Software Development Guide
T536 core board



Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

Table of contents

Disclaimer and Copyright Notice 1
1. Overview 2
2. Development environment preparation 2
2.1. Hardware Preparation 2
2.2. Virtual machine software deployment 3
2.3. Host software installation 5
2.4. Data Transfer 8
3. SDK Compilation and Build 10
3.1. SDK Acquisition 10
3.2. SDK Directory 11
3.3. SDK Configuration 11
3.4. Compile SDK 13
3.5. Separate compilation 14
3.6. Compilation Output 19
3.7. Image packaging 14
4. Development board adaptation 19
4.1. Solution Configuration 21
4.2. SPL 23
4.3. u-boot 25
4.4. Kernel 28
4.5. File System 29
4.6. RISC-V + DSP 31
5. How to burn and update the system image 31
5.1. Official tool burning 32
5.2. Make an SD card boot card 34
5.3. Mass production card burning 37
5.4. Possible problems 40
5.5. Existing system updates 42
6. References 46
7. Revision Notes 46
8. About Us ..... 错误！未定义书签。

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Disclaimer and Copyright Notice

The information in this document is subject to change without notice. The document is provided "as is" without warranty of any kind, including any warranty of merchantability, fitness for a particular purpose, or non-infringement, and any warranty otherwise provided by any proposal, specification or sample. No liability is assumed for this document, including liability for infringement of any patent arising from the use of the information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document.

The test data obtained in this article are all obtained by Ebyte Laboratory testing, and the actual results may be slightly different.

All trade names, trademarks and registered trademarks mentioned herein are the property of their respective owners and are hereby acknowledged.

The final right of interpretation belongs to Chengdu EBYTE Electronic Technology Co., Ltd.

Notice:

Due to product version upgrades or other reasons, the contents of this manual may be changed. EBYTE Electronic Technology Co., Ltd. reserves the right to modify the contents of this manual without any notice or reminder. This manual is only used as a guide. Chengdu EBYTE Electronic Technology Co., Ltd. tries its best to provide accurate information in this manual, but Chengdu EBYTE Electronic Technology Co., Ltd. does not ensure that the contents of the manual are completely error-free, and all statements, information and suggestions in this manual do not constitute any express or implied warranty.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1. Overview

This article mainly introduces the complete process of customizing a complete embedded Linux system based on the Ebyte core board, including the preparation of the development environment, code acquisition, and how to transplant the Boot loader and Kernel, customize the Rootfs suitable for your own application needs, etc. We first introduce how to build a system image suitable for the ECK30-T13IA core board based on the SDK source code we provide, and how to burn the built image to the development board. For those users who develop projects based on the ECK30-T13IA core board, we focus on the methods and key points of transplanting this system to the user's hardware platform, and through some actual transplantation cases and Rootfs customization cases based on Build root, users can quickly customize the system image suitable for their own hardware.

Please note that this document does not include the introduction of build root projects and basic knowledge related to Linux systems. It is suitable for embedded Linux system developers and embedded Linux BSP developers with certain development experience.

2. Development environment preparation

This chapter mainly introduces the necessary software and hardware environments required before using the ECK33 core board for development, including the necessary hardware configuration, and an introduction to the development and burning environments.

By reading this chapter, you will learn about the installation and use of relevant hardware tools and software development and debugging tools. You will also be able to quickly set up the relevant development environment, preparing for the subsequent development and debugging.

Host: i7-12700 + win11 22H2

Virtual machine: Ubuntu 22.04 LTS

2.1. Hardware Preparation

2.1.1. Product Model

Product Model Table

Serial number	Product specifications and models	Model Description
1	ECK33-B2N4E32-I	Allwinner T536 processor + 2T NPU + 4GB memory + 32GB eMMC, industrial-grade fully domestic core board.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2	ECK33-BCE2E16-I	Allwinner T536 processor + 2GB memory + 16GB eMMC, industrial-grade fully domestic core board.

2.1.2. Debug serial port

Debug serial port: baud rate 115200, data bits 8, stop bit 1, no parity check, no hardware flow control. Please configure the actual serial port yourself. The configuration method is detailed in the following text.

2.1.3. Startup method

According to the user manual of the T536 chip, T536will start in sequence according to the boot pin configuration. The startup method and sequence are detailed in the table below.

Startup sequence table

BOOT_PIN[1:0]	Boot sequence
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

Users of our ECK33 core board do not need to worry about this part, as the relevant products have implemented two corresponding configurations, `10` and `11`, on the board. What users should remember is that, generally speaking, SMHC0 is configured as an SD card function, and once a bootable SD card is inserted, the system will boot from the SD card. If the SD card cannot be started, it will boot from the onboard storage device.

2.2.Virtual machine software deployment

This chapter will introduce how to build the development environment of T113-i. By reading this chapter, you will understand the installation and use of related development and debugging tools. And you can quickly build the relevant development environment to prepare for subsequent development and debugging.

2.2.1. Virtual Machine Configuration

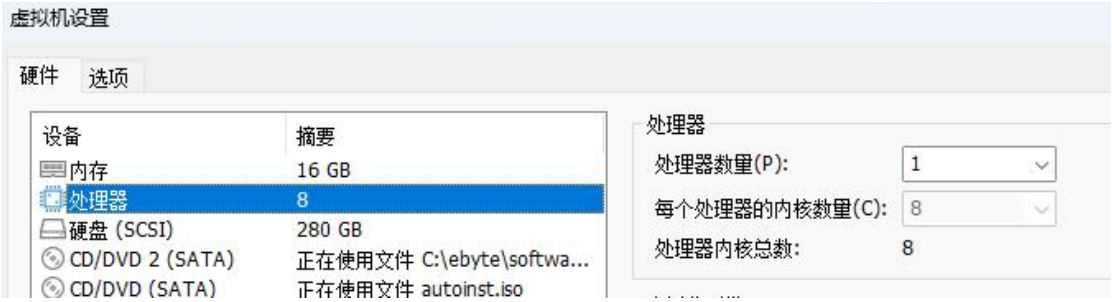
Currently, the more common virtual machine software includes Virtual Box, VM ware, etc. The configuration methods of different platforms are different, and user habits are also different. Therefore, please configure the virtual machine to use bridging to connect to the Internet according to the virtual machine software you use, and turn on the directory sharing function with Windows.

It is recommended that the virtual machine be set to have a memory size of 16 GB or more

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

and 8 cores, otherwise the compilation will fail.



2.2.2. Configure the compilation environment

Recommend users to use the Ubuntu 22.04 64-bit distribution.

Install software:

```
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
```

Create a working directory:

```
mkdir ~/work
```

2.2.3. Enable the TFTP service of the virtual machine

In the later debugging stage, TFTP is often used to load the kernel boot instead of burning the storage device every time. Therefore, you must first install and enable the TFTP service in Ubuntu. Use the following command to install the TFTP service:

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

By default, the files in the /var/lib/tftpboot directory will be shared, and can only be downloaded, not uploaded. If you want to customize the configuration, please refer to the official documentation:https://help.ubuntu.com/community/TFTP, this document has modified the path to /srv/tftp.

