容有可能变更。亿佰特电子科技有限公司保留在没有任何通知或者提示的情况下对本手册的内容进行修改的权利。本手册仅作为使用指导，成都亿佰特电子科技有限公司尽全力在本手册中提供准确的信息，但是成都亿佰特电子科技有限公司并不确保手册内容完全没有错误，本手册中的所有陈述、信息和建议也不构成任何明示或暗示的担保。

# 1. 概述

本文主要介绍基于亿佰特核心板定制一个完整的嵌入式 Linux 系统的完整流程，其中包括开发环境的准备，代码的获取，以及如何进行 Bootloader, Kernel 的移植，定制适合自身应用需求的 Rootfs 等。我们首先介绍如何基于我们提供的 SDK 源代码构建适用于 ECK30-T13IA 核心板的系统镜像，如何将构建好的镜像烧录到开发板。针对那些基于 ECK30-T13IA 核心板进行项目开发的用户，我们重点介绍了将这一套系统移植到用户的硬件平台上的方法和一些要点，并通过一些实际的移植案例和基于 Buildroot 的 Rootfs 定制案例，使用户能够迅速定制适合自己硬件的系统镜像。

请注意本文档并不包含 buildroot 项目以及 Linux 系统相关基础知识的介绍，适合有一定开发经验的嵌入式 Linux 系统开发人员和嵌入式 Linux BSP 开发人员。

# 2. 开发环境准备

本章主要介绍基于 ECK30-T13IA 核心板使用开发前需要的一些软硬件环境，包括必要的硬件配置，开发、烧录所需环境介绍。

通过阅读本章节，您将了解相关硬件工具，软件开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。

主机：i7-12700 + win11 22H2

虚拟机：Ubuntu 18.04

## 2.1.硬件准备

### 2.1.1. 产品型号

产品型号表

| 序号 | 产品规格型号 | 型号说明 |
|---|---|---|
| 1 | ECK30-T13IA2MN2M-I | 全志 T113-i 处理器+256MB 内存+256MB FLASH，国产工业级核心板。 |
| 2 | ECK30-T13IA5ME8G-I | 全志 T113-i 处理器+512MB 内存+8GB eMMC，国产工业级核心板。 |
| 3 | ECK30-T13IA1GE8G-I | 全志 T113-i 处理器+1GB 内存+8GB eMMC，国产工业级核心板。 |

### 2.1.2. 调试串口

调试串口：波特率 115200，数据位 8，停止位 1，无奇偶校验，无硬件流控，实际串口请自行配置，配置方式详见后文。

### 2.1.3. 启动方式

依据 T113-i 芯片的用户手册，T113-i 将根据 boot 引脚配置，依顺序启动，启动方式与

顺序详见下表。

启动顺序表

| BOOT_PIN[1:0] | 启动顺序 |
|---|---|
| 00 | SPI NOR —> SPI NAND |
| 01 | SMHC0 —> SPI NOR —> other media |
| 10 | SMHC0 —> SPI NAND —> other media |
| 11 | SMHC0 —> EMMC2_BOOT —> EMMC2_USR —> other media |

对于使用本司 ECK30-T13IA 核心板用户无需关心该部分，相关产品已经在板上分别实现了`10`和`11`两种对应配置。用户应该记住的是，一般来说，配置 SMHC0 为 SD 卡功能，而后一旦插入可启动的 SD 卡，系统将从 SD 卡启动，SD 卡无法启动，则从板上存储设备启动。

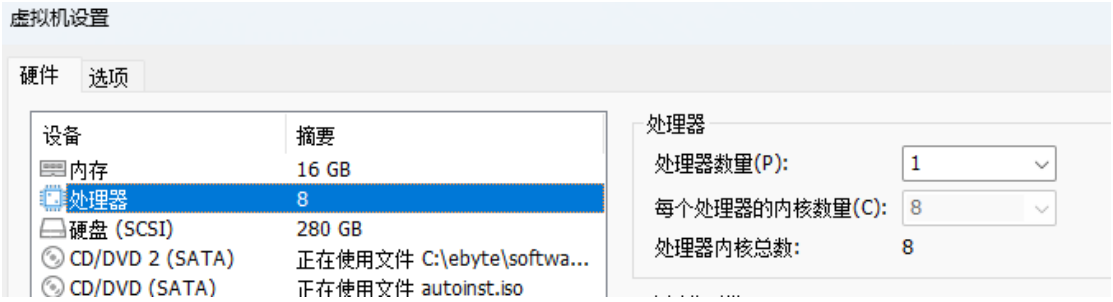
## 2.2.虚拟机软件部署

本章节将介绍如何搭建 T113-i 的开发环境。通过阅读本章节，您将了解相关开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。

### 2.2.1. 虚拟机配置

当前较为常见的虚拟机软件有 VirtualBox，VMware 等，不同平台的配置方式有所区别，用户使用习惯亦有所区别，故请用用户自行根据使用的虚拟机软件配置虚拟机使用桥接的方式联网，并打开与 windows 的目录共享功能。

虚拟机建议设置内存大小为 16G 以上，设置为 8 核，否则在编译时会导致无法通过。




### 2.2.2. 配置编译环境

由于 T113-i 的 SDK 中有会使用 buildroot 201902 版本，故推荐用户使用 Ubuntu 18.04 64bit 发行版，对于更高版本发行版，也可以尝试使用 buildroot 的 202205 版本，但依然请确保版本与 buildroot 版本兼容。

安装软件：

```
sudo apt update && sudo apt upgrade -y
```

```
sudo apt install -y git make gcc g++ flex bison xz-utils libncurses-dev device-tree-compiler netpbm bc libssl-dev lzop libncursesw5-dev bear gcc-arm-linux-gnueabihf flex bison xz-utils libssl-dev rsync fakeroot libtool texinfo automake unzip

sudo apt install -y libc6:i386 libgcc1:i386 libstdc++6:i386

sudo ln -sf /usr/bin/python3 /usr/bin/python
```

创建工作目录：

```
mkdir ~/work
```

### 2.2.3. 开启虚拟机的 TFTP 服务

后面调试阶段往往会使用 TFTP 加载内核启动，而不是每次去烧写存储设备，因此要先在 Ubuntu 中安装并开启 TFTP 服务，使用如下命令安装 TFTP 服务：

```
sudo apt install tftpd-hpa -y
```

默认情况下，将分享/var/lib/tftpboot 目录下的文件，且只能下载，不能上传，如果想个性化配置，请参考官方文档：https://help.ubuntu.com/community/TFTP，本文档已经修改路径为/srv/tftp。

添加用户到 tftp 组并修改共享目录权限：

```
sudo usermod -a -G tftp $(whoami)

sudo chmod 775 /srv/tftp/

sudo chgrp tftp /srv/tftp/
```

如果没有执行上述操作，可能出现权限问题，非 root 用户权限下无法写入文件到共享目录。

### 2.2.4. 开启虚拟机的 NFS 服务

后面调试阶段往往需要用 NFS 共享文件或者直接通过 NFS 挂载文件系统，因此要先在 Ubuntu 中安装并开启 NFS 服务，使用如下命令安装 NFS 服务：

```
sudo apt install nfs-kernel-server -y
```

等待安装完成，安装完成以后设置需要导出的目录，此处直接使用 tftp 默认目录，用户可自行修改，也可使用顺手的文本编辑器修改/ext/exports 文件，配置方法参考 man 手册：

```
echo "/srv/tftp *(rw,sync,no_subtree_check,no_root_squash,fsid=root)" | sudo tee -a /etc/
```

```
exports

    man exports
```

### 2.2.5. 开启虚拟机的 SSH 服务

开启 Ubuntu 的 SSH 服务以后我们就可以在 Windwos 下使用 ssh 客户端软件登陆到 Ubuntu，比如使用 MobaXterm、VSCode，Ubuntu 下使用如下命令开启 SSH 服务：

```
    sudo apt install openssh-server -y
```

上述命令安装 ssh 服务，ssh 的配置文件为/etc/ssh/sshd_config，使用默认配置即可，更多配置参考 sshd_config 相关的 man 手册。

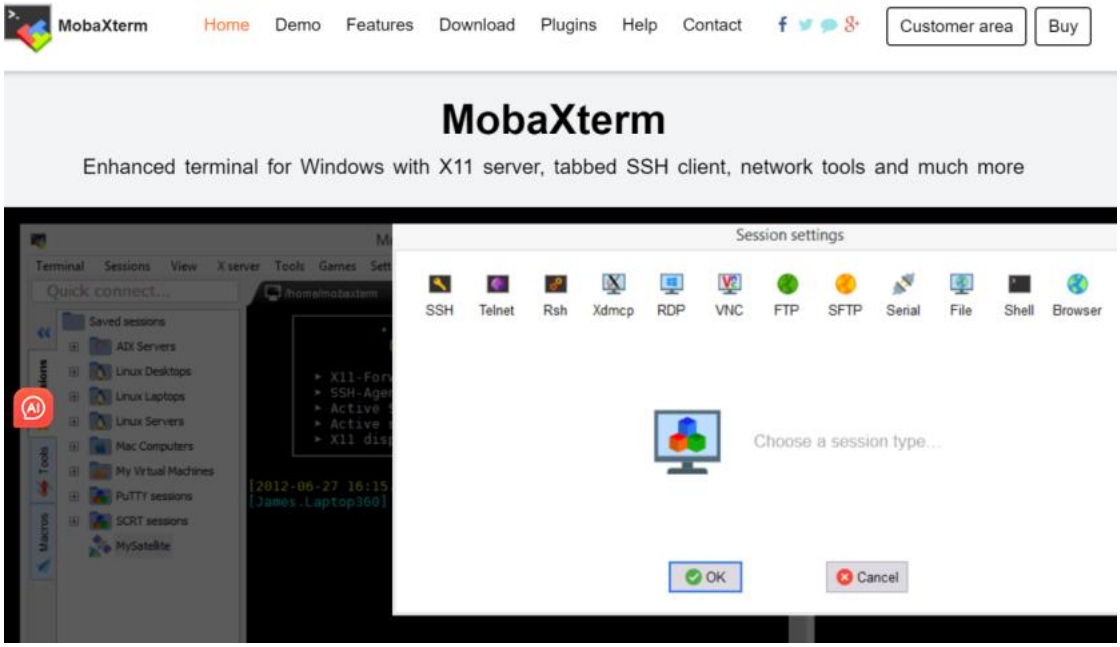
## 2.3.主机软件安装

本章节将介绍如何搭建 T113-i 的烧录调试环境。通过阅读本章节，您将了解相关烧写调试工具的安装和使用。

### 2.3.1. MobaXterm

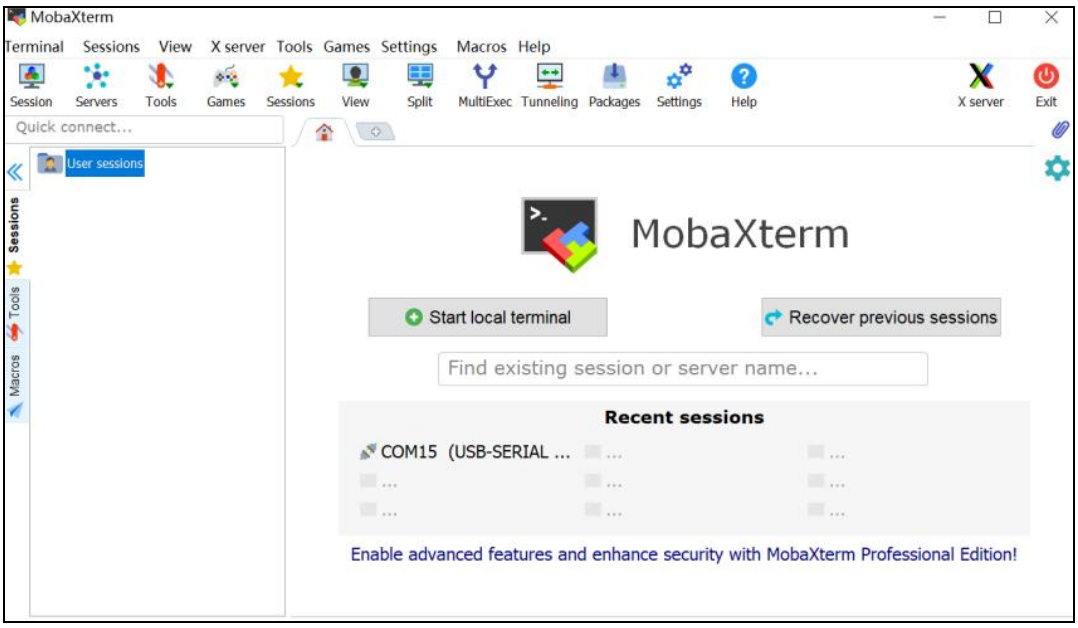
#### 2.3.1.1. 软件下载安装

MobaXterm 是一款终端软件，功能强大而且免费，推荐使用此软件作为终端调试软件，MobaXterm 软件在其官网下载即可，地址为 https://mobaxterm.mobatek.net/，如图所示：

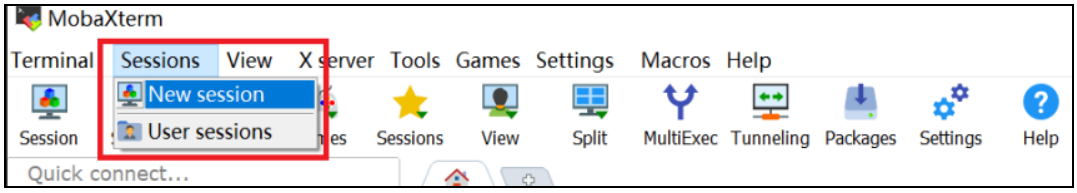



#### 2.3.1.2. 软件使用

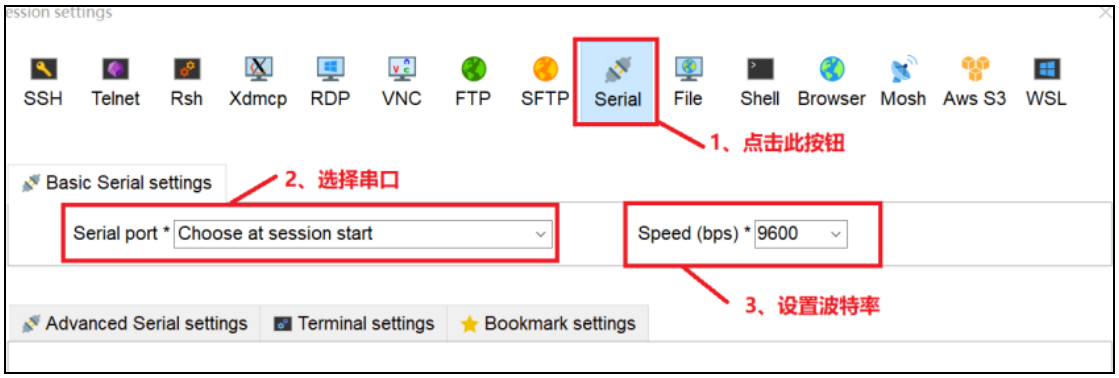
软件界面如图所示：




点击菜单栏中的“Sessions->New session”按钮，打开新建会话窗口，如图所示




建立 Serial 连接，也就是串口连接，因为我们使用 MobaXterm 的主要目的就是作为串口终端使用。点击图中的“Serial”按钮，打开串口设置界面，如图所示：