Add users to the tftp group and modify the shared directory permissions:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
sudo usermod -a -G tftp $(whoami)
sudo chmod 775 /srv/tftp/
sudo chgrp tftp /srv/tftp/
```

If the preceding operations are not performed, permission issues may occur and files cannot be written to the shared directory without root user permissions.

2.2.4. Enable the NFS service for the virtual machine

In the later debugging stage, you often need to share files with NFS or mount the file system directly through NFS. Therefore, you must first install and enable the NFS service in Ubuntu. Use the following command to install the NFS service:

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

Wait for the installation to complete. After the installation is complete, set the directory to be exported. Here, the default directory of tftp is used directly. Users can modify it by themselves, or use a convenient text editor to modify the /ext/exports file. For configuration methods, refer to the man manual:

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

2.2.5. Enable the SSH service of the virtual machine

After enabling Ubuntu's SSH service, we can use ssh client software to log in to Ubuntu under Windows, such as using MobaXterm, VSCode, and use the following command to enable SSH service under Ubuntu:

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

The above command installs the ssh service. The ssh configuration file is /etc/ssh/sshd_config. Use the default configuration. For more configuration, refer to the man manual related to sshd_config.

2.3.Host software installation

This chapter will introduce how to build the T113-i programming and debugging

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

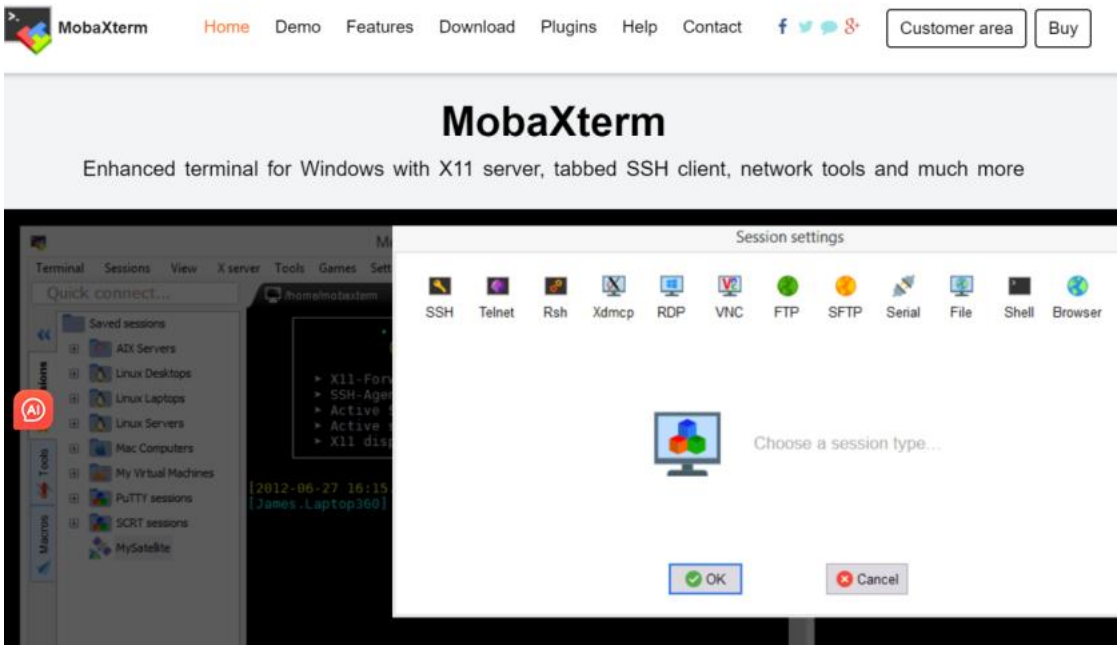
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

environment. By reading this chapter, you will understand the installation and use of related programming and debugging tools.

2.3.1. MobaXterm

2.3.1.1. Software download and installation

MobaXterm is a terminal software with powerful functions and is free. It is recommended to use this software as terminal debugging software. MobaXterm software can be downloaded from its official website athttps://mobaxterm.mobatek.net/, as shown in the figure:

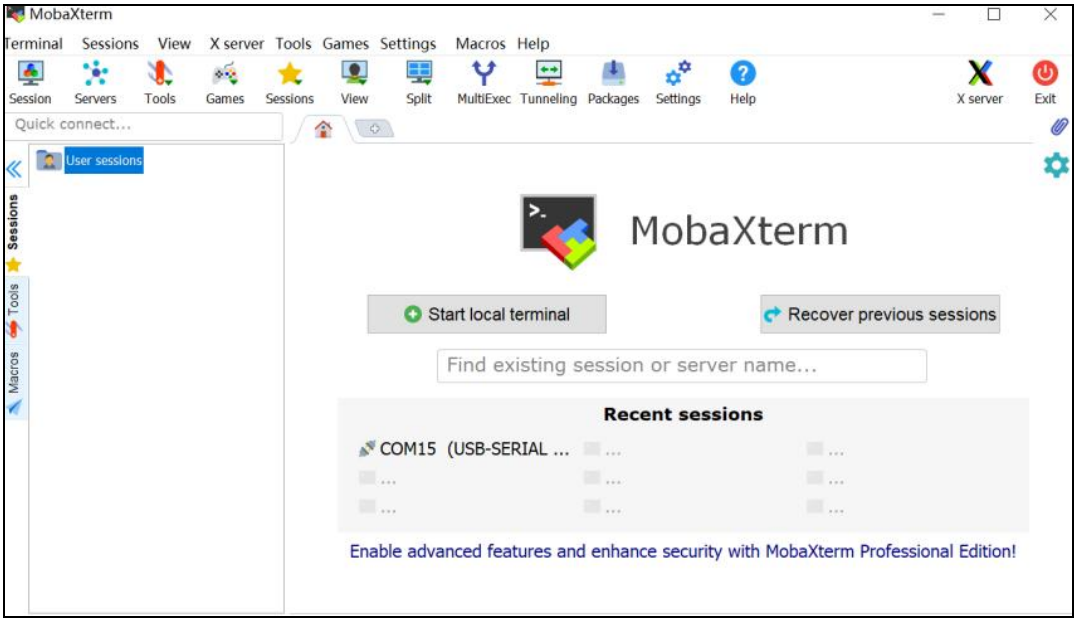


2.3.1.2. Software Usage

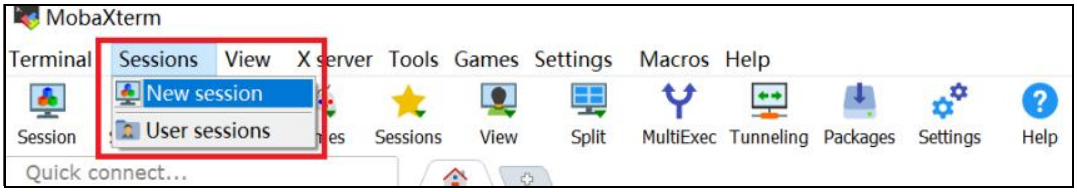
The software interface is shown in the figure:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

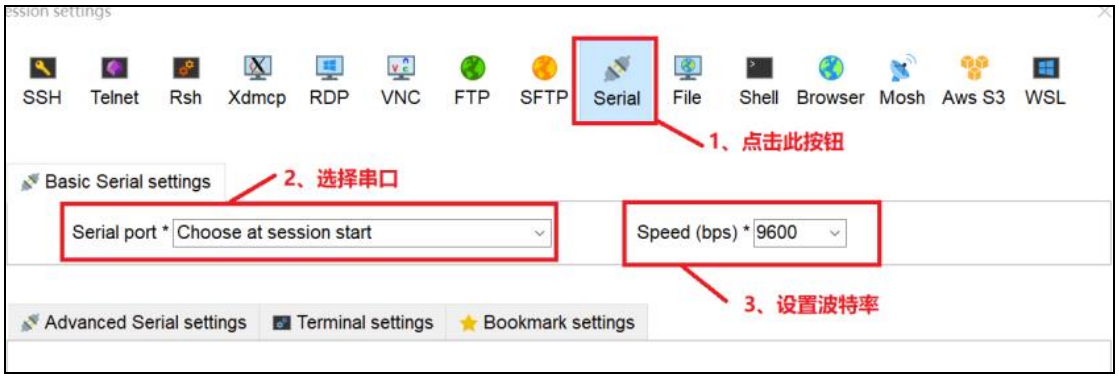
Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



Click the "Sessions->New session" button in the menu bar to open the new session window, as shown in the figure



Establish a Serial connection, that is, a serial port connection, because the main purpose of using MobaXterm is to use it as a serial port terminal. Click the "Serial" button in the figure to open the serial port setting interface, as shown in the figure:



After opening the serial port setting window, first select the serial port number to be set. Therefore, you must first connect the development board to the computer with a serial port cable. Since the default serial port baud rate used by T113-i is 115200, you need to set the baud rate to 115200.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.3.2. PhoenixSuit

This tool is officially provided by Allwinner to burn the system from USB. The software has been stored in the directory:“04_Tools/PhoenixSuit”. For detailed usage, please refer to the section “Official Tool Burning” below.

2.3.3. PhoenixCard

This is a tool officially provided by Allwinner for making system cards. You can choose to burn and make mass production cards and startup cards. The software has been stored in the directory: “04_Tools/PhoenixCard”. For detailed usage, please refer to the section “Making an SD Card Startup Card” below.

2.4.Data Transfer

In the previous chapters, we have configured various network services for the virtual machine. Now we just need to understand how to use them on the client and development board.

2.4.1. TFTP Client

In general, the development board only needs to use the download function. In this case, you can simply use the curl tool to download. Before using it, you need to select the curl function in buildroot, and then use curl to download the file (only for Linux version curl, Windows can choose to use filezilla):

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

1. /path/dist_file: Where to download, the name and path of the output file.

2. HOST: The IP address of the virtual machine. You need to pay attention to whether this address is accessible.

3. src_file: file directory and name. Note that you do not need to add the "TFTP_DIRECTORY" part of the configuration. For example, if the file is stored in /srv/tftp/aaa/bbb/ccc, the src_file value is "aaa/bbb/ccc".

2.4.2. NFS Client

In general, the development board only needs to use the mount command to mount the NFS file system, but it is necessary to install basic support and then mount NFS:

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

1. HOST: The IP address of the virtual machine. You need to pay attention to whether this

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

address is accessible.

2. path: The path on the NFS server to be mounted, such as /srv/tftp/aaa/bbb.

3. mnt_point: Where you want to mount the NFS file system, such as /mnt.

2.4.3. SSH

Generally speaking, we can also use the SFTP and SCP functions that come with SSH to download files, but there are not many cases of directly using commands to upload and download. More third-party tools are used, such as the SFTP function after using MobaXterm to log in to ssh, or VSCode using the remote-SSH plug-in to implement the download function during the development process. The way to use it is to right-click the mouse on the file to be downloaded, and then select Download.

2.4.4. USB flash drive/SD card

For SD cards, you need to use a card reader, and the actual usage is the same as that of a USB flash drive.

2.4.4.1. Copying data from a virtual machine

Users need to transfer the USB flash drive/SD card to the virtual machine according to the virtual machine software they are using. Generally, it will be mounted automatically and can be directly accessed in the file manager. After copying the data, uninstall the device in the file manager and then remove the device from the virtual machine.

If it is not mounted automatically, you can use the following two commands to mount it manually:

1. Method 1

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

2. Method 2

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

In this case, you need to have some understanding of the USB disk/SD card partitions before you can specify the Y value in sdXY. The X value is usually 'b', but you also need to judge the situation yourself based on the information returned by lsblk.

Use method 1 to mount. You can access it without root privileges and without specifying a

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

path, but you need to have the `udisks2` package installed in your system.

Manual uninstallation corresponds to the following command:

1. Method 1

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

2. Method 2

```
sudo umount /mnt_point
```

2.4.4.2. Copy data to the development board

For development boards, they are generally streamlined and may not have a udisks2 package or a GUI interface. Even if there is an interface, it may not support automatic mounting. However, users often have root permissions, so mount operations are generally performed from the command line.

1. Mount

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

2. copy

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

3. Unmount

```
umount /mnt_point
```

3. SDK Compilation and Build

This section describes the complete process from obtaining the SDK to compiling and generating a usable image.

3.1. SDK Acquisition

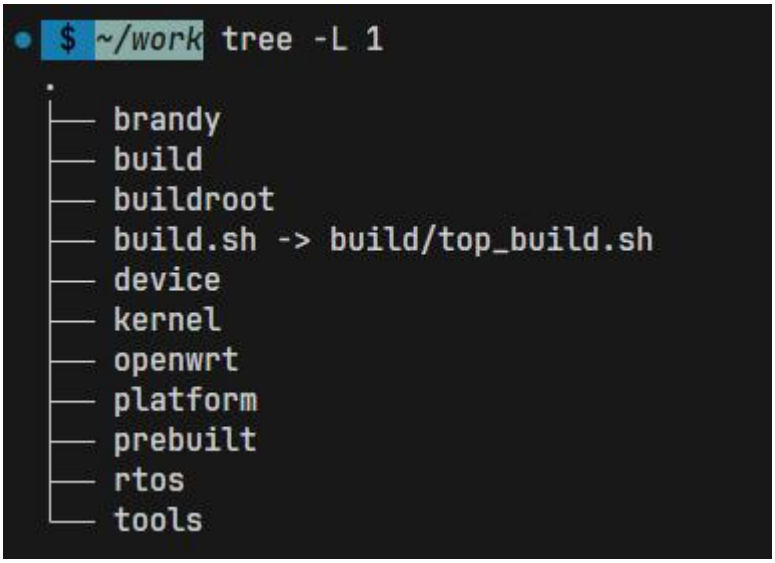
The T536 development SDK is included in the download file. Please unzip it to the development directory.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2. SDK Directory

After decompression, the entire SDK directory is as follows:



We need to pay attention to the following directories:

Top-level directory	Subdirectories	effect
Brandy	brandy-2.0/spl-pub/	SPL source code is generally not modified, and most of the official files provided by Allwinner are binary files
	brandy-2.0/u-boot-2018/	U-boot source code
build		Compile script directory, generally not modified
buildroot	buildroot-201902	File system source code
	buildroot-202205	
	package/auto/sdk_demo/	Sample Program
device	config/chips/t113_i	The chip configuration directory supported by the SDK, including configuration solutions for multiple development boards
	product	By default, no, after executing SDK configuration, point to the target chip, here is T113_i
kernel	linux-5.4	Kernel source code
openwrt	target/t113_i/	openwrt solution configuration
	openwrt/	openwrt source code
platform		Platform extension related source code, such as lvgl, qt, etc.
prebuilt		Kernel, buildroot and openwrt compilation tools
rtos		rtos source code, mainly related to heterogeneous cores and DSP
tools	tools_win	Some Windows tools

3.3. SDK Configuration

Since the SDK supports multiple chips and multiple solutions, users are required to select the configuration that matches their own development board before officially starting to compile. The SDK also provides the following configuration methods:

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

ECK33 selects Buildroot for Linux, ebyte_t536_default, and linux-5.10-origin.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
hu@hu:~/linux/t536/V1.1$ ./build.sh config
06-26 10:04:53.779 3284535 D mkcommon  : ========ACTION List: mk_config ;========
06-26 10:04:53.783 3284535 D mkcommon  : options :
All available platform:
   0. android
   1. linux
Choice [linux]:
All available linux_dev:
   0. bsp
   1. ubuntu
   2. buildroot
Choice [ubuntu]: 2
All available ic:
   0. t536
Choice [t536]:
All available board:
   0. demo
   1. demo_amp
   2. demo_kylo
   3. demo_nand
   4. demo_nor
   5. demo_raw_nand
   6. ebyte_t536
Choice [ebyte_t536]:
All available flash:
   0. default
   1. nor
Choice [default]:
All available kern_name:
   0. linux-5.10-euler
   1. linux-5.10-origin
   2. linux-5.10-rt
   3. linux-5.10-xenomai
   4. linux-5.15-origin
Choice [linux-5.10-origin]:
```

Take ECK33 as an example. The compilation configuration used here is `SDK/device/config/chips/t536/configs/ebyte_t536/buildroot/BoardConfig.mk`. After the configuration is completed, the final compilation rule configuration can be viewed in the `.buildconfig` file in the SDK directory. Below are the more important parts listed:

```
LICHEE_PLATFORM //Compile platform

LICHEE_LINUX_DEV //Compiled solution

LICHEE_IC // Compiled IC

LICHEE_BOARD //Compiled board configuration

LICHEE_FLASH //Compiled flash configuration

LICHEE_CHIP //Compiled chip code name

LICHEE_KERN_VER //Compiled kernel version

LICHEE_KERN_DEFCONF //Compile kernel default configuration
```

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

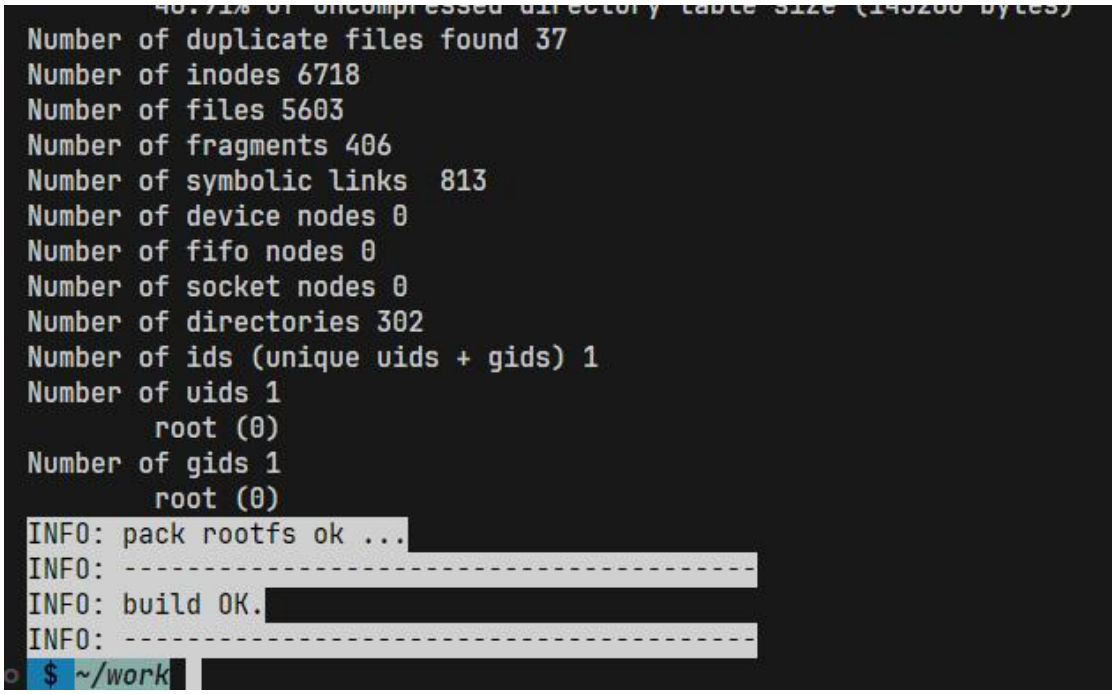
```
LICHEE_BUILDING_SYSTEM //Compiled construction system
LICHEE_BR_VER //buildroot version
LICHEE_BR_DEFCONF //Default configuration of buildroot
LICHEE_BR_RAMFS_CONF //buildroot's ramfs default configuration
LICHEE_BRANDY_VER //uboot version
LICHEE_BRANDY_DEFCONF //Default configuration of brandy
LICHEE_CROSS_COMPILER //Cross-compilation chain used for compilation
LICHEE_CHIP_CONFIG_DIR //Compile system IC configuration directory
LICHEE_OUT_DIR //Compiled output directory
```

3.4. Compile SDK

Wait for the SDK to finish configuration, then start compiling:

```
./build.sh
```

It is estimated that it will take about half an hour to compile and generate SPL, U-Boot, Linux kernel and Buildroot file system image files. After the compilation is completed, the following output will be displayed:



Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.5. Image packaging

After the previous compilation, various compiled files are output, but most of them cannot be used directly and need to be packaged into a final usable image:

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

After waiting for the packaging to be completed, check the out directory and you will find that there is an additional pack_out directory and an img file. The img file is the image file that needs to be burned to the development board in the end, and the pack_out directory is the intermediate file for packaging the image, some of which can also be used in the end.

Let's look at the pack_out directory again:

```
$ ~/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
```

You can find that it is a soft link from t113_i/<board>/pack_out:

```
$ ~/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
```

There is a lot of content, we mainly focus on the following files:

- boot_package.fex: packaged u-boot image, including contents, see boot_package.cfg
- boot-resource.fex: Resources used during the startup process, such as logo images.
- env.fex: u-boot environment variable configuration, copy of

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

SDK/device/product/configs/<board>/[buildroot]/env.cfg.

- ❖ boot.fex: soft link of the kernel image boot.img compiled above.
- ❖ rootfs*.fex: The soft link of the buildroot root file system compiled above.

3.5.1. Possible problems

The following situations may occur during the packaging process:

3.5.1.1. 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
```

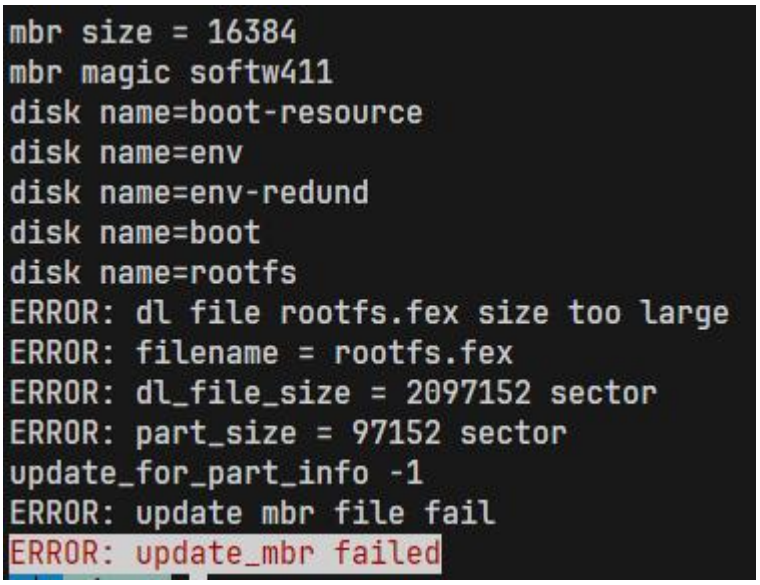
If the user does not need dsp, you can comment in SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex as follows.

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

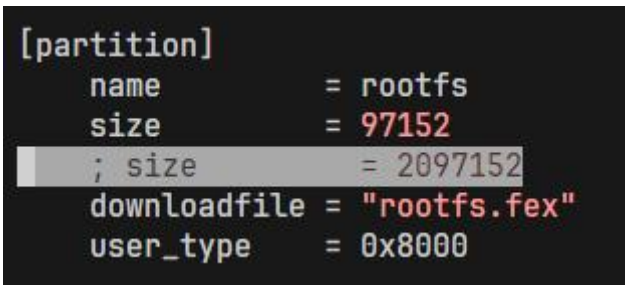
Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.5.1.2. dl file XXXX size too large



If this happens, it means that the partition size configured by the user does not match the final output file. You can find the partition corresponding to the filename in the figure in `SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex, and then modify the partition size to adapt to the corresponding file. The minimum partition size is described as dl_file_size in the figure above:



For users of nand + ubi solution, please also note that size here should be an integer multiple of 504, and the actual effective size of the ubi partition is size*252K.

3.6. Separate compilation

3.6.1. Compile u-boot separately

You can compile u-boot separately by using the following command, which can save compilation time when you only need to modify the u-boot stage:

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

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
$ ~/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
#
```

The following log is output after compilation:

```
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.
```

3.6.2. Compile the kernel separately

The kernel can be compiled separately by the following command, which can save compilation time when only the kernel stage needs to be modified:

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

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
$ ~/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
```

After compilation, the following log is output:

```
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'
```

3.6.3. Compile buildroot separately

You can compile buildroot separately with the following command, which can save compilation time when you only need to modify the buildroot stage:

```
./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']
```

After compilation, the following log is output:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
# @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.
```

3.7. Compilation Output

After the compilation is complete, you will find an out directory in the SDK directory, which stores all the compiled output of the SDK.

```
hu@hu:~/linux/t536/V1.1/out$ ls
ebyte_t536_uart0_linux-5.10-origin.img  kernel  pack_out  serversocket  t536  toolchain
```

The "kernel" directory is a symbolic link to the "t536/kernel" directory, representing the complete compilation output of the kernel. The "toolchain" directory contains the cross-compilation toolchain used during the compilation process.

Enter the <SDK>/out/out/t536/ebyte_t536/buildroot directory:

```
hu@hu:~/linux/t113/allwinner/out/t113_i/ebyte_t113i/buildroot$ ls
arisc                                boot0_nandfastboot_sun8iw20p1.bin  buildroot          ramfs.cpio.gz        sunxi.dtb                          vmlinux
bImage                               boot0_nand_sun8iw20p1.bin          dist               rootfs.ext4          System.map                         vmlinux.tar.bz2
boot0_mmc_car_fastboot_sun8iw20p1.bin  boot0_sdcard_sun8iw20p1.bin      dtc                rootfs.squashfs      t113_i_linux_ebyte_t113i_uart0.img  zImage
boot0_mmcfastboot_sun8iw20p1.bin     boot0_spinor_sun8iw20p1.bin        fes1_sun8iw20p1.bin  rootfs.ubifs         u-boot-sun8iw20p1.bin
boot0_nand_car_fastboot_sun8iw20p1.bin  boot.img                        lib                sboot_sun8iw20p1.bin  uImage
```

The main documents we need to focus on are the following ones:

- boot0_*_sun8iw20p1.bin: SPL image, used to boot u-boot
- u-boot-sun8iw20p1.bin: u-boot image, used to boot the kernel
- boot.img: kernel image, including kernel, device tree, Android image format
- sunxi.dtb: The compiled kernel device tree file will eventually be packaged into boot.img.
- zImage: The compiled kernel file will eventually be packaged into boot.img.
- rootfs.*: The root file system generated by buildroot. Usually the one with the suffix ext4 is used for eMMC/SD card, and the one with the suffix ubifs is used for NAND.