打开串口设置窗口以后先选择要设置的串口号，因此要先用串口线将开发板连接到电脑上，由于 T113-i 默认使用的串口波特率为 115200，需要设置波特率为 115200。

## 2.3.2. PhoenixSuit

该工具为全志官方提供的从 USB 烧录系统的工具，该软件已存放在目录：“04_Tools/PhoenixSuit”。具体使用方式详见后文《官方工具烧录》一节。

### 2.3.3. PhoenixCard

这是全志官方提供的用于制作系统卡的工具，可以选择烧录制作量产卡与启动卡，该软件已存放在目录：“04_Tools/PhoenixCard”。具体使用方式详见后文《制作 SD 卡启动卡》一节。

## 2.4.数据传输

在前面章节我们已经配置好了虚拟机部分各种网络服务，现在只需要了解如何在客户端与开发板的使用即可。

### 2.4.1. TFTP 客户端

一般情况下，开发板只需要使用下载功能即可，此时可以简单的使用 curl 工具下载，使用之前需要先在 buildroot 中选配 curl 功能，然后使用 curl 下载文件(仅限 Linux 版 curl，windows 可选择使用 filezilla)：

```
curl -o /path/dist_file tftp://HOST/src_file
```

1. /path/dist_file：要下载到哪里，输出文件名字的名字与路径。

2. HOST：虚拟机的 ip 地址，需要注意这个地址是否可以访问。

3. src_file：文件目录以及名字，需要注意的是不需要添加配置中的"TFTP_DIRECTORY"部分，比如文件存放在`/srv/tftp/aaa/bbb/ccc`，src_file 值为"aaa/bbb/ccc"。

### 2.4.2. NFS 客户端

一般情况下，开发板只需要简单的使用 mount 命令即可挂载 NFS 文件系统，但是需要安装基本的支持，然后挂载 NFS：

```
mount -t nfs HOST:/path /mnt_point
```

1. HOST：虚拟机的 ip 地址，需要注意这个地址是否可以访问。

2. path：想要挂载的 NFS 服务器中的路径，比如/srv/tftp/aaa/bbb。

3. mnt_point：想要把 NFS 文件系统挂载到哪里，比如/mnt。

### 2.4.3. SSH

一般来说我们也可以使用 SSH 自带的 SFTP 与 SCP 功能进行文件下载，但直接使用命令上传下载的情况不多，更多使用第三方工具进行，如使用 MobaXterm 登录 ssh 后的 SFTP 功能，或者 VSCode 使用 remote-SSH 插件实现开发过程中的下载功能，使用方式都是鼠标放在需要下载的文件上右键，然后选择下载。

### 2.4.4. U 盘/SD 卡

对于 SD 卡，需要使用读卡器，实际用法和 U 盘一致。

#### 2.4.4.1. 从虚拟机拷贝数据

用户需要根据自身使用的虚拟机软件先将 U 盘/SD 卡传递到虚拟机当中，一般来说会自动挂载，在文件管理器中可以直接访问，拷贝数据完毕后在文件管理器中卸载设备，然后从虚拟机中移除设备即可。

如果没有自动挂载，可以使用下方的两种指令手动挂载：

1. 方式一

```
udisksctl mount --block-device /dev/sdXY
```

2. 方式二

```
sudo mount /dev/sdXY /mnt_point
```

这种情况需要对 U 盘/SD 卡分区有一点了解，才可以指定使用 sdXY 中的 Y 值，X 值一般是'b'，但是也需要自行通过 lsblk 的返回信息判断情况。

使用方式一挂载，无需 root 权限即可访问，也不用指定路径，但是需要系统中有安装了`udisks2`这个程序包。

手动卸载对应如下命令：

1. 方式一

```
udisksctl unmount --block-device /dev/sdXY

udisksctl power-off --block-device /dev/sdXY
```

2. 方式二

```
sudo umount /mnt_point
```

#### 2.4.4.2. 拷贝数据到开发板

对于开发板，一般会进行精简，不一定有 udisks2 包，也不一定有一个 GUI 界面，有界面也不一定支持自动挂载，但是往往用户都有 root 权限，所以一般从命令行执行 mount 操作。

1. 挂载

```
mount /dev/sdXY /mnt_point
```

2. 拷贝

```
cp  /mnt_point/file  /out_path/out_file

cp  -ra  /mnt_point/dir  /out_path/out_dir
```

3. 取消挂载

```
umount  /mnt_point
```

# 3. SDK 编译构建

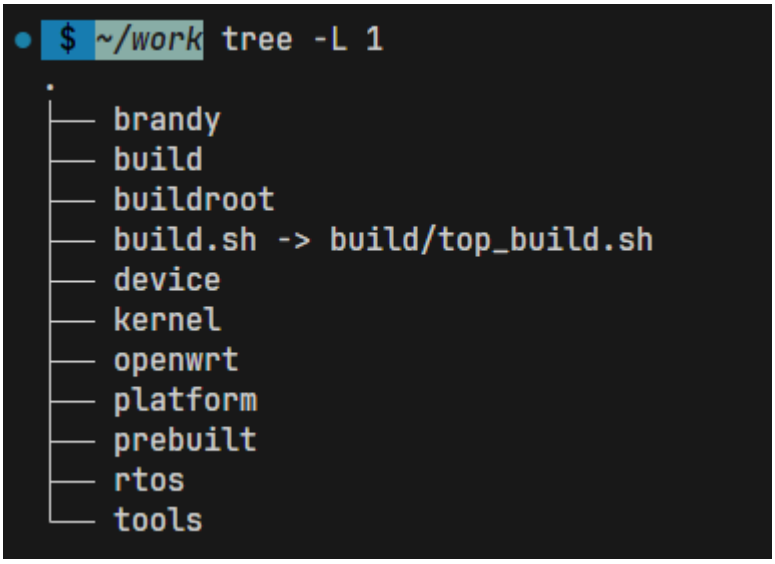
本章节将描述从 SDK 的获取到编译生成一个可用镜像的完整流程。

## 3.1.SDK 获取

T113-I 的开发 SDK 已经随附在下载文件中，请用户自行解压至开发目录中。

## 3.2.SDK 目录

解压完毕后，整个 SDK 目录情况如下图：




我们需要关注下面这些目录：

| 顶级目录 | 子目录 | 作用 |
|---|---|---|
| brandy | brandy-2.0/spl-pub/ | SPL 源码，一般不修改，全志官方提供的也大部分是二进制文件 |
| | brandy-2.0/u-boot-2018/ | U-boot 源码 |
| build | | 编译脚本目录，一般不修改 |
| buildroot | buildroot-201902 | 文件系统源码 |
| | buildroot-202205 | |
| | package/auto/sdk_demo/ | 示例程序 |
| device | config/chips/t113_i | SDK 支持的芯片配置目录，包括多款开发板的配置方案 |
| | product | 默认没有，在执行 SDK 配置后指向目标芯片，此处为 T113_i |

| | | |
|---|---|---|
| kernel | linux-5.4 | 内核源码 |
| openwrt | target/t113_i/ | openwrt 方案配置 |
| | openwrt/ | openwrt 源码 |
| platform | | 平台扩展相关源码，如 lvgl，qt 等 |
| prebuilt | | 内核、buildroot 和 openwrt 编译工具 |
| rtos | | rtos 源码，主要与异构核心和 DSP 相关 |
| tools | tools_win | 一些 windows 下的工具 |

## 3.3.SDK 配置

由于 SDK 支持多款芯片，也支持多种方案，所以在正式开始编译之前需要用户选择符合自身开发板的配置，SDK 中也提供了配置方式：

```
./build.sh  config
```

执行完毕后需要同意全志的一些许可认证：

```
$ ~/work ./build.sh config

Before using this files, please make sure that you note the following important information.
*************************************************************************

Copyright (c) 2019-2022 Allwinner Technology Co., Ltd. ALL rights reserved.

Allwinner is a trademark of Allwinner Technology Co.,Ltd., registered in
the the People's Republic of China and other countries.
All Allwinner Technology Co.,Ltd. trademarks are used with permission.

DISCLAIMER
THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT.
IF YOU NEED TO INTEGRATE THIRD PARTY'S TECHNOLOGY (SONY, DTS, DOLBY, AVS OR MPEGLA, ETC.)
IN ALLWINNERS'SDK OR PRODUCTS, YOU SHALL BE SOLELY RESPONSIBLE TO OBTAIN
ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES.
ALLWINNER SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS
COVERED UNDER ANY REQUIRED THIRD PARTY LICENSE.
YOU ARE SOLELY RESPONSIBLE FOR YOUR USAGE OF THIRD PARTY'S TECHNOLOGY.


THIS SOFTWARE IS PROVIDED BY ALLWINNER"AS IS" AND TO THE MAXIMUM EXTENT
PERMITTED BY LAW, ALLWINNER EXPRESSLY DISCLAIMS ALL WARRANTIES OF ANY KIND,
WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION REGARDING
THE TITLE, NON-INFRINGEMENT, ACCURACY, CONDITION, COMPLETENESS, PERFORMANCE
OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL ALLWINNER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS, OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.

*************************************************************************
You can read /home/me/work/build/disclaimer/Allwinnertech_Disclaimer(Cn_En)_20181122.md for detailed information.

You read time left 8 seconds....
I have already read, understood and accepted the above terms? [Y/N]
```

在此处敲击`Y`并回车确认，进入后续选择：这里以亿佰特的 ECB30 的 T113-i 为例，按照图 1 进行配置，如果是亿佰特的 ECB31 的 T113-S3，则按照图 2 进行配置，如果是亿佰特的 ECB31 的 T113-S4，则按照图 3 进行配置。

```
hu@ubuntu:~/linux/allwinner$ ./build.sh config
========ACTION List: mk_config ;========
options :
All available platform:
   0. android
   1. linux
Choice [linux]:
All available linux_dev:
   0. bsp
   1. buildroot
   2. openwrt
   3. ubuntu
Choice [buildroot]:
All available ic:
   0. t113
   1. t113_i
   2. t113_s3p
   3. t113_s4
   4. t113_s4p
   5. t113s2
Choice [t113_i]:
All available board:
   0. ebyte_t113i
   1. ebyte_t113i_BTB
   2. ebyte_t113i_nand
   3. evb1
   4. evb1_auto
   5. evb1_auto_buildroot2022_RT
   6. evb1_auto_buildroot2022_test
   7. evb1_auto_nand
   8. evb1_auto_nor
Choice [ebyte_t113i]:
All available flash:
   0. default
   1. nor
Choice [default]:
```

```
hu@ubuntu:~/linux/allwinner$ ./build.sh config
========ACTION List: mk_config ;========
options :
All available platform:
   0. android
   1. linux
Choice [linux]:
All available linux_dev:
   0. bsp
   1. buildroot
   2. openwrt
   3. ubuntu
Choice [buildroot]:
All available ic:
   0. t113
   1. t113_i
   2. t113_s3p
   3. t113_s4
   4. t113_s4p
   5. t113s2
Choice [t113]:
All available board:
   0. dev
   1. ebyte_t113
   2. ebyte_t113_nand
   3. evb1
   4. evb1_auto
   5. evb1_auto_nand
   6. evb1_auto_nand_fastboot
   7. evb1_auto_nor
   8. pro
Choice [ebyte_t113]:
All available flash:
   0. default
```

图一（左），图二（右），图三（下）

```
hu@ubuntu:~/linux/allwinner$ ./build.sh config
========ACTION List: mk_config ;========
options :
All available platform:
   0. android
   1. linux
Choice [linux]:
All available linux_dev:
   0. bsp
   1. buildroot
   2. openwrt
   3. ubuntu
Choice [buildroot]:
All available ic:
   0. t113
   1. t113_i
   2. t113_s3p
   3. t113_s4
   4. t113_s4p
   5. t113s2
Choice [t113_s4]:
All available board:
   0. ebyte_t113s4
   1. ebyte_t113s4_nand
   2. evb1_auto
   3. evb1_auto_fastboot_video
   4. evb1_auto_nand
   5. evb1_auto_nand_fastboot_video
Choice [ebyte_t113s4]:
All available flash:
   0. default
   1. nor
Choice [default]:
All available kern_name:
   0. linux-5.4
Choice [linux-5.4]:
```

此处使用的编译配置为`SDK/device/config/chips/t113_i/configs/ebyte_t113i/buildroot/BoardConfig.mk`。配置完毕后，最终的编译规则配置可以在 SDK 目录下的`.buildconfig`文件查看，下面列举其中比较重要的部分：

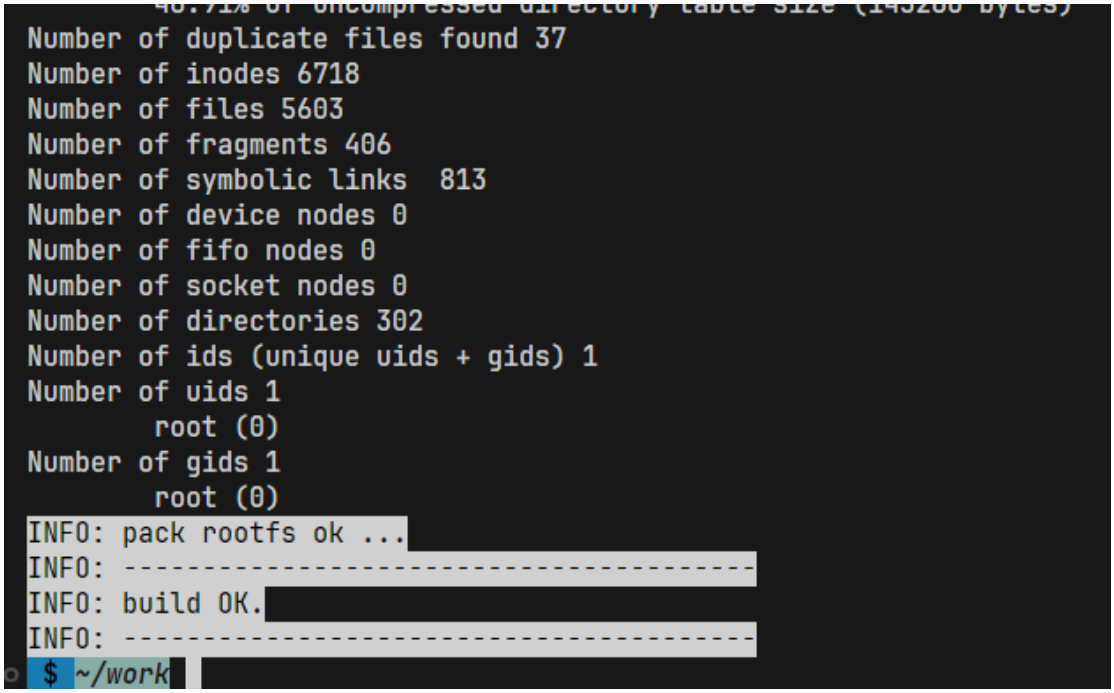
```
LICHEE_PLATFORM  //编译平台
LICHEE_LINUX_DEV  //编译的方案
LICHEE_IC  //  编译的 IC
LICHEE_BOARD  //编译的板级配置
LICHEE_FLASH  //编译的 flash 配置
LICHEE_CHIP  //编译的芯片代号名称
LICHEE_KERN_VER  //编译的内核版本
LICHEE_KERN_DEFCONF  //编译内核的默认配置
LICHEE_BUILDING_SYSTEM  //编译的构造系统
LICHEE_BR_VER  //buildroot 的版本
LICHEE_BR_DEFCONF  //buildroot 的默认配置
LICHEE_BR_RAMFS_CONF  //buildroot 的 ramfs 默认配置
LICHEE_BRANDY_VER  //uboot 的版本
LICHEE_BRANDY_DEFCONF  //brandy 的默认配置
LICHEE_CROSS_COMPILER  //编译使用的交叉编译链
LICHEE_CHIP_CONFIG_DIR  //编译系统的 IC 配置目录
LICHEE_OUT_DIR  //编译的输出目录
```