4. Development board adaptation

In order to adapt to the user's new hardware platform, the user needs to have a certain understanding of the CPU chip manual and the CPU chip user guide, and also needs to have a certain understanding of the SDK modification content. From the description of the SDK directory

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

above, we can know that the development board/solution configuration is stored in SDK/device/config/chips/t536/configs:

```
hu@hu:~/linux/t536/V1.1/device/config/chips/t536/configs$ tree -L 1
.
├── default
├── demo
├── demo_amp
├── demo_kylo
├── demo_nand
├── demo_nor
├── demo_raw_nand
└── ebyte_t536
```

In this level of directory, except for the default directory, all other directories can be considered to be different development board/solution configurations. The contents in the default directory are used as default configuration items when the required items are not found in the corresponding board-level configuration. Generally, they do not need to be modified and should not be modified.

Still taking evb1_auto as an example:

```
$ ~/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
```

Here are the files we need to pay attention to:

- BoardConfig.mk: The compilation option configuration required by the SDK.
- board.dts: Soft link of linux-5.4/board.dts, used for kernel device tree file.
- buildroot: Detailed configuration used by buildroot to build the root file system.
- linux-5.4: kernel-related configuration.
- sys_config.fex: system parameter configuration. The old version of the kernel plays the role of device tree. However, in Linux 5.4 and later versions, only a few modules (such as ddr, serial port debug level) configuration functions are retained, and they are mostly used in SPL and u-boot stages.
- uboot-board.dts: Device tree file at the u-boot stage.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

- sys_partition.fex: mirror partition configuration.
- env.cfg: environment variables in the u-boot stage.

We can find that some files with the same name in the upper directory also exist in the buildroot directory. This is an overwriting relationship. The overwriting relationship is as follows:

1. bsp, buildroot, openwrt and other system solutions are parallel directories and will not overwrite each other.

2. The BoardConfig.mk, env.cfg, and sys_partition.fex of bsp, buildroot, and other system solutions will overwrite the BoardConfig.mk, env.cfg, and sys_partition.fex of the default directory.

3. In the same level directory, BoardConfig_nor.mk, env_nor.cfg, and sys_partition_nor.fex will overwrite BoardConfig.mk, env.cfg, and sys_partition.fex. Note that this only happens when the nor solution is selected in the SDK configuration.

4. When the bsp, buildroot and other system solutions do not configure BoardConfig.mk, env.cfg, and sys_partition.fex, the BoardConfig.mk, env.cfg, and sys_partition.fex in the default directory will be used.

5. Note that BoardConfig.mk is an inclusion relationship, which is included by the BoardConfig.mk of the system solution to the upper layer. If the system solution and the upper layer have the same properties, they will be overwritten, otherwise they will be considered as added.

4.1. Solution Configuration

From the above information, we can understand that when the SDK is compiled, it will be initialized according to the configuration of the BoardConfig.mk file and decide which scheme to start compiling and packaging.

Taking the buildroot solution of evb1_auto as an example, the content of the BoardConfig.mk file is as follows:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
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

We mainly focus on the following:

- LICHEE_KERN_DEFCONF: The name of the kernel defconfig file, which is located in the linux-5.4 directory.
- LICHEE_BR_VER: The buildroot version, which will automatically search for the corresponding version in the SDK/buildroot/ directory.
- LICHEE_BR_DEFCONF: The default configuration of buildroot, in the path where the source code of the corresponding buildroot version is located.
- LICHEE_COMPILER_TAR: cross-compilation toolchain path. The corresponding cross-compilation chain compressed package should be stored in SDK/prebuilt/kernelbuilt/.
- LICHEE_BRANDY_DEFCONF: This is the default configuration of u-boot, in SDK/brandy/brandy-2.0/u-boot-2018.
- LICHEE_ROOTFS: If you do not want to use the buildroot build system, you can modify the value of this item to point to other root file systems you have built. Before using it, you should copy the corresponding root file system to SDK/device/config/rootfs_tar/.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2. SPL

From the above information, we can understand that for the SPL stage, in most cases we only need to modify the sys_config.fex file.

This file is used for SPL and u-boot related configurations starting from kernel version 5.4 or higher. SDK will make some configurations for the current development board based on this file configuration when compiling/packaging. The main parts we need to modify are as follows:

4.2.1. Target board platform related information

```
;------------------------------------------------------------------------------
; debug_mode=0 means turning off the printing log of Boot0/SPL during startup, the default is 8.
; 0: Turn off all printing
; 1: Only print key nodes
; 4: Print error message
; 8: Print all log information
;------------------------------------------------------------------------------
; eraseflag indicates whether to perform a burn or erase operation
; 0x0: Do not erase.
; 0x01: Erase, private, secure storage, user data, mbr partitions are retained, and others are erased.
; 0x11: Erase, private and secure storage partitions are retained, and others are erased.
; 0x12: Force erase (entire flash).
;------------------------------------------------------------------------------
[platform]
eraseflag = 1
debug_mode = 8
```

4.2.2. Target board storage device information

```
;------------------------------------------------------------------------------
```

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
;storage_type = boot medium, 0-nand, 1-sd, 2-emmc, 3-nor, 4-emmc3, 5-spinand -1(defualt)auto scan
;----------------------------------------------------------------------------
[target]
storage_type = -1 ; It is recommended to use automatic identification or configure according to the actual storage device used
burn_key = 0 ; Whether to support DragonSN_V2.0 to burn sn number. 0-not supported, 1-supported
nand_use_ubi= 1   ; Whether to use ubi format partition for NAND storage, 1-use, 0-do not use
```

4.2.3. Target board debugging serial port information

```
;----------------------------------------------------------------------------
; Describe the form of gpio: Port: port + group number <function allocation><internal resistance status><drive capability><output level status>
;----------------------------------------------------------------------------
[uart_para]
uart_debug_port = 0
uart_debug_tx   = port:PB9<2><1><default><default>
uart_debug_rx   = port:PB10<2><1><default><default>
```

Here, you should determine which serial port to use as the debugging serial port based on your own baseboard configuration. It is recommended to use serial port 0. Otherwise, you need to modify the relevant content in the root file system built by buildroot. However, there is no requirement for which GPIOs are actually used by serial port 0. You can check the " T536_Datasheet_V0.91.pdf " document to find out which pins can be reused as serial port 0.

4.2.4. Other requirements

By default, it is not recommended to modify SPL. If you really need to perform some special modifications in the SPL stage, please refer to the "Linux_SPL-PUB_Development Guide.pdf" document.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.3. u-boot

4.3.1. Device Tree

The uboot-board.dts file is the device tree file used in the early stage of uboot and is only used in the initial stage. When u-boot executes the sunxi_replace_fdt_v2 function, it will start to use the device tree information of the kernel stage, so only a few situations need to be configured in this file.

4.3.2. Environment variables

The env.cfg file is the environment variable file of the u-boot stage. Users can configure it by themselves, but there are several special places:

4.3.2.1. bootcmd variables

```
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
    }
```

The above code exists in the SDK/brandy/brandy-2.0/u-boot-2018/board/sunxi/board_helper.c file, which means that the value of the bootcmd variable will be automatically switched according to the actual boot memory device. The switching rule is to change "setargs_nand" to the corresponding environment, so users can keep using the following configuration and let u-boot automatically switch.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
bootcmd=run setargs_nand boot_normal
```

4.3.2.2. mmc_root variable

```
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);
```

The above code exists in the SDK/brandy/brandy-2.0/u-boot-2018/board/sunxi/board_helper.c file, which means that the value of the xxx_root variable will be automatically switched according to the actual boot memory device. The switching rule is to change "xxx_root" to the root_part_name value. Under normal circumstances, it should be automatically switched by u-boot. However, for users of eMMC core boards using SD cards, the mmc0 device may not correspond to the SD card, which may cause the boot to get stuck and fail to enter the file system.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

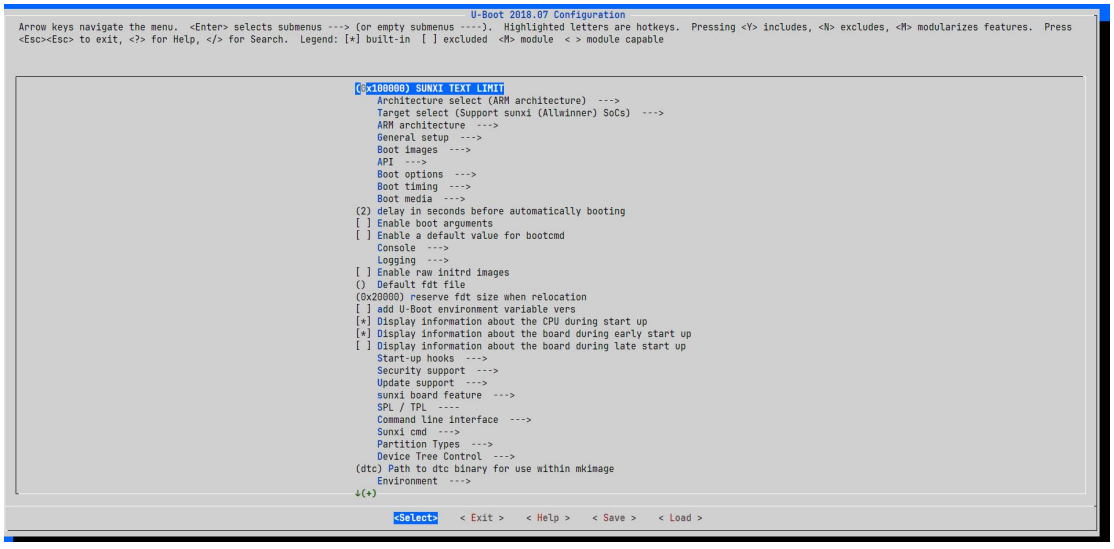
```
[    0.584401] 001: cfg80211: failed to load regulatory.db
[    0.606698] 001: Waiting for root device /dev/mmcblk0p5 ...
[    0.779385] 001: usb 1-1: new high-speed USB device number 2 using sunxi-ehci
[    1.031897] 001: hub 1-1:1.0: USB hub found
[    1.035936] 001: hub 1-1:1.0: 4 ports detected
[    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
```

However, you can see from the above code that the requirement for assigning the value of root_part_name is that a partition with the same name as the root_partition in the environment variable can be found in the storage device partition. Therefore, for users who need to use SD cards, please do not try to configure the root_partition variable in env.cfg, and then modify the mmc_root variable value according to the actual situation. Generally, the variable value may be /dev/mmcblk0p5 or /dev/mmcblk1p5.