## 3.4.编译 SDK

等待 SDK 结束配置，然后开始执行编译：

```
./build.sh
```

预计耗时约半小时编译完成后，生成 SPL、U-Boot、Linux 内核和 Buildroot 文件系统镜像文件。编译结束后会有如下输出：




## 3.5.单独编译

### 3.5.1. 单独编译 u-boot

通过如下命令可以单独编译 u-boot，对于只需要修改 u-boot 阶段的时候可以节省编译时间：

```
./build.sh  bootloader
```

```
$ ~/work ./build.sh bootloader
========ACTION List: build_bootloader ;========
options :
find: '/home/me/work/brandy/brandy-2.0/spl': No such file or directory
find: '/home/me/work/brandy/brandy-2.0/u-boot-bsp': No such file or directory
find: '/home/me/work/brandy/dramlib': No such file or directory
INFO: build_bootloader: brandy_path=/home/me/work/brandy/brandy-2.0
INFO: uboot-board.dts updated.
build_option:-p sun8iw20p1_auto_t113_i -b t113_i
grep: /home/me/work/brandy/brandy-2.0/spl/Makefile: No such file or directory
_MAKE PATH: /home/me/work/brandy/brandy-2.0/tools/make_dir/make4.1/bin/make
Prepare toolchain ...
uboot version:u-boot-2018
build for sun8iw20p1_auto_t113_i_defconfig ...
cat: .tmp_config_from_defconfig.o.md5sum: No such file or directory
md5sum: .config: No such file or directory
  CLEAN   scripts/basic
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
board/sunxi/Kconfig:326:warning: defaults for choice values not supported
#
# configuration written to .config
#
```

编译结束输出如下 log：

```
bootaddr is 0x20480
/home/me/work/brandy/brandy-2.0/spl-pub/../tools/toolchain/gcc-linaro-7.2.1-2017.11-x86_64_arm
ot.a /home/me/work/brandy/brandy-2.0/spl-pub/sboot/main/sboot_main.o  -L /home/me/work/brandy/
7.2.1 -lgcc  --gc-sections --gc-sections -Tsboot.lds -o sboot.elf -Map sboot.map
/home/me/work/brandy/brandy-2.0/spl-pub/../tools/toolchain/gcc-linaro-7.2.1-2017.11-x86_64_arm
'sboot_sun8iw20p1.bin' -> '/home/me/work/brandy/brandy-2.0/spl-pub/out/sun8iw20p1/bin/sboot_su
'sboot_sun8iw20p1.bin' -> '/home/me/work/device/config/chips/t113_i/bin/sboot_sun8iw20p1.bin'
'sboot_sun8iw20p1.bin' -> '/home/me/work/out/t113_i/evb1_auto/buildroot/sboot_sun8iw20p1.bin'
INFO: build brandy OK.
```

### 3.5.2. 单独编译 kernel

通过如下命令可以单独编译内核，对于只需要修改内核阶段的时候可以节省编译时间：

```
./build.sh  kernel
```

```
$ ~/work ./build.sh kernel
========ACTION List: build_kernel ;========
options :
INFO: build kernel ...
INFO: prepare_buildserver
INFO: Prepare toolchain ...

update .config with /home/me/work/device/config/chips/t113_i/configs/evb1_auto/linux-5.4/config-5.4 ...

make[1]: Entering directory '/home/me/work/out/t113_i/kernel/build'
  GEN     Makefile
*** Default configuration is based on '../../../../../device/config/chips/t113_i/configs/evb1_auto/linux-5.4/config-5.4'
#
# No change to .config
#
make[1]: Leaving directory '/home/me/work/out/t113_i/kernel/build'
Building kernel
make[1]: Entering directory '/home/me/work/out/t113_i/kernel/build'
  GEN     Makefile
  DTC     arch/arm/boot/dts/board.dtb
  CALL    /home/me/work/kernel/linux-5.4/scripts/atomic/check-atomics.sh
  CALL    /home/me/work/kernel/linux-5.4/scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  Kernel: arch/arm/boot/Image is ready
  Building modules, stage 2.
  MODPOST 29 modules
```

编译完毕输出如下 log：

```
sun8iw20p1 compile all(Kernel+modules+boot.img) successful


INFO: build dts ...
INFO: Prepare toolchain ...
removed '/home/me/work/out/t113_i/evb1_auto/buildroot/.board.dtb.d.dtc.tmp'
removed '/home/me/work/out/t113_i/evb1_auto/buildroot/.board.dtb.dts.tmp'
'/home/me/work/out/t113_i/kernel/build/arch/arm/boot/dts/.board.dtb.d.dtc.tmp' -> '/home/me/work/out/t113_i/evb1
'/home/me/work/out/t113_i/kernel/build/arch/arm/boot/dts/.board.dtb.dts.tmp' -> '/home/me/work/out/t113_i/evb1_a
'/home/me/work/out/t113_i/kernel/staging/sunxi.dtb' -> '/home/me/work/out/t113_i/evb1_auto/buildroot/sunxi.dtb'
```

## 3.5.3. 单独编译 buildroot

通过如下命令可以单独编译 buildroot，对于只需要修改 buildroot 阶段的时候可以节省编译时间：

```
./build.sh buildroot_rootfs
```

```
$ ~/work ./build.sh buildroot_rootfs
========ACTION List: build_buildroot_rootfs ;========
options :
INFO: build buildroot ...
make: Entering directory '/home/me/work/buildroot/buildroot-201902'
>>>   Finalizing target directory
# Check files that are touched by more than one package
./support/scripts/check-uniq-files -t target /home/me/work/out/t113_i/evb1_auto/buildroot/buildroot/build/packages-file-list.txt
Warning: target file "./etc/fw_env.config" is touched by more than one package: ['busybox-init-base-files', 'uboot-tools']
Warning: target file "./usr/lib/libsubdecoder.so" is touched by more than one package: ['libcedarc', 'libcedarx']
Warning: target file "./usr/share/alsa/cards/aliases.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/AACI.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/ATIIXP.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/ATIIXP-SPDMA.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/ATIIXP-MODEM.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/AU8810.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/AU8820.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/AU8830.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/Audigy.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
Warning: target file "./usr/share/alsa/cards/Audigy2.conf" is touched by more than one package: ['alsa-lib', 'alsa-utils']
```

编译完毕输出如下 log：

```
# @echo -e '\e[1;33m cp -f mem_test /home/me/work/buildroot/package/auto/sdk_demo/bin \e[0m'
make finish!!!
make[1]: Leaving directory '/home/me/work/buildroot/package/auto/sdk_demo/mem_test'
make: Leaving directory '/home/me/work/buildroot/package/auto/sdk_demo'
build auto finish
INFO: copy the config files form device ...
make: Entering directory '/home/me/work/platform'
Makefile:35: "----------1--------"
Makefile:36: /home/me/work/platform
make: Nothing to be done for 'INSTALL_FILES'.
make: Leaving directory '/home/me/work/platform'
INFO: build buildroot OK.
```

## 3.6.编译输出

在编译完毕以后会发现在 SDK 目录中出现了一个 out 目录，这里面存放了 SDK 的全部编译输出。

```
$ ~/work ls
brandy  build  buildroot  build.sh  device  kernel  openwrt  out  platform  prebuilt  rtos  tools
$ ~/work ls out/
kernel  serversocket  t113_i  toolchain
$ ~/work
```

其中 kernel 目录为 t113_i/kernel 目录的软连接，是内核的完整编译输出，toolchain 目录为编译过程中使用到交叉编译工具链，我们主要来看 t113_i 目录：

```
$ ~/work/out/t113_i tree -L 2
.
├── evb1_auto
│   └── buildroot
└── kernel
    ├── build
    └── staging
```

进入到 t113_i/evb1_auto/buildroot 目录：

```
$ ~/work/out/t113_i/evb1_auto/buildroot ls
arisc                                  boot0_nand_sun8iw20p1.bin    dtc                  rootfs.ubifs          vmlinux
bImage                                 boot0_sdcard_sun8iw20p1.bin  fes1_sun8iw20p1.bin  sboot_sun8iw20p1.bin  vmlinux.tar.bz2
boot0_mmc_car_fastboot_sun8iw20p1.bin  boot0_spinor_sun8iw20p1.bin  lib                  sunxi.dtb             zImage
boot0_mmcfastboot_sun8iw20p1.bin       boot.img                     ramfs.cpio.gz        System.map
boot0_nand_car_fastboot_sun8iw20p1.bin buildroot                    rootfs.ext4          u-boot-sun8iw20p1.bin
boot0_nandfastboot_sun8iw20p1.bin      dist                         rootfs.squashfs      uImage
$ ~/work/out/t113_i/evb1_auto/buildroot
```

我们主要需要关注其中下面这几个文件：

❖boot0_*_sun8iw20p1.bin：SPL 镜像，用于引导 u-boot

❖u-boot-sun8iw20p1.bin：u-boot 镜像，用于引导 kernel

❖boot.img：内核镜像，包含 kernel，设备树，安卓镜像格式

❖sunxi.dtb：编译出来的内核设备树文件，最终会打包到 boot.img 中。

❖zImage：编译出来的内核文件，最终会打包到 boot.img 中。

❖rootfs.*：buildroot 生成的根文件系统。通常后缀为 ext4 的用于 eMMC/SD 卡，后缀

为 ubifs 的用于 NAND。

## 3.7.镜像打包

经过之前的编译后，输出了各种编译文件，但是大多无法直接使用，需要打包成最终可用的镜像：

```
./build.sh  pack
```

等待打包结束后查看 out 目录，会发现多出来了一个 pack_out 目录与一个 img 文件，该 img 文件就是最终需要烧录到开发板上的镜像文件，而 pack_out 目录则是打包该镜像的中间文件，其中部分文件也是最终可以用到的。

再来看 pack_out 目录：

```
$ ~/work/out ls -l
total 266652
lrwxrwxrwx 1 me me        13 Sep 25 17:47 kernel -> t113_i/kernel
lrwxrwxrwx 1 me me        25 Sep 25 17:48 pack_out -> t113_i/evb1_auto/pack_out
srwxrw-rw- 1 me me         0 Sep 25 16:41 serversocket
drwxrwxr-x 4 me me      4096 Sep 25 16:41 t113_i
-rwxrwxr-x 2 me me 273043456 Sep 25 17:48 t113_i_linux_evb1_auto_uart0.img
drwxrwxr-x 3 me me      4096 Sep 25 16:40 toolchain
```

可以发现他是一个从 t113_i/<board>/pack_out 过来的软连接：

```
$ ~/work/out/t113_i/evb1_auto/pack_out ls
amp_rv0.fex                       boot-resource          kernel.fex              sys_partition_ab.fex
arisc.fex                         boot-resource.fex      new_fdt.dtb             sys_partition.bin
aultls32.fex                      boot-resource.ini      optee.fex               sys_partition_dump.fex
aultools.fex                      cardscript.fex         parameter.fex           sys_partition.fex
battery_charge.bmp                cardscript_secure.fex  postinstall_A.sh        sys_partition_nor.fex
battery_charge.bmp.lzma           cardtool.fex           postinstall_B.sh        sys_partition_private.fex
bempty.bmp                        cnf_base.cnf           preinstall_A.sh         sys_partition-recovery.fex
bempty.bmp.lzma                   config.fex             preinstall_B.sh         sysrecovery.fex
board_config.bin                  default.awlic          rdiff                   temp_ubootnodtb.bin
BoardConfig-conditions.mk         diskfs.fex             rootfs.fex              toc0.fex
board_config.fex                  dlinfo.fex             rootfs_nor.fex          toc0_ft.fex
BoardConfig.mk                    dragon_toc.cfg         rootfs-ubifs.fex        toc0_nand.fex
BoardConfig_nor.mk                env_ab.cfg             sboot.bin               toc0_sdcard.fex
board.fex                         env_burn.cfg           split_xxxx.fex          toc0_ufs.fex
boot0_nandfastboot_sun8iw20p1.fex env.cfg                sunxi.fex               toc1.fex
boot0_nand.fex                    env_dragon.cfg         sunxi_gpt.fex           u-boot-crash.fex
boot0_sdcard.fex                  env.fex                sunxi_mbr.fex           u-boot.fex
boot0_spinor.fex                  env_nor.cfg            sunxi_version.fex       usbtool_crash.fex
boot.fex                          env-recovery.cfg       sw-description-ab       usbtool.fex
bootlogo.bmp                      esm.fex                sw-description-ab-rdiff usbtool_test.fex
bootlogo.bmp.lzma                 fes1.fex               sw-description-recovery Vboot.fex
bootlogo.bmp.lzma.head            fit-image.its          sw-subimgs-ab.cfg       Vboot-resource.fex
bootlogo.fex                      image.cfg              sw-subimgs-ab-rdiff.cfg Venv.fex
boot_package.cfg                  image_crashdump.cfg    sw-subimgs-recovery.cfg version_base.mk
boot_package.fex                  image_linux.cfg        sys_config.bin          vmlinux.fex
boot_package_nor.cfg              image_nor.cfg          sys_config.fex          Vrootfs.fex
$ ~/work/out/t113_i/evb1_auto/pack_out
```

其中内容比较多，我们主要关注下面这几个文件：

❖boot_package.fex：打包后的 u-boot 镜像，包含内容可以查看 boot_package.cfg

❖boot-resource.fex：启动过程中用到的资源，如 logo 图片。

❖env.fex：u-boot 的环境变量配置，SDK/device/product/configs/<board>/[buildroot]/env.

cfg 的复制。

❖boot.fex：上文编译出来的内核镜像 boot.img 的软连接。

❖rootfs*.fex：上文编译出来的 buildroot 根文件系统的软连接。

## 3.7.1. 可能遇到的问题

在打包过程中可能会遇到下面几种情况：

### 3.7.1.1. unable to open file amp_dsp0.fex

```
partitation file Path=/home/me/work/out/t113_i/evb1_auto/pack_out/sys_partition.bin
mbr_name file Path=/home/me/work/out/t113_i/evb1_auto/pack_out/sunxi_mbr.fex
download_name file Path=/home/me/work/out/t113_i/evb1_auto/pack_out/dlinfo.fex

mbr size = 16384
mbr magic softw411
disk name=boot-resource
disk name=env
disk name=env-redund
disk name=boot
disk name=rootfs
disk name=dsp0
ERROR:  unable to open file amp_dsp0.fex
update_for_part_info -1
ERROR: update mbr file fail
ERROR: update_mbr failed
```

若用户不需要 dsp，则可以在`SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex 中仿照如下方式注释即可。

```
[partition]
    name         = dsp0
    size         = 2048
;    downloadfile = "amp_dsp0.fex"
    user_type    = 0x8000
```

### 3.7.1.2. dl file XXXX size too large

```
mbr size = 16384
mbr magic softw411
disk name=boot-resource
disk name=env
disk name=env-redund
disk name=boot
disk name=rootfs
ERROR: dl file rootfs.fex size too large
ERROR: filename = rootfs.fex
ERROR: dl_file_size = 2097152 sector
ERROR: part_size = 97152 sector
update_for_part_info -1
ERROR: update mbr file fail
ERROR: update_mbr failed
```

若出现此种情况，表示用户配置的分区大小不适配最终输出文件，可以在`SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex 中查找图中 filename 对应文件对应分区，然后修改该分区大小以适配相应文件，分区大小最小值如上图中 dl_file_size 描述：

```
[partition]
    name         = rootfs
    size         = 97152
    ; size       = 2097152
    downloadfile = "rootfs.fex"
    user_type    = 0x8000
```

对于 nand + ubi 方案的用户，还需要注意，此处 size 应该为 504 的整数倍，ubi 分区实际有效大小为 size*252K。

# 4. 开发板适配

为了适配用户新的硬件平台，用户需要对 CPU 芯片手册，CPU 芯片用户指南有一定了解，同时需要对 SDK 修改内容用一定了解，从上面 SDK 个目录的描述中可以知道，在 SDK/device/product/configs/中存放的是开发板/方案配置：

```
$ ~/work/device/product/configs ls
default  evb1  evb1_auto  evb1_auto_buildroot2022_RT  evb1_auto_buildroot2022_test  evb1_auto_nand  evb1_auto_nor
$ ~/work/device/product/configs tree -L 1
.
├── default
├── evb1
├── evb1_auto
├── evb1_auto_buildroot2022_RT
├── evb1_auto_buildroot2022_test
├── evb1_auto_nand
└── evb1_auto_nor

7 directories, 0 files
$ ~/work/device/product/configs
```

这一级目录中，除了 default 目录以外，可以认为都是不同的开发板/方案配置，而 default 目录中的内容是作为对应板级配置中找不到需要项的时候的默认配置项，一般不用修改，也不应该修改。

依然以 evb1_auto 为例：

```
$ ~/work ls device/product/configs/evb1_auto
bin  BoardConfig.mk  board.dts  bsp  buildroot  env.cfg  linux-5.4  openwrt  sys_config.fex  sys_partition.fex  uboot-board.dts
$ ~/work tree -L 1 device/product/configs/evb1_auto
device/product/configs/evb1_auto
├── bin
├── BoardConfig.mk
├── board.dts -> linux-5.4/board.dts
├── bsp
├── buildroot
├── env.cfg
├── linux-5.4
├── openwrt
├── sys_config.fex
├── sys_partition.fex
└── uboot-board.dts

5 directories, 6 files
$ ~/work
```

在这里面我们主要需要注意的文件有下面几个：

❖BoardConfig.mk：SDK 需要使用的编译选项配置。

❖board.dts：linux-5.4/board.dts 的软连接，用于内核的设备树文件。

❖buildroot：buildroot 构建根文件系统使用的详细配置。

❖linux-5.4：内核相关的配置。

❖sys_config.fex：系统参数配置，旧版本内核起到设备树的作用，但是在在 Linux5.4 以后版本，仅保留少部分模块 (如 ddr、串口 debug 等级) 配置功能，且多用于 SPL 和 u-boot 阶段。

❖uboot-board.dts：u-boot 阶段的设备树文件。

❖sys_partition.fex：镜像分区配置。

❖env.cfg：u-boot 阶段的环境变量。

我们可以发现，在 buildroot 目录下也存在部分上一级目录中同名的文件，这是一种覆盖关系，其覆盖关系如下：

1. bsp、buildroot、openwrt 等系统方案，属于并行目录，不会相互覆盖。

2. bsp、buildroot 等系统方案的 BoardConfig.mk、env.cfg、sys_partition.fex 会覆盖 default 目录的 BoardConfig.mk、env.cfg、sys_partition.fex。

3. 同级别目录的，BoardConfig_nor.mk、env_nor.cfg、sys_partition_nor.fex 会覆盖 BoardConfig.mk、env.cfg、sys_partition.fex。注意，仅当在 SDK 配置中选择 nor 方案才会。

4. 当 bsp、buildroot 等系统方案没有配置 BoardConfig.mk、env.cfg、sys_partition.fex，会使用 default 目录的 BoardConfig.mk、env.cfg、sys_partition.fex。

5. 注意，BoardConfig.mk 是包含关系，由系统方案的 BoardConfig.mk 往上层包含，如果系统方案和上层有相同属性，才会覆盖，否则视为增加。

## 4.1.方案配置

从上面的信息可以了解到，SDK 编译时，会根据 BoardConfig.mk 文件的配置进行初始化，决定依照何种方案开始编译打包。

以 evb1_auto 的 buildroot 方案为例，BoardConfig.mk 文件内容如下：

```
LICHEE_CHIP:=sun8iw20p1
LICHEE_PRODUCT:=t113_evb1_auto
LICHEE_BOARD:=t113_evb1_auto
LICHEE_FLASH:=
LICHEE_ARCH:=arm
LICHEE_KERN_DEFCONF:=config-5.4
LICHEE_KERN_DEFCONF_RECOVERY:=config-5.4-recovery
LICHEE_BUILDING_SYSTEM:=buildroot
LICHEE_BR_VER:=201902
LICHEE_BR_DEFCONF:=sun8iw20p1_t113_defconfig
LICHEE_COMPILER_TAR=arm/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz
LICHEE_BRANDY_VER:=2.0
LICHEE_BRANDY_DEFCONF:=sun8iw20p1_auto_t113_i_defconfig
LICHEE_REDUNDANT_ENV_SIZE:=0x20000
LICHEE_ROOTFS:=target-arm-linaro-5.3.tar.bz2
LICHEE_COMPRESS:=gzip
LICHEE_NO_RAMDISK_NEEDED:=y
```

evb1_auto/BoardConfig.mk

```
<t113_i/configs/evb1_auto/BoardConfig.mk
LICHEE_KERN_DEFCONF:=config-5.4
#LICHEE_KERN_DEFCONF_RECOVERY:=sun8iw20p1smp_t113_recovery_defconfig
LICHEE_BUILDING_SYSTEM:=buildroot
LICHEE_BR_VER:=201902
LICHEE_BR_DEFCONF:=sun8iw20p1_t113_defconfig
LICHEE_COMPILER_TAR=arm/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabi.tar.xz
LICHEE_BRANDY_DEFCONF:=sun8iw20p1_auto_t113_i_defconfig
LICHEE_REDUNDANT_ENV_SIZE:=0x20000
LICHEE_RTOS_PROJECT_NAME:=t113_i_c906_evb1_auto
```

evb1_auto/buildroot/BoardConfig.mk

我们主要关注其中这几项：

❖LICHEE_KERN_DEFCONF：内核 defconfig 文件名称，该文件位于 linux-5.4 目录中。

❖LICHEE_BR_VER：buildroot 的版本，会自动去 SDK/buildroot/目录中寻找对应版本。

❖LICHEE_BR_DEFCONF：buildroot 的默认配置，在对应 buildroot 版本源码所在路径。

❖LICHEE_COMPILER_TAR：交叉编译工具链路径，对应的交叉编译链压缩包应该存放在 SDK/prebuilt/kernelbuilt/中。

❖LICHEE_BRANDY_DEFCONF：这是 u-boot 的默认配置，在 SDK/brandy/brandy-2.0/u-boot-2018 中。

❖LICHEE_ROOTFS：若不想使用使用 buildroot 构建系统，用户可以通过修改该项对应的值，指向自己构建的其他根文件系统。在使用之前用户应该先拷贝对应的根文件

系统到 SDK/device/config/rootfs_tar/中。

## 4.2.SPL

从上面的的信息可以了解到，对于 SPL 阶段，我们大部分情况下只需要修改 sys_config.fex 文件即可。

该文件在 kernel 版本大于等于 5.4 版本开始，主要用于 SPL，u-boot 相关配置，SDK 在编译/打包时会根据该文件配置针对当前开发板做一些配置，我们主要需要修改的部分有下面几个：

### 4.2.1. 目标板平台相关信息

```
;------------------------------------------------------------------------
; debug_mode=0 表示关闭启动时 Boot0/SPL 的打印 log，默认为 8。
; 0：关闭所有打印
; 1：只显示关键节点打印
; 4：打印错误信息
; 8：打印所有 log 信息
;------------------------------------------------------------------------
; earseflag 表示是否执行烧写是是否执行擦除操作
; 0x0：不进行擦除。
; 0x01：进行擦除，private，secure storage、user data、mbr 分区保留，其他擦除。
; 0x11：进行擦除，private，secure storage 分区保留，其他擦除。
; 0x12：强制擦除(整个 flash)。
;------------------------------------------------------------------------
[platform]
eraseflag    = 1
debug_mode   = 8
```

### 4.2.2. 目标板存储设备信息

```
;------------------------------------------------------------------------
;storage_type    = boot medium, 0-nand, 1-sd, 2-emmc, 3-nor, 4-emmc3, 5-spinand -1
```

```
(defualt)auto  scan
    ;--------------------------------------------------------------------------------
    [target]
    storage_type      = -1      ；推荐使用自动识别方式，或根据实际使用的存储设备配置
    burn_key          = 0     ；是否支持 DragonSN_V2.0 烧录 sn 号。0-不支持，1-支持
    nand_use_ubi= 1         ；NAND 存储情况下是否使用 ubi 格式分区，1-使用，0-不使用
```

### 4.2.3. 目标板调试串口信息

```
    ;--------------------------------------------------------------------------------
    ；描述 gpio 的形式：Port:端口+组内序号<功能分配><内部电阻状态><驱动能力><输出电平状态>
    ;--------------------------------------------------------------------------------
    [uart_para]
    uart_debug_port  =  0
    uart_debug_tx     =  port:PE2<6><1><default><default>
    uart_debug_rx     =  port:PE3<6><1><default><default>
```

此处应该针对用户自己的底板配置来确定使用哪一个串口作为调试串口，建议使用串口0，否则还需要修改 buildroot 构建的根文件系统中相关的内容。但是对于串口 0 实际使用哪些 GPIO 不做要求，可以通过查看《T113-i_Series_Datasheet_V2.0.pdf》文档查找哪些引脚可以复用为串口 0。

### 4.2.4. 其他需求

默认情况下不推荐修改 SPL，如确实需要在 SPL 阶段执行一些特殊修改的用户，请参考《Linux_SPL-PUB_开发指南.pdf》文档。

## 4.3.u-boot

### 4.3.1. 设备树

uboot-board.dts 文件是 uboot 阶段初期使用设备树文件，只作为初始阶段使用，当 u-boot 执行到 sunxi_replace_fdt_v2 函数后，就会开始使用 kernel 阶段的设备树信息，所以该文件中只需要配置极少部分情况即可。

### 4.3.2. 环境变量

env.cfg 文件是 u-boot 阶段的环境变量文件，用户可自行配置，但是其中有几个地方比较特殊：

#### 4.3.2.1. bootcmd 变量

```
int sunxi_update_bootcmd(void)
{
    char  boot_commond[128];
    int   storage_type = get_boot_storage_type();
    memset(boot_commond, 0x0, 128);
    strncpy(boot_commond, env_get("bootcmd"), sizeof(boot_commond)-1);
    debug("base bootcmd=%s\n", boot_commond);

    if ((storage_type == STORAGE_SD) || (storage_type == STORAGE_EMMC) ||
        (storage_type == STORAGE_EMMC3) || (storage_type == STORAGE_EMMC0)) {
        sunxi_str_replace(boot_commond, "setargs_nand", "setargs_mmc");
        debug("bootcmd set setargs_mmc\n");
    } else if (storage_type == STORAGE_NOR) {
        sunxi_str_replace(boot_commond, "setargs_nand", "setargs_nor");
    } else if (storage_type == STORAGE_NAND) {
#ifdef CONFIG_SUNXI_UBIFS
#ifndef CONFIG_MACH_SUN8IW18
        if (nand_use_ubi()) {
            sunxi_str_replace(boot_commond, "setargs_nand", "setargs_nand_ubi");
            debug("bootcmd set setargs_nand_ubi\n");
        }
#endif
#else
        debug("bootcmd set setargs_nand\n");
#endif
    }
```

上面的代码存在于 SDK/brandy/brandy-2.0/u-boot-2018/board/sunxi/board_helper.c 文件中，意思是对于 bootcmd 变量的值，会根据实际的启动存储器设备来自动切换，切换规则是改变"setargs_nand"到对应环境，所以用户可以保持使用下面这样的配置，由 u-boot 自动切换。

```
bootcmd=run setargs_nand boot_normal
```