4.3.3. defconfig

The defconfig configuration file in the u-boot stage is determined by the value of the LICHEE_BRANDY_DEFCONF item in the BoardConfig.mk file. In the buildroot solution of evb1_auto, the file is SDK/brandy/brandy-2.0/u-boot-bsp/configs/sun55iw6p1_t536_defconfig. It is generally not recommended to modify it. If the user really needs to modify it, please follow the steps below to enter the menuconfig interface:

```
./build.sh uboot_menuconfig
```




```
./build.sh uboot_saveconfig
```

Execute the above command to save the configuration.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.4. Kernel

According to device/config/chips/t536/configs/ebyte_t536/buildroot/BoardConfig.mk, the configuration file for the kernel is buildroot_linux_defconfig. The device tree is board.dts.

4.4.1. Device Tree

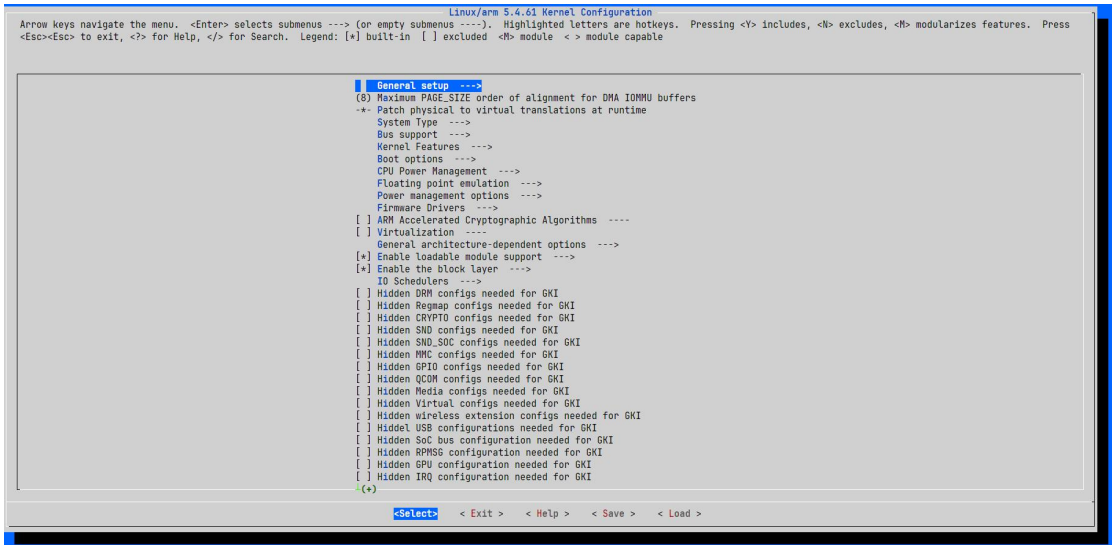
From the above, we know that the current board-level device tree configuration file is board.dts. For the board.dts file, most nodes have corresponding comments and explanations for users to refer to.

4.4.2. defconfig

From the above, we can know that the board-level defconfig file is determined by the LICHEE_KERN_DEFCONF value in the BoardConfig.mk file, and the configuration file must exist in the <SDK>/device/product/configs/<board>/linux-5.4 directory.

Here we take evb1_auto as an example, and its kernel defconfig file is <SDK>/device/product/configs/evb1_auto/linux-5.4/config-5.4. If the user needs to modify the configuration, he can use the following method to enter the menuconfig interface in the SDK top-level directory to configure:

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



Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
$ ~/work ./build.sh menuconfig
========ACTION List: handle_defconfig menuconfig;========
options :
INFO: Prepare toolchain ...
make: Entering directory '/home/me/work/kernel/linux-5.4'
make[1]: Entering directory '/home/me/work/out/t113_i/kernel/build'
  GEN     Makefile
scripts/kconfig/mconf  Kconfig


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

make[1]: Leaving directory '/home/me/work/out/t113_i/kernel/build'
make: Leaving directory '/home/me/work/kernel/linux-5.4'
```

After the modification is completed, the user can update and save the new configuration to the config-5.4 file using the following command:

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