### 4.3.2.2. mmc_root 变量

```
int sunxi_update_partinfo(void)

        memset(part_name, 0, PARTITION_NAME_MAX_SIZE);
        if (storage_type == STORAGE_NAND) {
            sprintf(part_name, "nand0p%d", index);
        } else if (storage_type == STORAGE_NOR) {
            sprintf(part_name, "mtdblock%d", index);
        } else {
            sprintf(part_name, "mmcblk0p%d", index);
        }

        temp_offset = strlen((char*)info.name) + strlen(part_name) + 2;
        if (temp_offset >= PARTITION_SETS_MAX_SIZE) {
            printf("partition_sets is too long, please reduces "
                   "partition name\n");
            break;
        }
        sprintf(partition_index, "%s@%s:", info.name, part_name);
        if (root_partition && strncmp(root_partition, (char *)info.name, sizeof(info.name)) == 0)
            sprintf(root_part_name, "/dev/%s", part_name);
        if (blkoops_partition && strncmp(blkoops_partition, (char *)info.name, sizeof(info.name)) == 0)
            sprintf(blkoops_part_name, "/dev/%s", part_name);
        offset += temp_offset;
        partition_index = partition_sets + offset;

    }
    partition_sets[offset - 1] = '\0';
    partition_sets[PARTITION_SETS_MAX_SIZE - 1] = '\0';
    env_set("partitions", partition_sets);

#ifdef CONFIG_SUNXI_DM_VERITY
    if (*root_part_name != 0) {
        printf("set root to dm-0\n");
        env_set("verity_dev", root_part_name);
        if (storage_type == STORAGE_NAND)
            env_set("nand_root", "/dev/dm-0");
        else if (storage_type == STORAGE_NOR)
            env_set("nor_root", "/dev/dm-0");
        else
            env_set("mmc_root", "/dev/dm-0");
    }
#else
    if (*root_part_name != 0) {
        printf("set root to %s\n", root_part_name);
        if (storage_type == STORAGE_NAND)      gerrit-sdk, 3个月前 • da-t113-tina5.0-v1.0 release
            env_set("nand_root", root_part_name);
        else if (storage_type == STORAGE_NOR)
            env_set("nor_root", root_part_name);
        else
            env_set("mmc_root", root_part_name);
```

上面的代码存在于 SDK/brandy/brandy-2.0/u-boot-2018/board/sunxi/board_helper.c 文件中，意思是对于 xxx_root 变量的值，会根据实际的启动存储器设备来自动切换，切换规则是改变"xxx_root"到 root_part_name 值，一般情况下应该交由 u-boot 自动切换，但是对于使用 SD 卡的 eMMC 核心板用户，可能 mmc0 设备对应的并不是 SD 卡，从而导致启动卡住，无法进入文件系统。

```
[    0.584401] 001: cfg80211: failed to load regulatory.db
[    0.606698] 001: Waiting for root device /dev/mmcblk0p5 ...
[    0.779385] 001: usb 1-1: new high-speed USB device number 2 using sunxi-ehci
[    1.031897] 001: hub 1-1:1.0: USB hub found
[    1.035936] 001: hub 1-1:1.0: 4 ports detected
[    1.120311] 000: [SNDCODEC][sunxi_check_hs_detect_status][191]:plugin --> switch:3
[    1.389589] 000: random: fast init done
[    1.489365] 001: usb 1-1.3: new high-speed USB device number 3 using sunxi-ehci
[    1.919379] 001: usb 1-1.4: new high-speed USB device number 4 using sunxi-ehci
[    2.222961] 001: cdc_ether 1-1.4:2.0 eth1: register 'cdc_ether' at usb-sunxi-ehci-1.4, CDC Ethernet Device, 00:e0:99:d3:98:19
[    4.649368] 001:
[    4.649368] 001: insmod_device_driver
[    4.649368] 001:
[    4.649661] 001: sunxi_usb_udc 4100000.udc-controller: 4100000.udc-controller supply udc not found, using dummy regulator
```

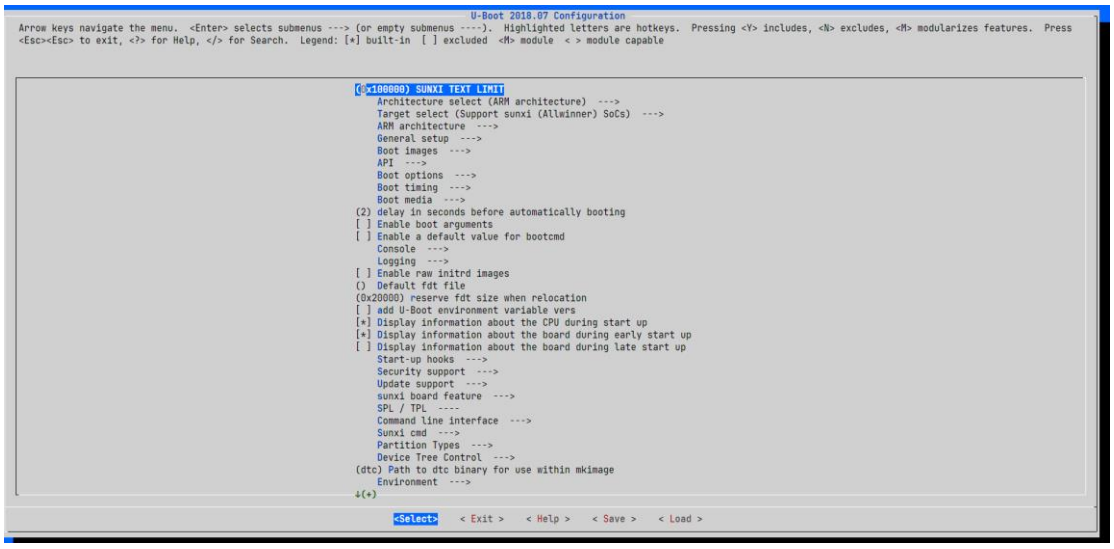
但是可以在看上面的代码，发现 root_part_name 赋值的要求是存储设备分区中能找到一个与环境变量中 root_partition 同名的分区，所以对于需要使用 SD 卡的用户，请不要尝试在 env.cfg 配置 root_partition 变量，然后自行依据实际情况修改 mmc_root 变量值，一般该变量值可能为/dev/mmcblk0p5 或者/dev/mmcblk1p5。

### 4.3.3. defconfig

u-boot 阶段的 defconfig 配置文件是根据 BoardConfig.mk 文件中 LICHEE_BRANDY_DEFCONF 项的值来确定的，在 evb1_auto 的 buildroot 方案中，该文件为 SDK/brandy/brandy-2.0/u-boot-2018/configs/sun8iw20p1_auto_t113_i_defconfig。一般不推荐修改，如果用户确实需要修改，请按照如下步骤进入 menuconfig 界面：

```
cd  brandy/brandy-2.0/u-boot-2018

make  menuconfig
```




需要注意的是，如果重新配置了 defconfig，则刚才通过 menuconfig 配置的会被覆盖掉。

## 4.4.内核

与内核相关的配置选项都存放在<SDK>/device/product/configs/<board>/linux-5.4 目录中，其中主要就是两个文件，一个为 board.dts，一个为 config-5.4。其中 config-5.4 文件可以任意修改文件名，只需要相应的修改 BoardConfig.mk 文件中 LICHEE_KERN_DEFCONF 选项的值即可，但是 board.dts 文件不应该修改文件名，SDK 在编译过程中会软连接该文件到 SDK/device/product/configs/<board>/board.dts，并最终生成 SDK/out/kernel/staging/sunxi.dtb

### 4.4.1. 设备树

从上面我们知道，当前的板级设备树配置文件是 board.dts，对于 board.dts 文件，大部分节点都有相应的注释描述解释供用户参考，但是有部分地方较为特殊：

1. spi-nand 节点

2. sdc0 节点

3. sdc2 节点

```
int sunxi_update_fdt_para_for_kernel(void)
#endif
    /* fix nand&sdmmc */
    switch (storage_type) {
    case STORAGE_NAND:
        fdt_enable_node("nand0", 1);
        fdt_enable_node("spi0", 0);
        break;
    case STORAGE_SPI_NAND:
#ifdef CONFIG_SUNXI_UBIFS
#ifdef CONFIG_MACH_SUN8IW18
        if (nand_use_ubi() == 0) {
            /* sun8iw18 aw-nftl config */
            fdt_enable_node("spinand", 1);
            fdt_enable_node("spi0", 0);
        }
#else
        /* sun50iw11 config */
        fdt_enable_node("spi0", 1);
        fdt_enable_node("/soc/spi/spi-nand", 1);
#endif
#else
        /* legacy config */
        fdt_enable_node("spinand", 1);      gerrit-sdk
        fdt_enable_node("spi0", 0);
#endif
        break;
    case STORAGE_EMMC:
        ret = fdt_enable_node("sunxi-mmc2", 1);
        if (ret)
            fdt_enable_node("mmc2", 1);
#ifdef CONFIG_SUNXI_SDMMC
        mmc_set_mmcblckx("sunxi-mmc2");
#endif
        break;
    case STORAGE_EMMC0:
        ret = fdt_enable_node("sunxi-mmc0", 1);
        if (ret)
            fdt_enable_node("mmc0", 1);
        break;
    case STORAGE_EMMC3:
        ret = fdt_enable_node("sunxi-mmc3", 1);
        if (ret)
            fdt_enable_node("mmc3", 1);
        break;
    case STORAGE_SD:
```

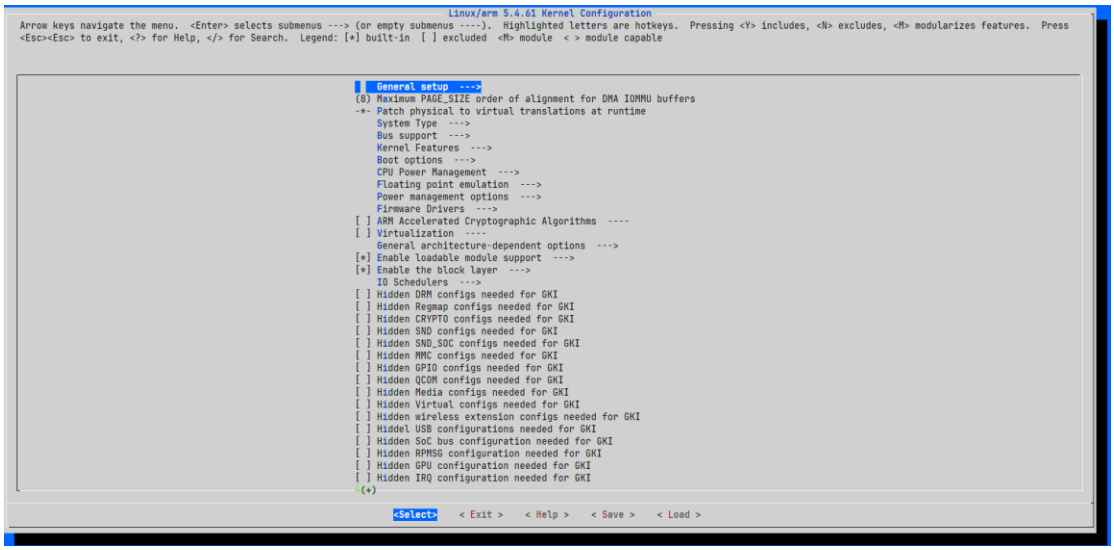
上面的代码存在于 SDK/brandy/brandy-2.0/u-boot-2018/board/sunxi/board_helper.c 文件中，意思是 u-boot 会根据启动时对应的存储设备信息来自动使能/失能 kernel 设备树中的部分节点，所以用户可能会发现虽然对应存储设备的设备节点配置为`status = "disabled"`依然可以正常进入系统。

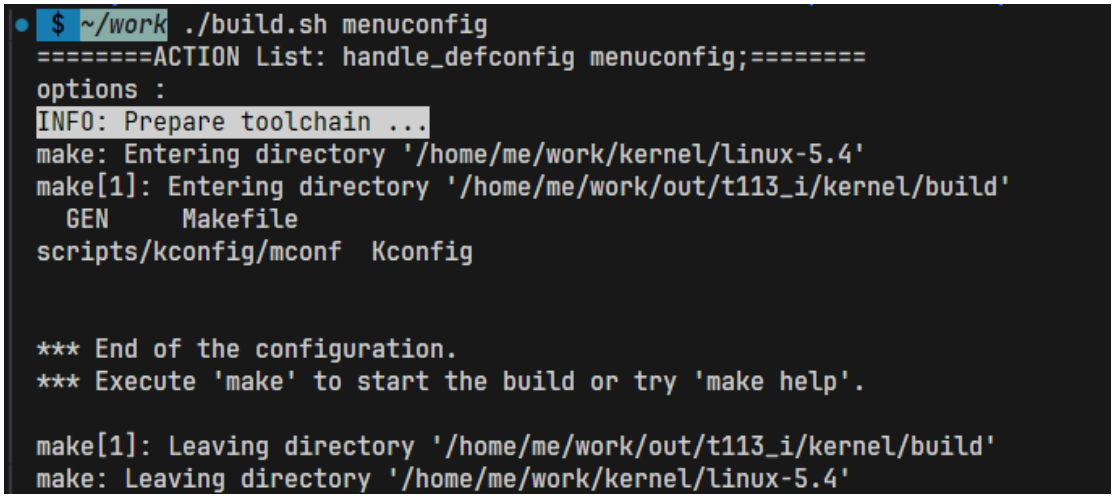
### 4.4.2. defconfig

从上面可以知道，板级的 defconfig 文件是根据 BoardConfig.mk 文件中 LICHEE_KERN_DEFCONF 值确定，配置文件一定存在于<SDK>/device/product/configs/<board>/linux-5.4 目录中。

此处以 evb1_auto 为例，其内核 defconfig 文件为<SDK>/device/product/configs/evb1_auto/linux-5.4/config-5.4。若用户需要修改配置，可在 SDK 顶层目录中使用下面的方式进入 menuconfig 界面进行配置：

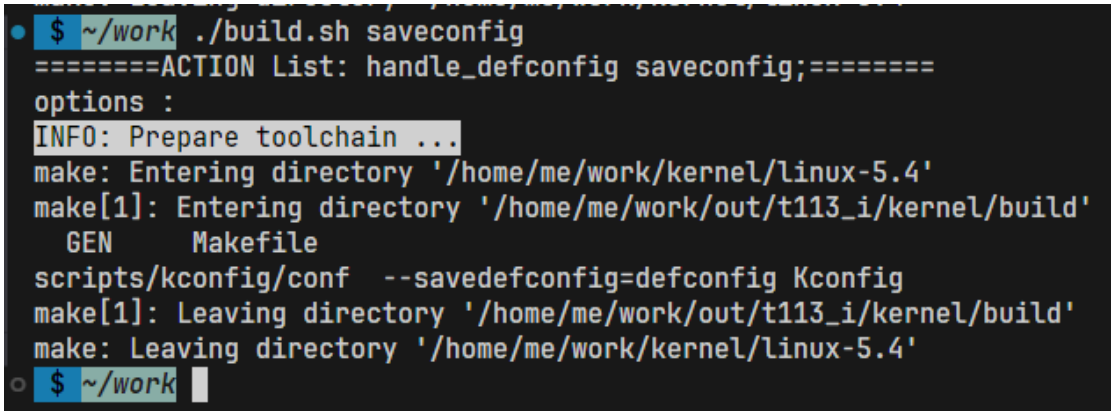
```
./build.sh menuconfig
```



修改完毕以后用户可以通过下面命令更新并保存新的配置到 config-5.4 文件：

```
./build.sh saveconfig
```




# 4.5.文件系统

## 4.5.1. Buildroot

从上文可以知道，与 buildroot 有关的板级相关配置存储在<SDK>/device/product/configs/<board>/[buildroot]/BoardConfig.mk 文件中，主要是 LICHEE_BR_VER 与 LICHEE_BR_DEFCONF。