```
$ ~/work ./build.sh saveconfig
========ACTION List: handle_defconfig saveconfig;========
options :
INFO: Prepare toolchain ...
make: Entering directory '/home/me/work/kernel/linux-5.4'
make[1]: Entering directory '/home/me/work/out/t113_i/kernel/build'
  GEN     Makefile
scripts/kconfig/conf  --savedefconfig=defconfig Kconfig
make[1]: Leaving directory '/home/me/work/out/t113_i/kernel/build'
make: Leaving directory '/home/me/work/kernel/linux-5.4'
$ ~/work
```

4.5. File System

4.5.1. Buildroot

From the above, we can know that the board-level configuration related to buildroot is stored in the <SDK>/device/product/configs/<board>/[buildroot]/BoardConfig.mk file, mainly LICHEE_BR_VER and LICHEE_BR_DEFCONF.

For more buildroot related information, please refer to the buildroot official manual:https://buildroot.org/downloads/manual/manual.html.

4.5.1.1. defconfig

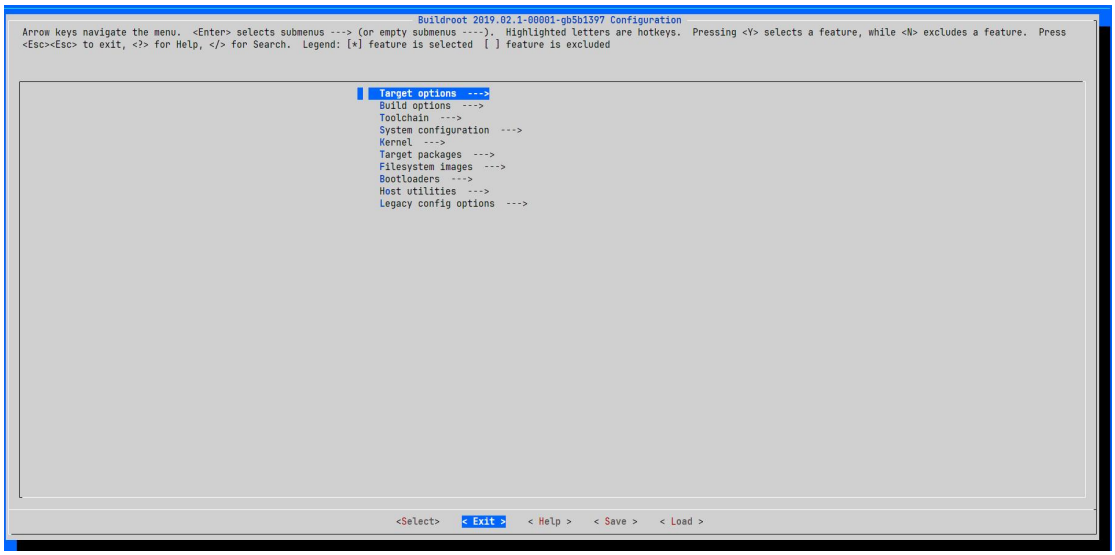
Here, we take ebyte_t536 as an example. The used buildroot version is 202205, and the source code file is located at SDK/buildroot/buildroot-201902. The compilation configuration file is located at <SDK>./buildroot/buildroot-202205/configs/sun55iw6p1_t536_defconfig.

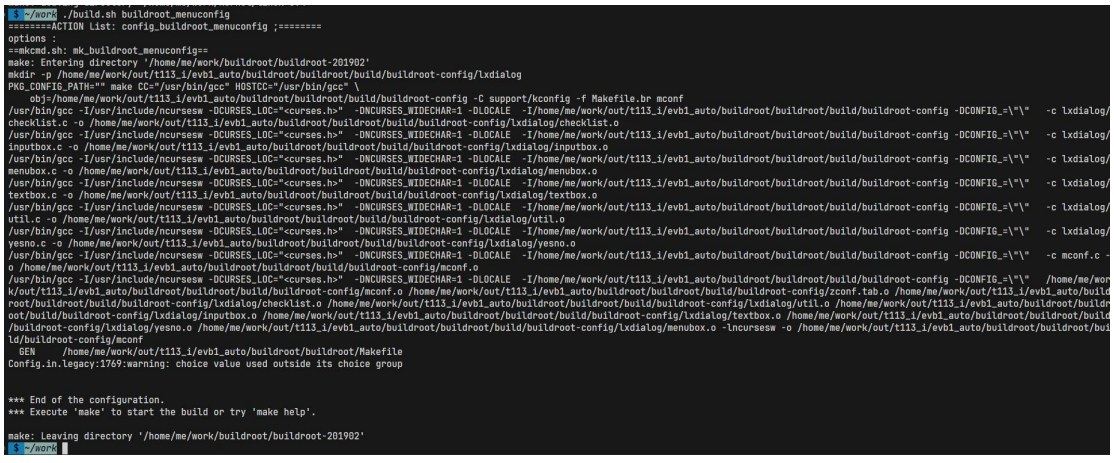
Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

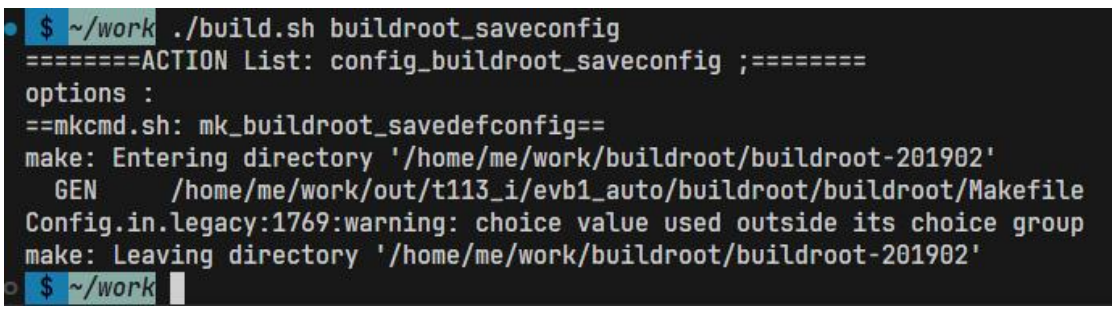
For most users, in addition to adapting the kernel to the baseboard, they also need to modify the root file system to adapt to their own usage requirements. It is recommended to use menuconfig to modify the configuration to further adapt to their own solution requirements. You can enter the menuconfig interface of buildroot with the following command:

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





After the modification is completed, the user can update and save the new configuration to the sun55iw6p1_t536_defconfig file using the following command:



Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

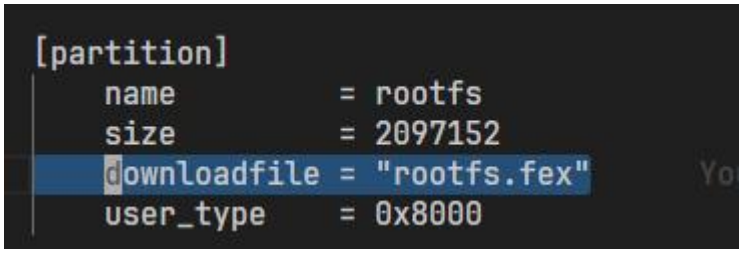
4.5.2. Change the file system to ubuntu

We execute "./build.sh config" in the SDK directory to make the configuration. Select "linux" -> "ubuntu" -> "t536" -> "ebyte_t536" -> "default". Finally, execute "./build.sh" and "./build.sh pack" to generate the image. The default password for the provided ubuntu image is "root".

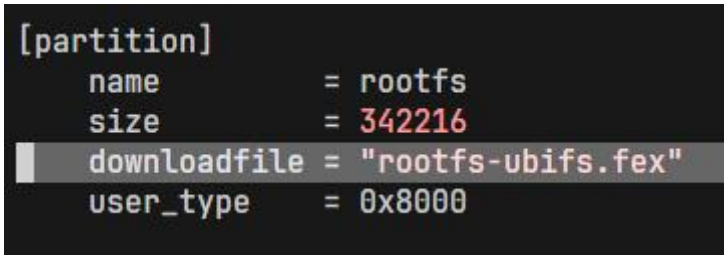
4.5.3. Partition

The sys_partition.fex file describes the partitioning of the packaged image. The main parts that users need to modify are the rootfs partition size and the root file system that should be placed in the partition. It must not be smaller than the actual root file system size, nor larger than the physical available size of the storage device. It should be noted that since the ubi solution reserves part of the storage size as a safety guarantee, the actual available size is slightly smaller than the theoretical size.

1. For eMMC or SD card storage devices, the rootfs partition should be roughly as follows:



2. For NAND storage devices, the rootfs partition should look like the following:



4.6. RISC-V + DSP

For RISC-V and DSP porting, customers can refer to "FreeRTOS_RTOS_System_Development Guide.pdf" and "DSP_Customer_User Guide.pdf" respectively.

5. How to burn and update the system image

This chapter mainly introduces the burning and updating method based on the ECK33 core board.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

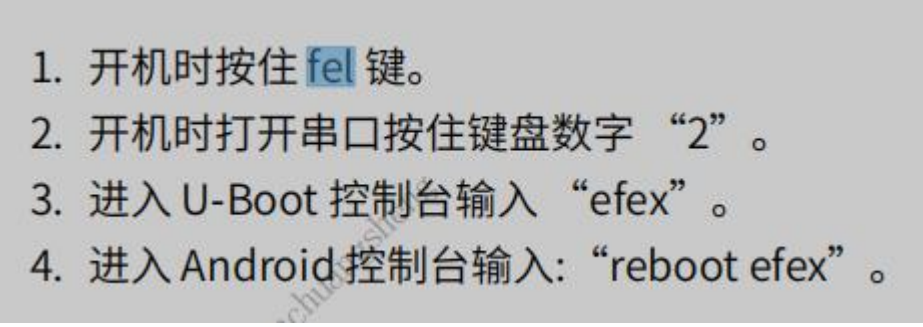
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.1. Official tool burning

This section describes how to use the Allwinner official tool PhoenixSuit to implement USB fast burning function.

5.1.1. Enter burning mode

Before using, we need to understand how to put the machine into programming mode. According to the official documentation, there are four ways:



1. Press and hold the FEL button while booting

2. Connect the serial port when booting, press and hold the number key "2"

3. Enter the u-boot console and enter: "efex"