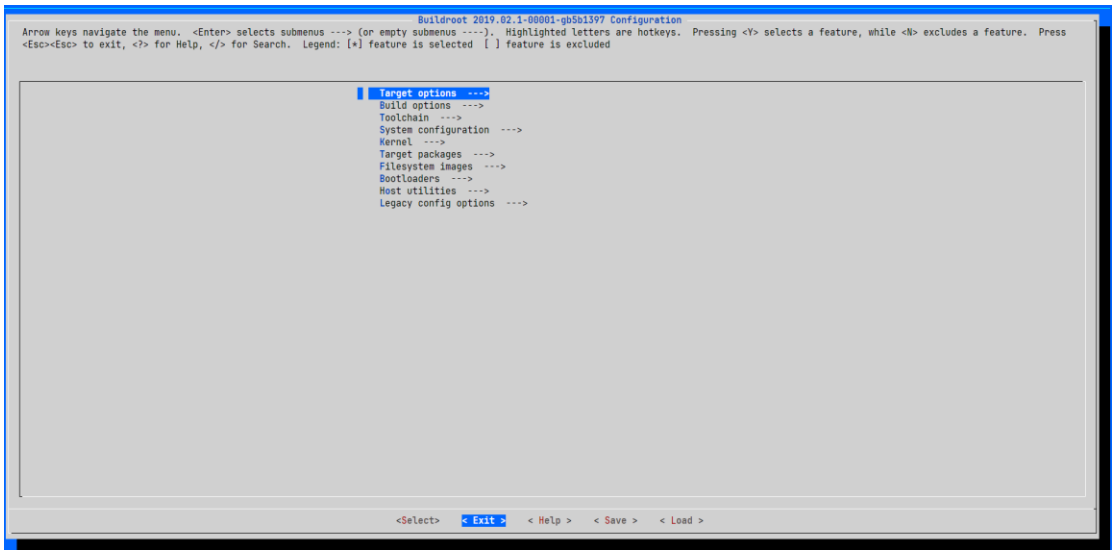
更多的 buildroot 相关使用资料请查阅 buildroot 官方手册：https://buildroot.org/downloads/manual/manual.html。

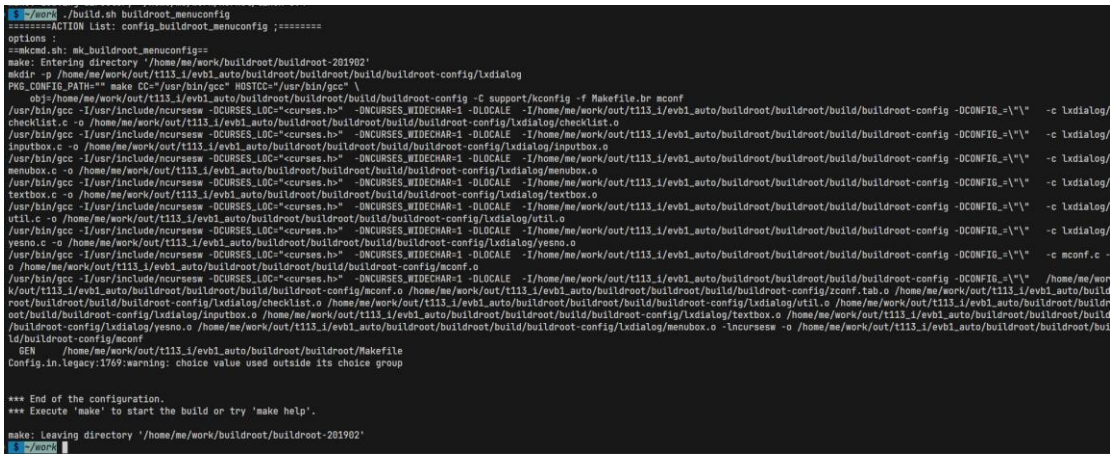
### 4.5.1.1. defconfig

此处我们以 ebyte_t113s4 为例，使用的 buildroot 版本为 201902，源码文件位于 SDK/buildroot/buildroot-201902，其编译配置文件位于<SDK>/buildroot/buildroot-201902/configs/sun8iw20p1_t113_defconfig。

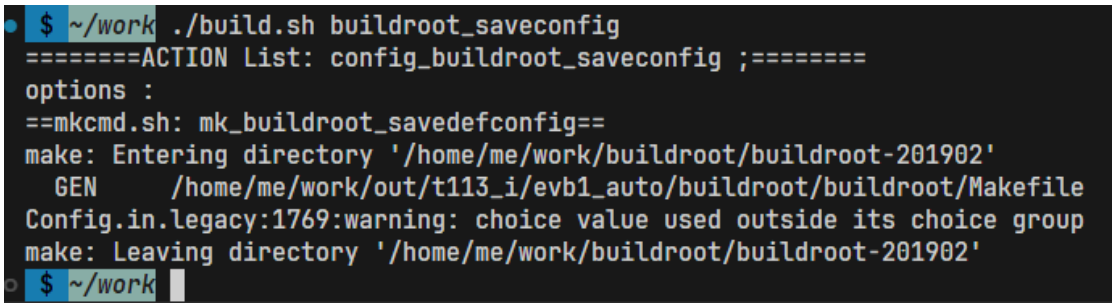
对于大多数用户，除了内核需要适配底板以外，还需要修改根文件系统以适配自己的使用需求，推荐使用 menuconfig 的方式修改配置，进一步的适配自己的方案需求。可以通过下面的命令进入 buildroot 的 menuconfig 界面：

```
./build.sh buildroot_menuconfig
```




修改完毕以后用户可以通过下面命令更新并保存新的配置到 sun8iw20p1_t113_defconfig 文件：




## 4.5.2. 修改文件系统为 buildroot202205

以 ECB30 的 T113-S4 为例，修改<SDK>/device/config/chips/t113_s4/configs/ebyte_t113s4/buildroot/BoardConfig.mk

修改：

LICHEE_BR_VER:=202205

LICHEE_BR_DEFCONF:=sun8iw20p1_t113_i_emmc_defconfig

然后执行修改./build.sh config

最后执行编译./build,sh

用户即可得到 buildroot2022 的文件系统镜像。但是亿佰特的所有功能验证均是在 build root2018 下，buildroot202205 需要用户自行验证或者添加功能。

#### 4.5.2.1. 无法登录

对于使用 buildroot 版本为 202205 的用户，如果出现无法登录或登录后发现需要连续输入两次相同内容才会在终端显示一次，如输入`aa`，终端显示`a`，请修改如下文件：`<SDK> /buildroot/config/buildroot/buildroot/post_build.sh`，在其中添加如下内容：

```
+                # change ttyAS0 to ttyS0
+                if grep -q "ttyAS0" ${TARGET_DIR}/etc/inittab; then
+                        echo "found ttyAS0, change it to ttyS0!"
+                        sed -i 's/ttyAS0/ttyS0/g' ${TARGET_DIR}/etc/inittab
+                fi
+

                #commented ttyS0, insert /bin/sh
                grep "#ttyS0::respawn:" ${TARGET_DIR}/etc/inittab >/dev/null
```

### 4.5.3. 修改文件系统为 ubuntu

这里以 T113I 为例，我们在 SDK 目录下执行,./build.sh config 来进行配置，选择 linux-> ubuntu->linux-5.4->t113_i->ebyte_t113i->default。最后执行./build.sh 和./build.sh pack 即可生成镜像。提供的 ubuntu 镜像的密码默认是 ebyte。

### 4.5.4. 分区

sys_partition.fex 文件描述了打包后镜像的分区情况，用户主要需要修改的部分为 rootfs 分区大小以及该分区应该放入的根文件系统，不得小于实际根文件系统大小，也不得大于存储设备物理可用大小，需要注意由于 ubi 方案保留了部分存储大小作为安全保障，实际可用大小是稍小于理论大小的。

1. 对于 eMMC 或 SD 卡存储设备，rootfs 分区应该大致如下面的情况：

```
[partition]
    name         = rootfs
    size         = 2097152
    downloadfile = "rootfs.fex"
    user_type    = 0x8000
```

2. 对于 NAND 存储设备，rootfs 分区应该大致如下面的情况：

```
[partition]
    name         = rootfs
    size         = 342216
    downloadfile = "rootfs-ubifs.fex"
    user_type    = 0x8000
```

## 4.6.RISC-V + DSP

对于 RISC-V 和 DSP 移植，客户可以分别参考《FreeRTOS_RTOS_系统_开发指南.pdf》与《DSP_客户_使用指南.pdf》。

# 5. 如何烧录更新系统镜像

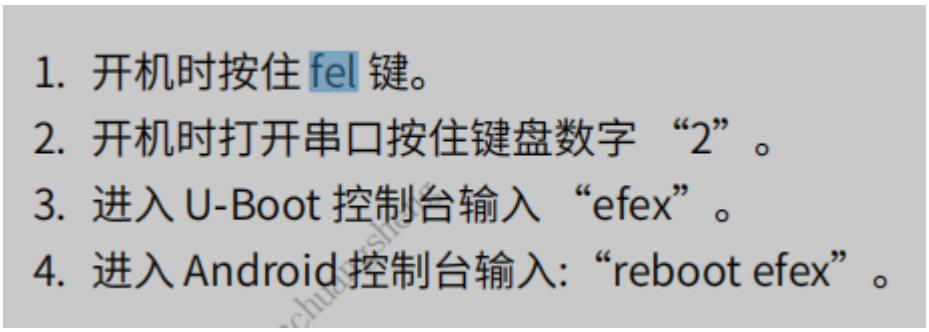
本章主要介绍基于 ECK30-T13IA 核心板的烧录更新方式。

## 5.1.官方工具烧录

本章节描述如何使用全志官方工具 PhoenixSuit 实现 USB 快速烧录功能。

### 5.1.1. 进入烧录模式

使用之前我们需要了解如何让小机进入烧写模式，按照官方文档，有如下四种方式：




1. 开机时按住 fel 按键

2. 开机时连接串口，按住数字键"2"

3. 进入 u-boot 控制台输入："efex"

4. 进入系统，在系统控制台输入："reboot efex"

一般常用方式为开机按住 fel 按键，这就需要用户在制作底板时拉出相应引脚并连接到一个按键上，这种方式无需存储设备上有可用系统。对于方式二，需要注意，这个功能是在全志官方提供的 boot 固件中实现的，如果用户在 sys_config.fex 中配置了错误的串口描述，方式二将失效，但是无需担心，还有下面两种办法解救：

1. 使用方式一，重新烧写修改正确的固件。

2. 通过后文《制作 SD 卡启动卡》方式，制作一个量产卡，重新烧录修改正确的固件。

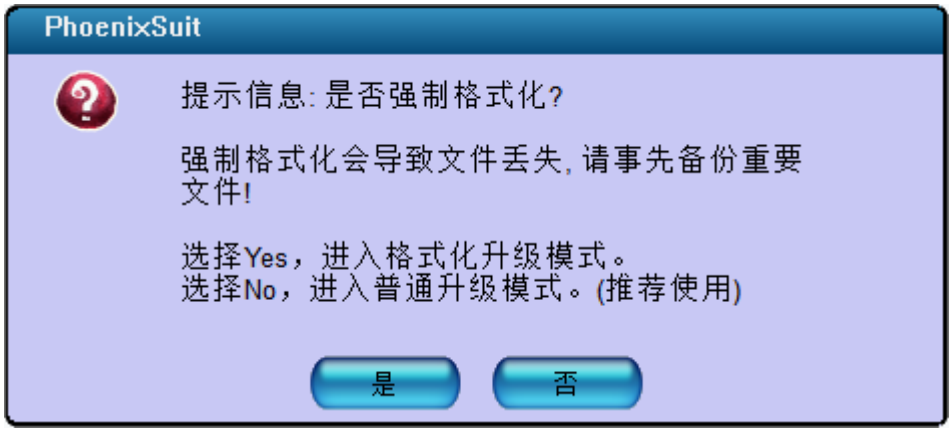
### 5.1.2. 烧录工具配置

再来看烧写工具，安装并打开该工具后，界面如下：


点击上方的“一键刷机” 项， 进入到镜像选择。我们提供的镜像分为 nand 版本和 emmc 版本，用户需要选择正确的镜像。ECB30 和 ECB31 系列目前都是使用 emmc 的核心板。选择后不需要点击立即升级按钮。


接下来我们连接两根 usb-typec 数据线，一根用来观察烧写日志，一根用来烧写板上 flash。连接好之后，我们需要进入烧写模式。

ECB30 引出了 FEL 按键，用户可以按下 FEL 然后保持，再按下复位按键或者给板子上电，即可进入烧写模式，弹出烧写询问框。ECB31 未引出 FEL 按键，用户可以先打开串口终端软件，使用小键盘在串口终端按住小键盘 2，然后再按下复位按键或者给板子上电，即可进入烧写模式，弹出烧写询问框。