4. Enter the system and enter in the system console: "reboot efex"

The common method is to press and hold the FEL button when turning on the computer. This requires the user to pull out the corresponding pins and connect them to a button when making the baseboard. This method does not require an available system on the storage device. For method 2, please note that this function is implemented in the boot firmware provided by Allwinner. If the user configures an incorrect serial port description in sys_config.fex, method 2 will fail. However, there is no need to worry. There are two other solutions:

1. Method 1: Re-burn and modify the correct firmware.

2. Make a mass production card by following the method in "Making an SD card boot card" and re-burn and modify the correct firmware.

5.1.2. Burning tool configuration

Use the official Allwinner tool PhoenixSuit to achieve rapid USB burning.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



Click on the "One-Click Flashing" option at the top to enter the image selection. After choosing ECB33, there is no need to click the "Upgrade Now" button.

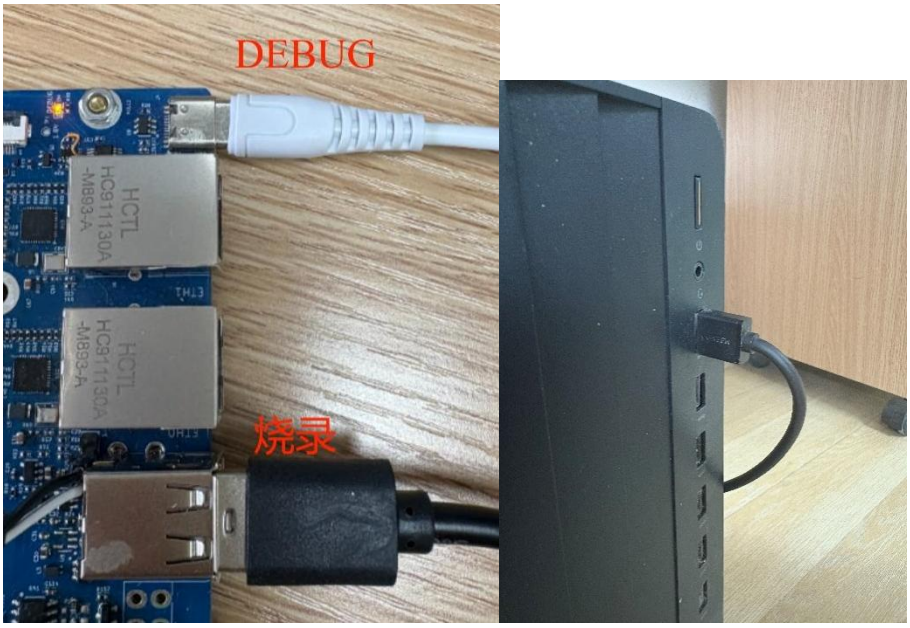
For ECB33, select ebyte_t536_uart0_linux-5.10-origin.img



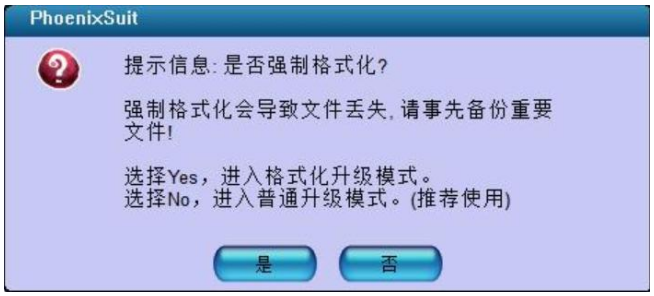
Next, we connect two type-a and type-c data cables. One is used to observe the burning log, and the other is used to burn the flash on the board. After the connection is completed, we need to enter the burning mode. The type-a evaluation board must be connected to the upper port on this side, and only then is the OTG on the upper side.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



The ECB33 has introduced the FEL button. Users can press FEL and hold it, then press the reset button or power on the board to enter the burning mode, and a burning inquiry box will pop up.



Users can choose whether to format and upgrade as needed and wait for the burning process to complete.

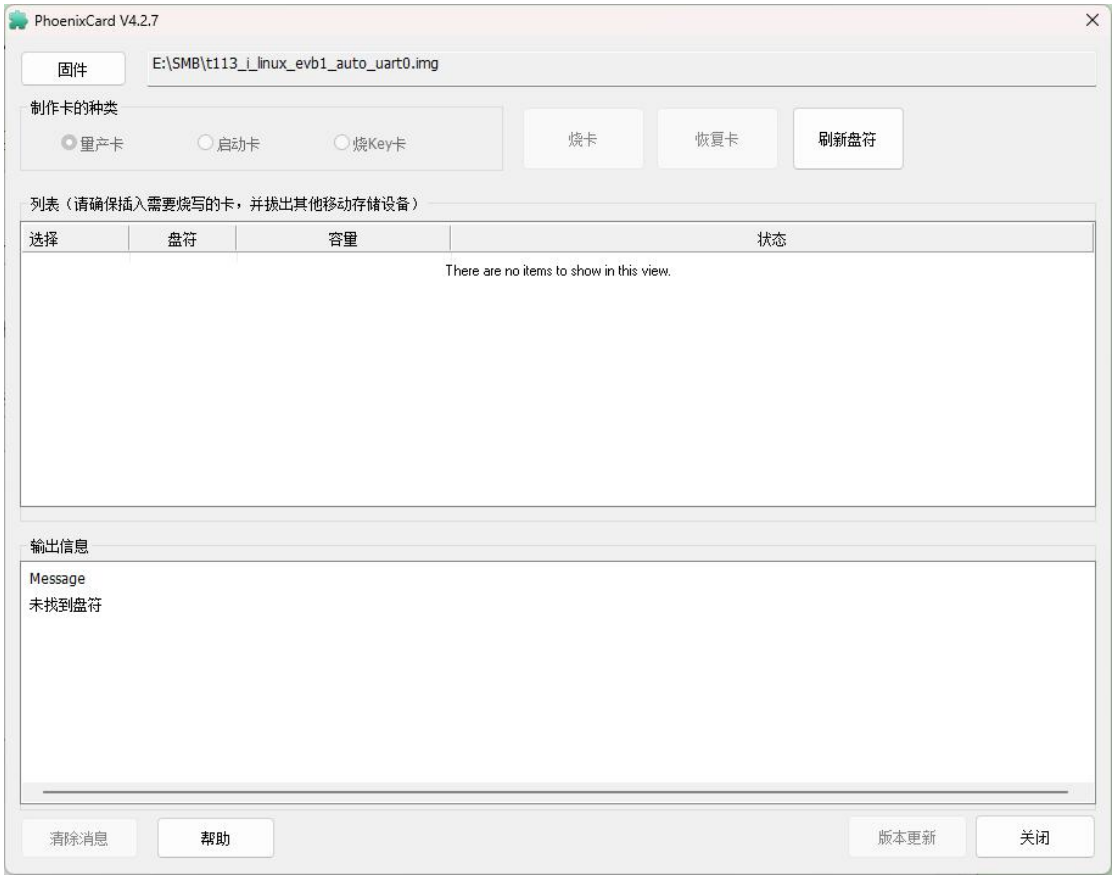
It should be noted that the T536 starts from the SD card first. After burning to the flash on the board, the sd card must be removed before it can boot from the flash on the board.

5.2. Make an SD card boot card

First open the PhoenixCard program, the interface is as follows:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

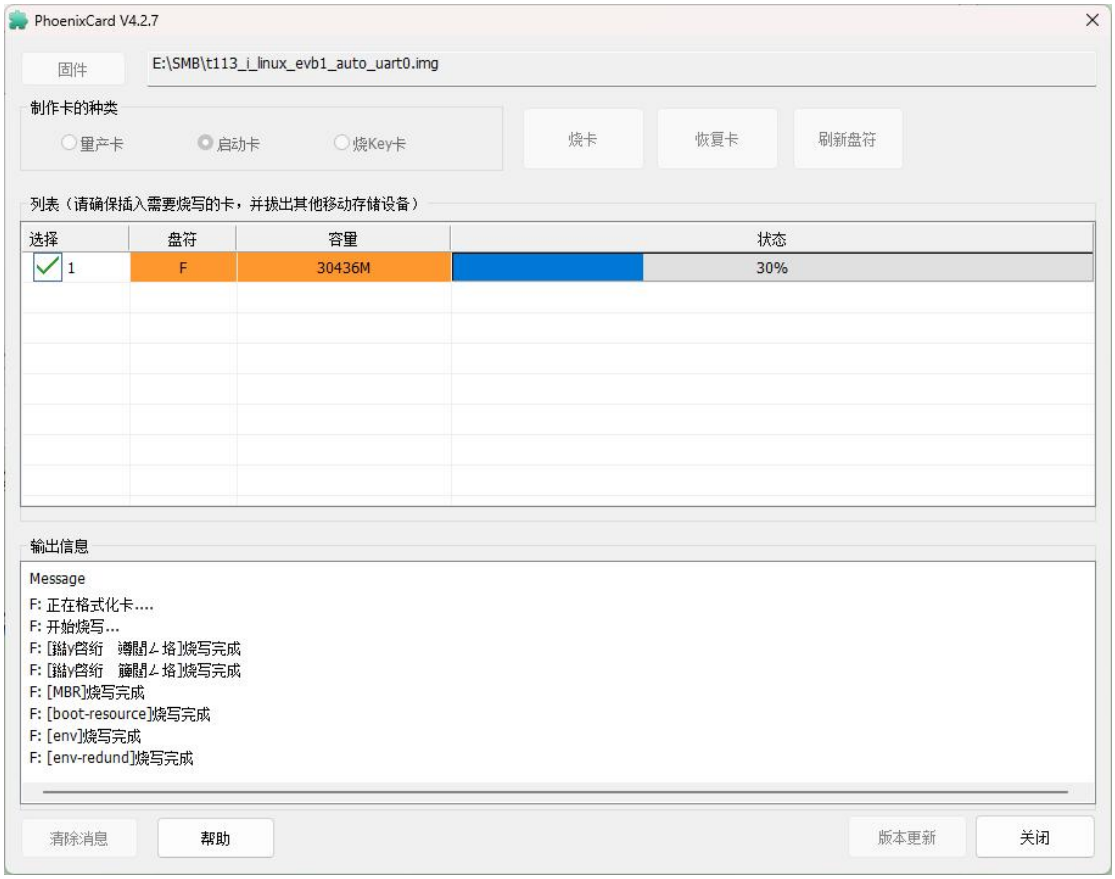
Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



The user should select the firmware to be burned, select the card type as "Startup Card", and then select the SD card device to be burned. The software will automatically scan the available mobile storage devices. Please be careful not to select the wrong device. Finally, click the "Burn Card" button, and the current progress will be displayed in real time during the burning process:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

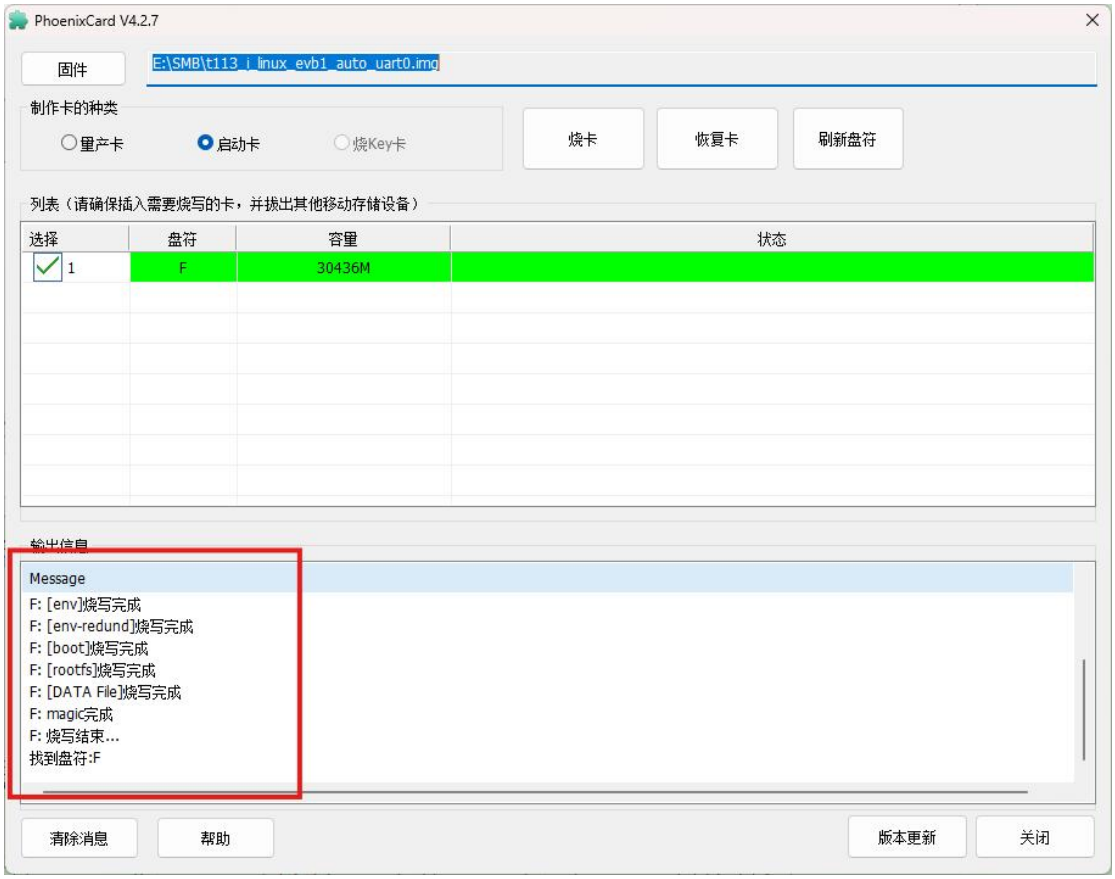
Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



After the flashing is completed, the interface is as follows. You can check whether the startup card is successfully made through the output information at the bottom of the software interface:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



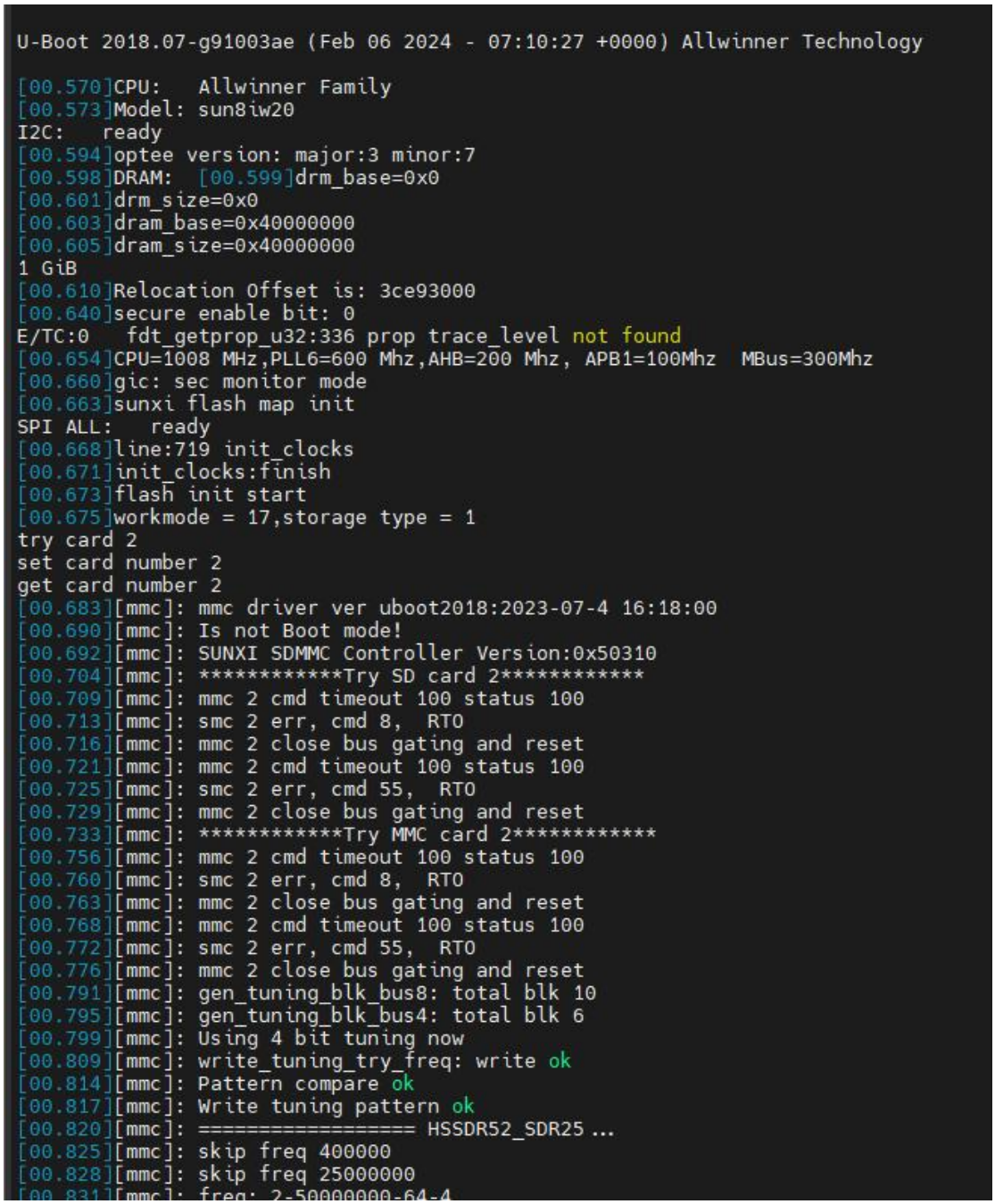
5.3. Mass production card burning

When making an SD card startup card, we will also find that there is a type of mass-produced card. This method can be used to make an SD card for pipeline operation of the burning function. Users do not need to use a separate image firmware, but can directly use the image that can be used correctly in the startup card stage.

After the production is completed, start the system from the mass production card, connect the debug serial port and we can view all the printed information of the burning process:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



After the system boots into the u-boot stage, the system begins to be burned.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
[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
```