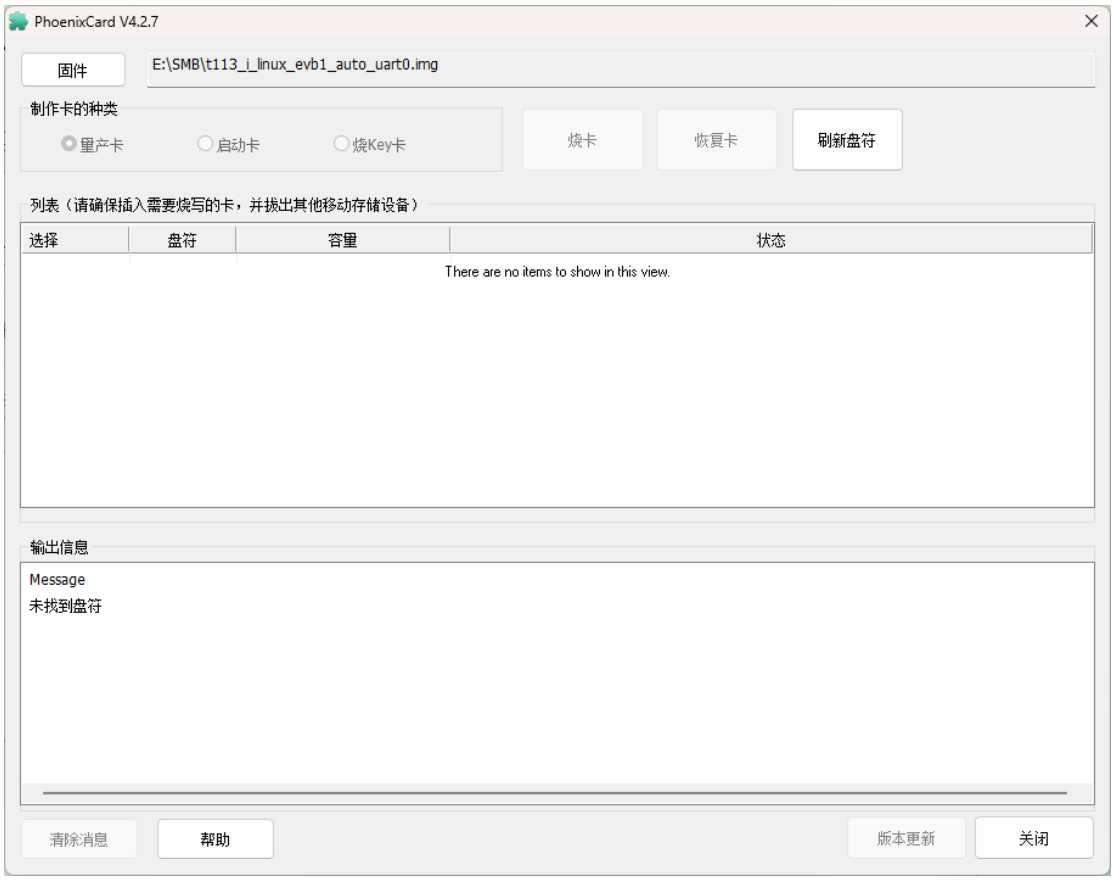



用户按需选择是否格式化升级。

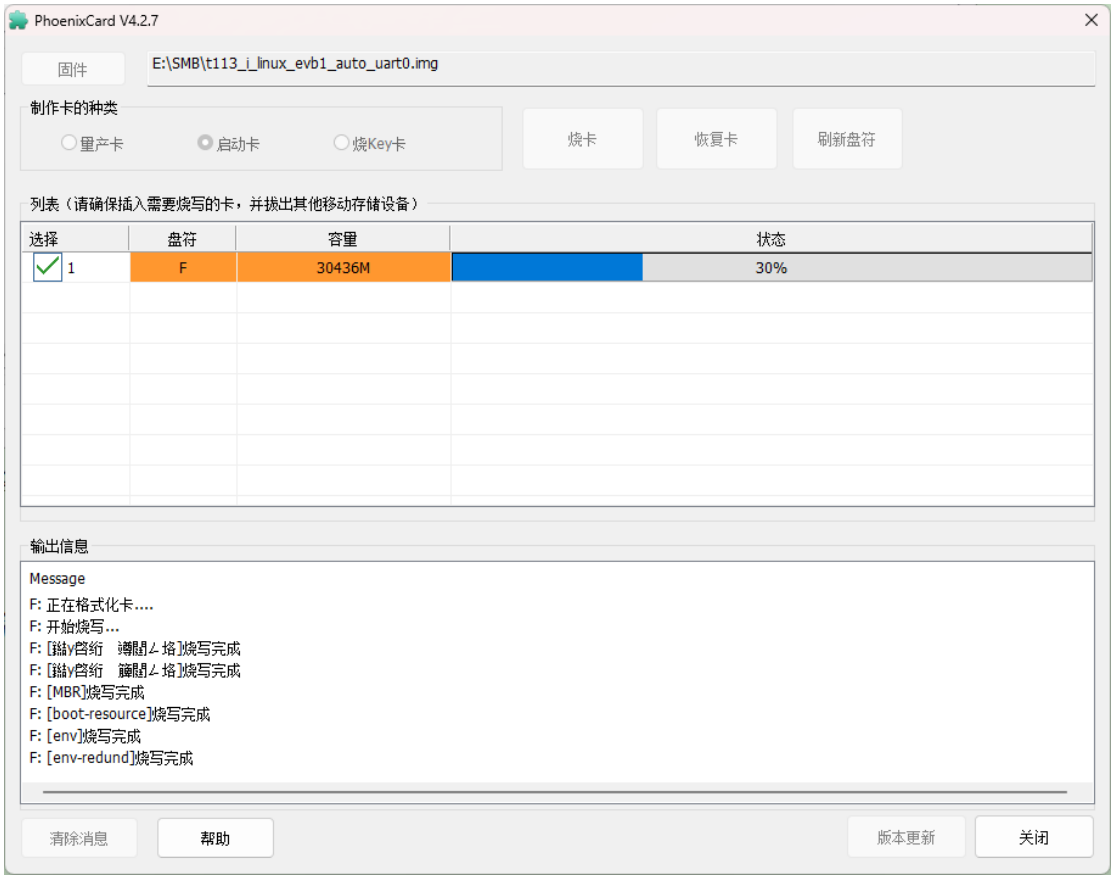
需要注意 T113 优先从 SD 卡进行启动，烧录到板上 flash 之后，拔掉 sd 卡才能从板上 flash 上启动。

## 5.2.制作 SD 卡启动卡

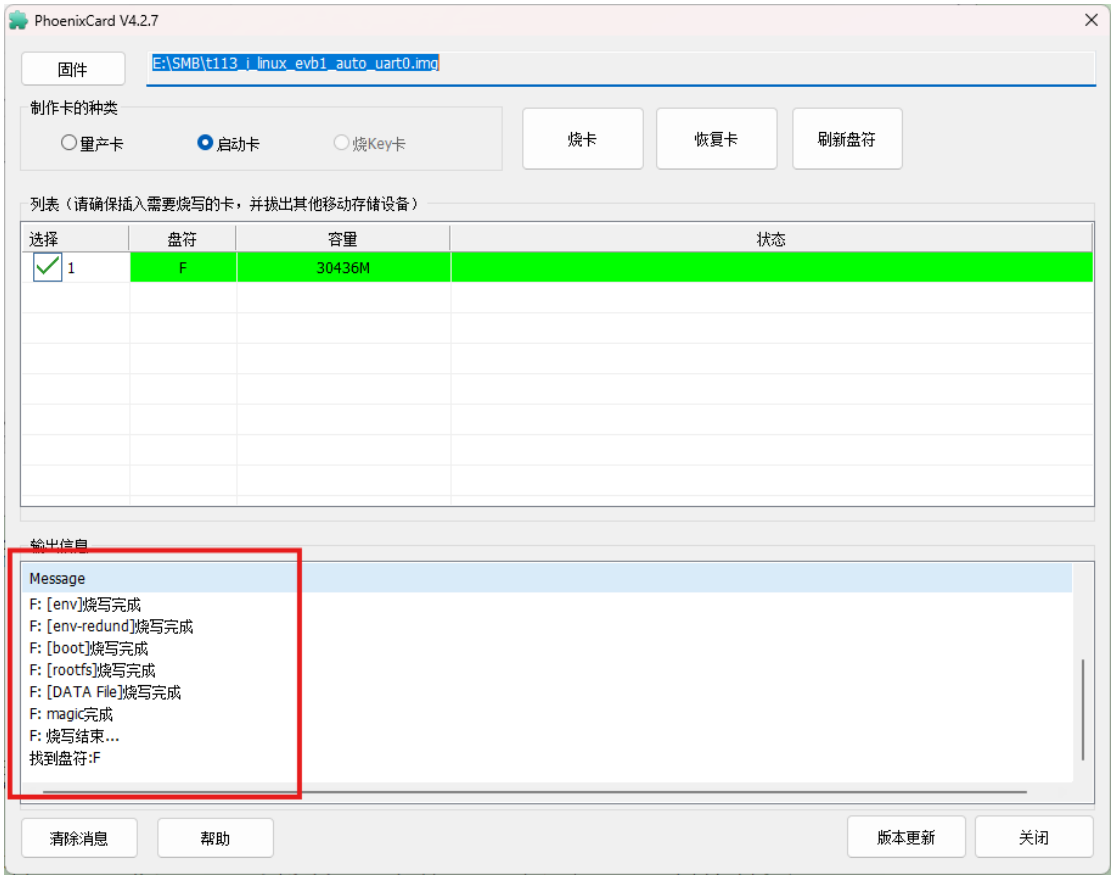
首先打开 PhoenixCard 程序，界面如下：




用户请依次选择待烧写固件，选择卡的种类为“启动卡”，而后选择待烧写的 SD 卡设备，该软件会自动扫描可用的移动存储设备，请注意不要选择了错误的设备。最后点击“烧卡”按钮，烧卡过程中会实时显示当前进度：




正式烧写完毕后界面如下，可以通过软件界面下方的输出信息查看是否确实制作启动卡成功：




## 5.3.量产卡烧录

在制作 SD 卡启动卡的时候我们还会发现其中存在一个量产卡的制作种类，该种方式可用通过制作一张 SD 卡来进行流水线操作的烧写功能，用户无需使用单独的镜像固件，直接使用启动卡阶段可以正确使用的镜像即可。

制作完毕以后从该量产卡启动系统，连接调试串口我们可以查看烧写过程全部的打印信息：

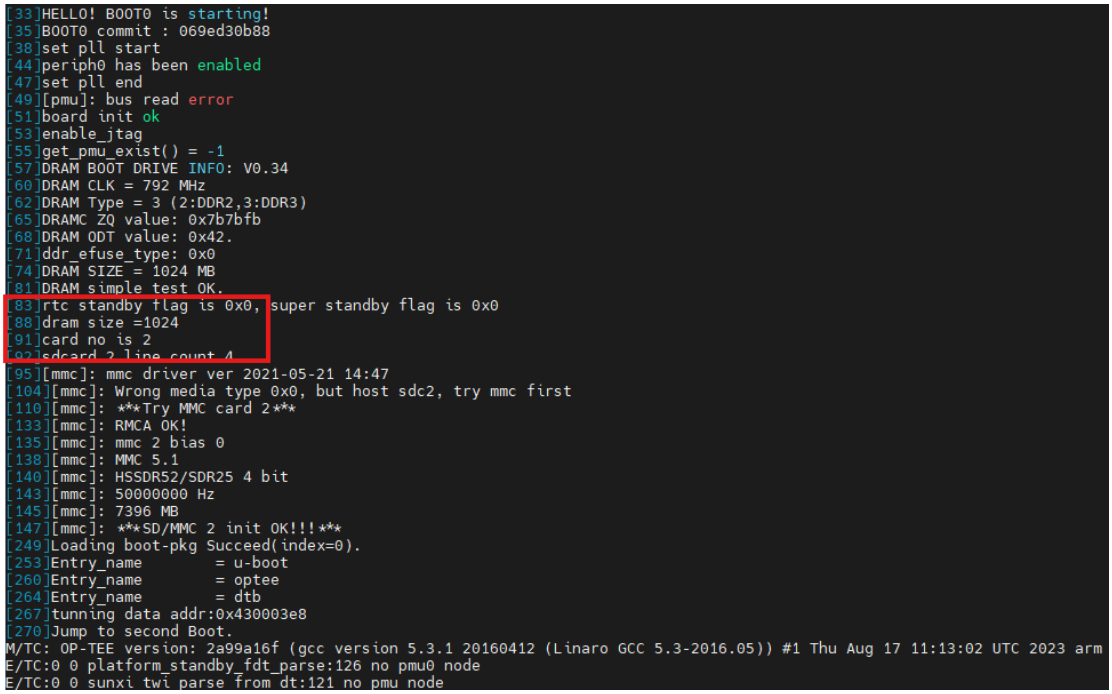
```
U-Boot 2018.07-g91003ae (Feb 06 2024 - 07:10:27 +0000) Allwinner Technology

[00.570]CPU:   Allwinner Family
[00.573]Model: sun8iw20
I2C:   ready
[00.594]optee version: major:3 minor:7
[00.598]DRAM:  [00.599]drm_base=0x0
[00.601]drm_size=0x0
[00.603]dram_base=0x40000000
[00.605]dram_size=0x40000000
1 GiB
[00.610]Relocation Offset is: 3ce93000
[00.640]secure enable bit: 0
E/TC:0   fdt_getprop_u32:336 prop trace_level not found
[00.654]CPU=1008 MHz,PLL6=600 Mhz,AHB=200 Mhz, APB1=100Mhz  MBus=300Mhz
[00.660]gic: sec monitor mode
[00.663]sunxi flash map init
SPI ALL:   ready
[00.668]line:719 init_clocks
[00.671]init_clocks:finish
[00.673]flash init start
[00.675]workmode = 17,storage type = 1
try card 2
set card number 2
get card number 2
[00.683][mmc]: mmc driver ver uboot2018:2023-07-4 16:18:00
[00.690][mmc]: Is not Boot mode!
[00.692][mmc]: SUNXI SDMMC Controller Version:0x50310
[00.704][mmc]: ************Try SD card 2************
[00.709][mmc]: mmc 2 cmd timeout 100 status 100
[00.713][mmc]: smc 2 err, cmd 8,  RTO
[00.716][mmc]: mmc 2 close bus gating and reset
[00.721][mmc]: mmc 2 cmd timeout 100 status 100
[00.725][mmc]: smc 2 err, cmd 55,  RTO
[00.729][mmc]: mmc 2 close bus gating and reset
[00.733][mmc]: ************Try MMC card 2************
[00.756][mmc]: mmc 2 cmd timeout 100 status 100
[00.760][mmc]: smc 2 err, cmd 8,  RTO
[00.763][mmc]: mmc 2 close bus gating and reset
[00.768][mmc]: mmc 2 cmd timeout 100 status 100
[00.772][mmc]: smc 2 err, cmd 55,  RTO
[00.776][mmc]: mmc 2 close bus gating and reset
[00.791][mmc]: gen_tuning_blk_bus8: total blk 10
[00.795][mmc]: gen_tuning_blk_bus4: total blk 6
[00.799][mmc]: Using 4 bit tuning now
[00.809][mmc]: write_tuning_try_freq: write ok
[00.814][mmc]: Pattern compare ok
[00.817][mmc]: Write tuning pattern ok
[00.820][mmc]: ================== HSSDR52_SDR25 ...
[00.825][mmc]: skip freq 400000
[00.828][mmc]: skip freq 25000000
[00.831][mmc]: freq: 2-50000000-64-4
```