After the above-mentioned content appears, the user can disconnect the power supply of the development board, unplug the mass production card, and then power it on again. The serial terminal will print the following startup information to confirm that it is started from the eMMC:

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
[33]HELLO! BOOT0 is starting!
[35]BOOT0 commit : 069ed30b88
[38]set pll start
[44]periph0 has been enabled
[47]set pll end
[49][pmu]: bus read error
[51]board init ok
[53]enable_jtag
[55]get_pmu_exist() = -1
[57]DRAM BOOT DRIVE INFO: V0.34
[60]DRAM CLK = 792 MHz
[62]DRAM Type = 3 (2:DDR2,3:DDR3)
[65]DRAMC ZQ value: 0x7b7bfb
[68]DRAM ODT value: 0x42.
[71]ddr_efuse_type: 0x0
[74]DRAM SIZE = 1024 MB
[81]DRAM simple test OK.
[83]rtc standby flag is 0x0, super standby flag is 0x0
[88]dram size =1024
[91]card no is 2
[92]sdcard 2 line count 4
[95][mmc]: mmc driver ver 2021-05-21 14:47
[104][mmc]: Wrong media type 0x0, but host sdc2, try mmc first
[110][mmc]: **Try MMC card 2**
[133][mmc]: RMCA OK!
[135][mmc]: mmc 2 bias 0
[138][mmc]: MMC 5.1
[140][mmc]: HSSDR52/SDR25 4 bit
[143][mmc]: 50000000 Hz
[145][mmc]: 7396 MB
[147][mmc]: **SD/MMC 2 init OK!!!**
[249]Loading boot-pkg Succeed(index=0).
[253]Entry_name        = u-boot
[260]Entry_name        = optee
[264]Entry_name        = dtb
[267]tunning data addr:0x430003e8
[270]Jump to second Boot.
M/TC: OP-TEE version: 2a99a16f (gcc version 5.3.1 20160412 (Linaro GCC 5.3-2016.05)) #1 Thu Aug 17 11:13:02 UTC 2023 arm
E/TC:0 0 platform_standby_fdt_parse:126 no pmu0 node
E/TC:0 0 sunxi_twi_parse_from_dt:121 no pmu node
```

For NAND boot, the following will be displayed:

```
[29]HELLO! BOOT0 is starting!T
[32]BOOT0 commit : 069ed30b88
[35]set pll start
[40]periph0 has been enabled
[43]set pll end
[45][pmu]: bus read error
[47]board init ok
[49]enable_jtag
[51]get_pmu_exist() = -1
[53]DRAM BOOT DRIVE INFO: V0.34
[56]DRAM CLK = 792 MHz
[58]DRAM Type = 3 (2:DDR2,3:DDR3)
[61]DRAMC ZQ value: 0x7b7bfb
[64]DRAM ODT value: 0x42.
[67]ddr_efuse_type: 0x0
[70]DRAM SIZE = 256 MB
[77]DRAM simple test OK.
[79]rtc standby flag is 0x0, super standby flag is 0x0
[84]dram size =256
[87]spinand UBOOT_START_BLK_NUM 8 UBOOT_LAST_BLK_NUM 32
[92]block from 8 to 32
[234]check is correct. Find a good uboot copy at block 8
[239]dma 0x2b3c4 int is not used yet
[242]dma 0x2b3c4 int is free, you do not need to free it again
[248]Entry_name        = u-boot
[255]Entry_name        = optee
[259]Entry_name        = dtb
[262]Jump to second Boot.
M/TC: OP-TEE version: 2a99a16f (gcc version 5.3.1 20160412 (Linaro GCC 5.3-2016.05)) #1 Thu Aug 17 11:13:02 UTC 2023 arm
E/TC:0 0 platform_standby_fdt_parse:126 no pmu0 node
E/TC:0 0 sunxi_twi_parse_from_dt:121 no pmu node
```

5.4. Possible problems

5.4.1. Can't find storage device when making startup card

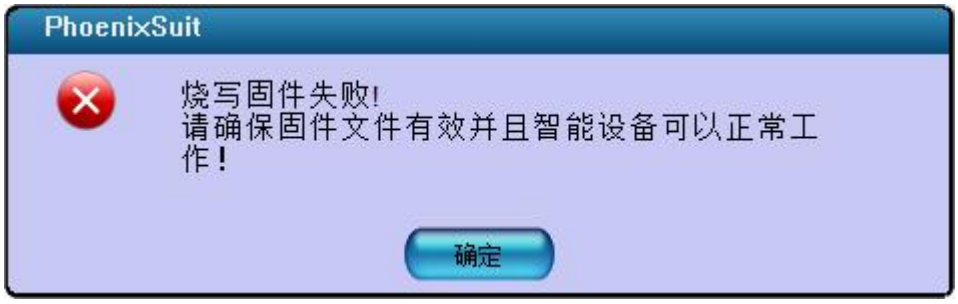
When using the tool to burn the startup card, the user may find that the tool cannot find the corresponding storage device after connecting the SD card. Please confirm whether the SD card is occupied by VMware. If it is occupied, please release the occupation and return the usage right to the host. If it is still unavailable, please try to change the USB interface or check whether it is occupied by other programs.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4.2. Failed to flash the firmware

When making a NAND board image and burning it, you may encounter a burning failure. For example, the burning tool will display the following message:



At this time, please open the serial port tool, connect the debug serial port, and check the error reported during the burning process. Take the following situation as an example:

```
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
```

Check the contents in the red box and you will find that the ubi volume size exceeds the actual device storage size. In connection with the above content about image packaging and development board adaptation, we can know that the ubi volume size is actually determined by the configuration in the file `SDK/device/product/configs/<board>/[buildroot]/sys_partition.fex`. At this time, we need to modify the size of one of the partitions to ensure that it does not exceed the physical storage size limit. The following is an example to modify the largest rootfs partition size

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

and reduce the ubi volume size:

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

At the same time, we also need to pay attention that the size cannot be smaller than the actual rootfs-ubifs.fex file size.

5.5. Existing system updates

In the previous chapters "Official Tool Burning" and "Making SD Card Boot Card", we have successfully solidified the Linux file system, and the system has been successfully started. But sometimes during the development process, we try to compile and make uboot, kernel and file system by ourselves, and later we only want to update one of uboot, device tree and file system, especially during the development process, we may need to update frequently. In order to save development time, we can use a separate update method to update the part we need to update, instead of spending more time to re-make an SD system boot card or use scripts to solidify the system to eMMC or NAND.

For the method of transferring files between the development host and the development board, please refer to this document "Data Transfer"chapter.

5.5.1. Related Documents

If you want to flash the system update, you will inevitably need to know the path of the relevant files. Here is a description of where the flash-related files are and what their names are:

Burning files

document	file name	True Path
Uboot Image	boot_package.fex	SDK/out/t536/<board>/pack_out/boot_package.fex
Kernel image	boot.fex	SDK/out/ t536/<board>/buildroot/boot.img
Root file system	rootfs.fex	SDK/out/ t536/ebyte_t536/buildroot/rootfs.ext4

5.5.2. Burning location

After understanding the relevant files, we also need to understand where these files should be burned:

Burning location

document	Programming location (NAND)	Burning location (eMMC)

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Uboot Image	/dev/mtd1	/dev/mmcblkX(seek 32800)
Kernel image	/dev/mtd8	/dev/mmcblk0p4
Root file system	/dev/mtd9	/dev/mmcblk0p5

For the kernel image and root file system, you also need to refer to the partition configuration in the `SDK/device/config/chips/t113_i/configs/<board>/[buildroot]/sys_partition.fex` file to determine.

5.5.3. eMMC + SD card

If the user is using an eMMC core board and wants to update the firmware in an existing system, we need to boot from eMMC or from an SD card to replace our own firmware.

5.5.3.1. U-boot

The uboot compiled by the SDK of T113-i is a packaged file boot_package.fex, which should be burned completely during the burning process.

1. Copy the compiled boot_package.fex file to the current directory of the eMMC/SD card system through nfs/tftp/U disk and other methods.

2. Find the path of the device to be flashed, and determine whether the device is eMMC or SD card according to the description of `timing spec` in the `ios` node. In the following text, `mmc0` is used to represent the eMMC device for demonstration. Users should determine it according to the actual situation.

```
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. Write the u-boot image to the mmcblk0 device

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

4. [Optional] Write the u-boot image to the backup partition

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

5. Aftermath and Restart

```
sync && reboot
```

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.5.3.2. Kernel image

The kernel compiled by the SDK of T536 has packaged the kernel and device tree into a whole according to the Android image, so you can directly burn the boot.img when burning.

1. Copy the compiled kernel and device tree files to the current directory of the eMMC/SD card system through nfs/tftp/U disk and other methods.

2. Find the path of the device to be flashed, and determine whether the device is eMMC or SD card according to the description of `timing spec` in the `ios` node. In the following text, `mmc0` is used to represent the eMMC device for demonstration. Users should determine it according to the actual situation.

```
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. Copy the kernel and device tree to the boot partition.

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

4. Aftermath and Restart

```
sync && reboot
```

5.5.3.3. Root file system

Generally speaking, onlyUse SD card boot to update the eMMC file system, or boot through eMMC to update the SD card. A running system cannot format itself!

1. Conversionrootfs.fex to a compressed package [in ububtu environment]

```
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 *
```

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2. Copy the converted root file system (rootfs.tar.gz) to the eMMC/SD card system via nfs/tftp/U disk.

3. Mountmmcblk0p5Partition

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

4. Clearmmcblk0p5Partition old data

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

5. Copy the new root file system tommcblk0p5Partition

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

6. Unmount the partition

```
umount dest_dir/
```

7. Restart the system

```
reboot
```

When copying rootfs.tar.gz, there may be insufficient available space. You can try to copy it to the `/mnt/UDISK` directory. If the user does not have this directory, please mount the /dev/mmcblk0p8` partition to any directory and copy rootfs to this directory.

5.5.4. NAND

If the user is using a NAND core board and wants to update the firmware in an existing system, we need to boot from the NAND itself.

5.5.4.1. U-boot

The uboot compiled by the SDK of T113-i is a packaged file boot_package.fex, which should be burned in its entirety during the burning process

1. Copy the compiled boot_package.fex file to the current directory of the system through nfs/tftp/U disk or other methods.

2. Erase old u-boot

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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

3. Write the u-boot image to the mtd1 device

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

4. Aftermath and Restart

```
sync && reboot
```

5.5.4.2. Kernel image

The kernel compiled by the SDK of T113-i has packaged the kernel and device tree into a whole according to the Android image, so you can directly burn the boot.img when burning.

1. Copy the compiled kernel and device tree files to the current directory of the system through nfs/tftp/U disk

2. Erase the old kernel image

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

3. Copy the kernel and device tree to the boot partition.

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

4. Aftermath and Restart

```
sync && reboot
```

6. References

- T536_Series_Datasheet_V2.0.pdf
- T536_Series_User_Manual_V2.0.pdf
- Linux kernel open source community:https://www.kernel.org/

7. Revision Notes

Revision Notes Table

Version	Modifications	Modification time	prepared by	Proofreading	Approval
V1.0	First Draft	2025-06-26	WYQ	WFX	WFX

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

8. About us

Technical support: service@cdebyte.com

Documents and RF Setting download link: www.cdebyte.com

Thank you for using Ebyte products! Please contact us with any questions or suggestions: info@cdebyte.com

---

Web: www.cdebyte.com

Address: B5 Mould Park, 199# Xiqu Ave, High-tech District, Sichuan, China

Chengdu Ebyte Electronic Technology Co.,Ltd.

Copyright©2012-2025, Chengdu Ebyte Electronic Technology Co., Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000