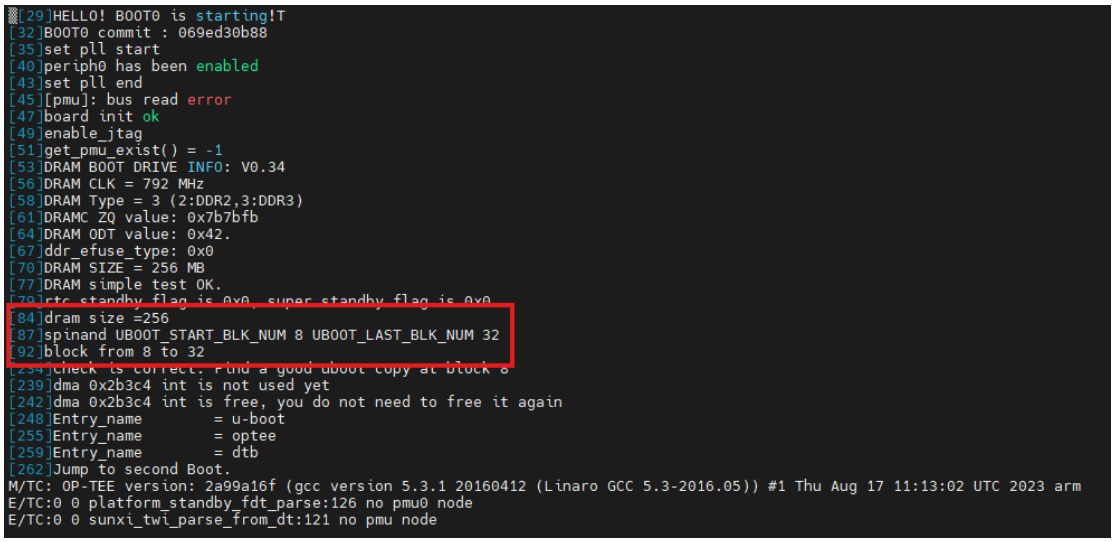
在系统引导进入 u-boot 阶段后开始正式烧写系统。

```
[61.690]successed in writting part dsp0
origin_verify value = 2e679ba6, active_verify value = 2e679ba6
[61.708]successed in verify part dsp0
[61.711]successed in download part dsp0
[61.716]successed in downloading part
uboot size = 0x168000
storage type = 2
sunxi_sprite_deal_uboot ok
[61.880]successed in downloading uboot
[61.886][mmc]: write mmc 2 info ok
dram para[0] = 318
dram para[1] = 3
dram para[2] = 7b7bfb
dram para[3] = 1
dram para[4] = 1102
dram para[5] = 4000000
dram para[6] = 1c70
dram para[7] = 42
dram para[8] = 18
dram para[9] = 0
dram para[10] = 4a2195
dram para[11] = 2423190
dram para[12] = 8b061
dram para[13] = b4787896
dram para[14] = 0
dram para[15] = 48484848
dram para[16] = 48
dram para[17] = 1620121e
dram para[18] = 0
dram para[19] = 0
dram para[20] = 0
dram para[21] = 770000
dram para[22] = 2
dram para[23] = b4056103
dram para[24] = 0
dram para[25] = 0
dram para[26] = 0
dram para[27] = 0
dram para[28] = 0
dram para[29] = 0
dram para[30] = 0
dram para[31] = 0
storage type = 2
[61.951]successed in downloading boot0
CARD OK
[61.955]sprite success
sprite_next_work=3
next work 3
SUNXI_UPDATE_NEXT_ACTION_SHUTDOWN
[64.963][mmc]: mmc exit start
[64.985][mmc]: mmc 2 exit ok
```

出现上图类似内容后用户可以断开开发板供电，拔出量产卡，而后重新上电即可，串口终端会打印如下启动信息用于确定是从 eMMC 上启动的：




对于 NAND 启动则是会显示如下内容：



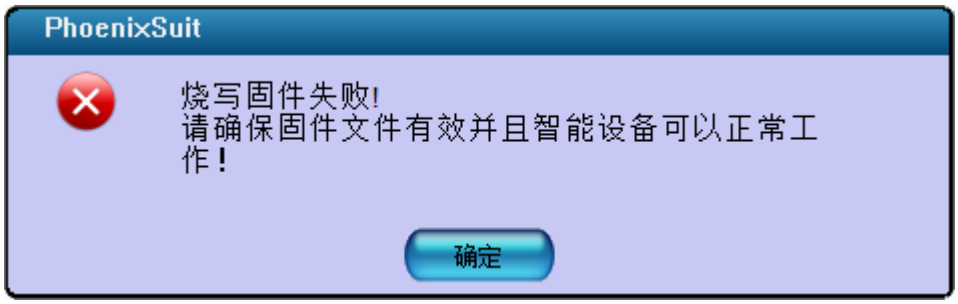

# 5.4.可能遇到的问题

## 5.4.1. 制作启动卡找不存储设备

用户在使用该工具烧写启动卡时，可能发现接入 SD 卡后该工具找不到相应存储设备，请确定该 SD 卡是否被 VMware 占用，若被占用请解除占用归还使用权给 host 主机，若依然不可用请尝试更换 USB 接口，或查看是否被其他程序占用。

## 5.4.2. 烧写固件失败

在制作 NAND 板镜像并烧录过程中可能会遇到烧写失败的情况，如烧写工具会显示如下信息：




此时请打开串口工具，连接调试串口，查看烧录过程中的报错情况。以下方这种情况为例：

```
defaults:
mtdids  : nand0=nand
mtdparts: mtdparts=nand:1024k@0(boot0)ro,3072k@1048576(uboot)ro,1024k@4194304(secure_storage)ro,-(sys)
[29.582]MTD info (4)
[29.584]pagesize: 0x1000
[29.586]blksize: 0x40000
[29.589]num  offset     bytes      name
[29.592]0    0x00000000 0x00100000 boot0
[29.596]1    0x00100000 0x00300000 uboot
[29.599]2    0x00400000 0x00100000 secure_storage
[29.604]3    0x00500000 0x0fb00000 sys
[29.607]MBR info (unalign):
[29.610]partno   addr       sects      type       name
[29.615]0        0x00000000 0x00008000 0x00000001 mbr
[29.619]1        0x00008000 0x00008686 0x00008000 boot-resource
[29.625]2        0x00010686 0x00000800 0x00008000 env
[29.630]3        0x00010e86 0x00000800 0x00008000 env-redund
[29.635]4        0x00011686 0x00008980 0x00008000 boot
[29.640]5        0x0001a006 0x00200000 0x00008000 rootfs
[29.645]6        0x0021a006 0x00000800 0x00008000 dsp0
[29.650]7        0x0021a806 0x00008000 0x00008000 private
[29.655]8        0x00222806 0x00000000 0x00008100 UDISK
[29.660]ubi attach the last part of mtd device: NO.3
[29.665]MBR info (align):
[29.667]partno   addr       sects      type       name
[29.672]0        0x00002800 0x000081f0 0x00000001 mbr
[29.677]1        0x0000a9f0 0x000087d8 0x00008000 boot-resource
[29.682]2        0x000131c8 0x000009d8 0x00008000 env
[29.687]3        0x00013ba0 0x000009d8 0x00008000 env-redund
[29.692]4        0x00014578 0x000089d0 0x00008000 boot
[29.697]5        0x0001cf48 0x002001f0 0x00008000 rootfs
[29.702]6        0x0021d138 0x000009d8 0x00008000 dsp0
[29.707]7        0x0021db10 0x000081f0 0x00008000 private
[29.717]ubi attach the last part of mtd device: NO.3
[29.722]ubi volume total size is larger than mtd size.
ubi_vol_total_bytes : 0x446a0000, mtd_bytes: 0xfb00000
[29.732]initialize sunxi spinand ubi failed
download_standard_gpt:write mbr sectors fail ret = 0
```

查看红色框中内容，会发现是 ubi 卷大小超过了实际设备存储大小。在联系上文中关于镜像打包和开发板适配部分内容可以知道，ubi 卷大小实际由`SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex`文件中的配置决定，此时我们需要修改其中个分区的大小，以保证不会超过物理可以存储大小限制，以下方为例，修改最大的 rootfs 分区大小，减少 ubi 卷大小：

```
[partition]
    name         = rootfs
    size         = 342216
    downloadfile = "rootfs-ubifs.fex"
    user_type    = 0x8000
```

同时我们也需要注意，该大小又不能小于实际 rootfs-ubifs.fex 文件大小。

## 5.5.现有系统更新

在之前章节“官方工具烧录”和“制作 SD 卡启动卡”中我们已经成功固化了 Linux 文件系统，系统也已经成功启动了。但有时候我们开发过程中，尝试自己编译制作 uboot、内核与文件系统，后续我们只想更新 uboot、设备树和文件系统中的某一个，特别是在开发过程中可能要频繁更新，为了节省开发时间，我们可以采用单独更新的方法来更新我们需要更新的部分，而不必花更长的时间去重新制作一张 SD 系统启动卡或者使用脚本固化系统到 eMMC 或 NAND 中。

开发主机和开发板文件互传的方法可以参考本文档“数据传输”章节。

### 5.5.1. 相关文件

想要烧写系统更新，不可避免我们需要了解相关文件路径，这里描述一下烧写相关文件都在哪里，它们的名字都是什么：

烧写文件

| 文件 | 文件名 | 真实路径 |
|---|---|---|
| Uboot 镜像 | boot_package.fex | SDK/out/t113_i/<board>/pack_out/boot_package.fex |
| 内核镜像 | boot.fex | SDK/out/t113_i/<board>/buildroot/boot.img |
| 根文件系统 | rootfs.fex | SDK/out/t113_i/tlt113-evm-emmc/buildroot/rootfs.ext4 |

### 5.5.2. 烧写位置

了解相关文件以后，我们还需要了解这部分文件应该烧写到什么位置：

烧写位置

| 文件 | 烧写位置(NAND) | 烧写位置(eMMC) |
|---|---|---|
| Uboot 镜像 | /dev/mtd1 | /dev/mmcblkX(seek 32800) |
| 内核镜像 | /dev/mtd8 | /dev/mmcblk0p4 |
| 根文件系统 | /dev/mtd9 | /dev/mmcblk0p5 |

对于内核镜像与根文件系统，还需要参考`SDK/device/config/chips/t113_i/configs/<board>/[buildroot]/sys_partition.fex`文件中的分区配置来确定。

### 5.5.3. eMMC + SD 卡

如果用户使用的是 eMMC 的核心板，想要在已有的系统中更新固件。我们需要从 eMMC 启动或者从 SD 卡启动替换自己固件。

#### 5.5.3.1. U-boot

T113-i 的 SDK 编译出来的 uboot，是一个打包后的文件 boot_package.fex，在烧写过程

中应该完整烧写。

1. 将编译生成的 boot_package.fex 文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下。

2. 查找待烧写设备路径，根据`ios`节点中`timing spec`描述确定该设备是 eMMC 还是 SD 卡，后文中以`mmc0`表示 eMMC 设备作为演示，用户请根据实际情况确定。

```
mount -t debugfs debugfs /sys/kernel/debug/
ls /sys/kernel/debug/mmc*
cat /sys/kernel/debug/mmc0/ios
cat /sys/kernel/debug/mmc1/ios
```

3. 写入 u-boot 镜像到 mmcblk0 设备

```
dd if=boot_package.fex of=/dev/mmcblk0 seek=32800
```

4. [可选]写入 u-boot 镜像到备份分区

```
dd if=boot_package.fex of=/dev/mmcblk0 seek=24576
```

5. 善后并重启

```
sync && reboot
```

#### 5.5.3.2. 内核镜像

T113-i 的 SDK 编译出来的内核，已经把 kernel 和设备树按照安卓镜像打包为一整个了，所以烧写时直接烧写该 boot.img 即可。

1. 将编译生成的内核与设备树文件通过 nfs/tftp/U 盘等方式拷贝到 eMMC/SD 卡系统中当前目录下。

2. 查找待烧写设备路径，根据`ios`节点中`timing spec`描述确定该设备是 eMMC 还是 SD 卡，后文中以`mmc0`表示 eMMC 设备作为演示，用户请根据实际情况确定。

```
mount -t debugfs debugfs /sys/kernel/debug/
ls /sys/kernel/debug/mmc*
cat /sys/kernel/debug/mmc0/ios
cat /sys/kernel/debug/mmc1/ios
```

3. 拷贝内核与设备树到 boot 分区。

```
dd if=boot.fex of=/dev/mmcblk0p4
```

4. 善后并重启

```
sync && reboot
```

### 5.5.3.3. 根文件系统

一般来说只能使用 SD 卡启动来更新 eMMC 的文件系统，或者通过 eMMC 启动来更新 SD 卡，正在运行的一个系统不能把自己给格式化！

1. 转换 rootfs.fex 到压缩包[在 ububtu 环境]

```
sudo apt install android-tools-fsutils
simg2img rootfs.fex rootfs.img
mkdir rootfs_dir
sudo mount rootfs.img rootfs_dir
cd rootfs_dir
sudo tar -cavf ../rootfs.tar.gz *
```

2. 通过 nfs/tftp/U 盘等方式拷贝转换好的根文件系统（rootfs.tar.gz）到 eMMC/SD 卡系统中。

3. 挂载 mmcblk0p5 分区

```
mkdir dest_dir
mount /dev/mmcblk0p5 ./dest_dir
```

4. 清空 mmcblk0p5 分区旧数据

```
rm -rf dest_dir/*
```

5. 拷贝新的根文件系统到 mmcblk0p5 分区

```
tar -xavf rootfs.tar.gz -C dest_dir/
sync
```

6. 取消挂载分区

```
umount dest_dir/
```

7. 重启系统

```
reboot
```

拷贝 rootfs.tar.gz 时，可能出现可用空间不足的问题，可尝试拷贝至`/mnt/UDISK`目录中，若用户无该目录，请挂载/dev/mmcblk0p8`分区到任意目录后拷贝 rootfs 到该目录。

## 5.5.4. NAND

如果用户使用的是 NAND 的核心板，想要在已有的系统中更新固件。我们需要从 NAND 自身启动。

### 5.5.4.1. U-boot

T113-i 的 SDK 编译出来的 uboot，是一个打包后的文件 boot_package.fex，在烧写过程中应该完整烧写

1. 将编译生成的 boot_package.fex 文件通过 nfs/tftp/U 盘等方式拷贝到系统中当前目录下。

2. 擦除旧有 u-boot

```
flash_erase  /dev/mtd1  0  0
```

3. 写入 u-boot 镜像到 mtd1 设备

```
dd  if=boot_package.fex  of=/dev/mtd1
```

4. 善后并重启

```
sync  &&  reboot
```

### 5.5.4.2. 内核镜像

T113-i 的 SDK 编译出来的内核，已经把 kernel 和设备树按照安卓镜像打包为一整个了，所以烧写时直接烧写该 boot.img 即可。

1. 将编译生成的内核与设备树文件通过 nfs/tftp/U 盘等方式拷贝到系统中当前目录下

2. 擦除旧有内核镜像

```
flash_erase  /dev/mtd8  0  0
```

3. 拷贝内核与设备树到 boot 分区。

```
dd  if=boot.fex  of=/dev/mtd8
```

4. 善后并重启

```
sync  &&  reboot
```

# 6. 参考资料

- ❖T113-i_Series_Datasheet_V2.0.pdf
- ❖T113-i_Series_User_Manual_V2.0.pdf
- ❖Linux kernel 开源社区：https://www.kernel.org/

# 7. 修订说明

修订说明表

| 版本 | 修改内容 | 修改时间 | 编制 | 校对 | 审批 |
|---|---|---|---|---|---|
| V1.0 | 初稿 | 2024-9-25 | WYQ | WFX | WFX |
| V1.1 | 修复遗漏的开发环境，解决 buildroot 202205 无法登录问题 | 2024-11-20 | WYQ | WFX | WFX |
| V1.2 | 添加 ubuntu 和修改若干操作描述 | 2025-3-17 | HSL | WFX | WFX |

# 8. 关于我们

销售热线：4000-330-990
技术支持：support@cdebyte.com 官方网站：https://www.ebyte.com
公司地址：四川省成都市高新西区西区大道 199 号 B5 栋

成都亿佰特电子科技有限公司
Chengdu Ebyte Electronic Technology Co.,Ltd.