Software Development Guide
ECK10-13xA core board



Chengdu Ebyte Electronic Technology Co.,Ltd.

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

Contents

Disclaimer 2
1. Overview 3
1.1. Software Resources 3
2. How to burn the system image 3
2.1. Use STM32CubeProg to program 3
2.2. Create an SD card launcher 5
3. Development environment preparation 6
3.1. Vmware shared directory 7
3.2. NFS Service 8
3.3. SSH service 9
3.4. TFTP Service 9
3.5. Windows installation software development tools 10
3.6. Ubuntu configuration development environment 14
4. How to build an image 18
4.1. TF-A 18
4.2. Optee-os 21
4.3. U-boot 23
4.4. Fiptool encapsulates OP-TEE and U-Boot firmware 29
4.5. Kernel 31
4.6. summary 33
5. How to adapt to different hardware platforms 33
5.1. Adding hardware resources to the device tree 34
5.2. Adding device tree 35
5.3. How to configure CPU function pins according to hardware 36
6. Making a file system 41
6.1. Buildroot creates a minimal system 41
6.2. Use ubuntu -base to create a system 44
7. Create an image file that can be burned to N and Flash 48
8. References 52
9. Revision Notes 52
10. About Us 53

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

Disclaimer

The information in this document is subject to change without notice. The document is provided "as is" without any warranty, 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 rights arising from the use of the information in this document. No license to any intellectual property, whether express or implied, by estoppel or otherwise, 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.

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 Bootloader and Kernel, customize Rootfs suitable for your own application needs, etc. We first introduce how to build a system image suitable for the ECB10-TB13X development board based on the source code we provide, and how to burn the built image to the development board. For those users who develop projects based on the ECK10-135A5M1G-I 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, users can quickly customize the system image suitable for their own hardware.

1.1. Software Resources

ECB10-TB13X is equipped with an operating system based on the Linux 6.1.28 kernel. The development board comes with a cross-compilation tool chain required for embedded Linux system development, TF-A source code, Optee-os source code, U-boot source code, Linux kernel and driver module source code, as well as various development and debugging tools for Windows desktop environment and Linux desktop environment, application development samples, etc.

2. How to burn the system image

Use stm32cubeprogrammer to burn into the NandFlash memory on the ECB10 board. Or you can make an SD card bootloader to boot the ECB10 from the SD card.

2.1. Use STM32CubeProg to program

2.1.1. Burning tools and materials

CH340 serial cable and serial software

USB Type-C data cable

Power adapter 12V

STM32CubeProgrammer software

Software image package

The installation of the serial port software mobaexterm and STM32CubeProgrammer is explained in chapter 3.5.

2.1.2. Setting up the hardware and programmer

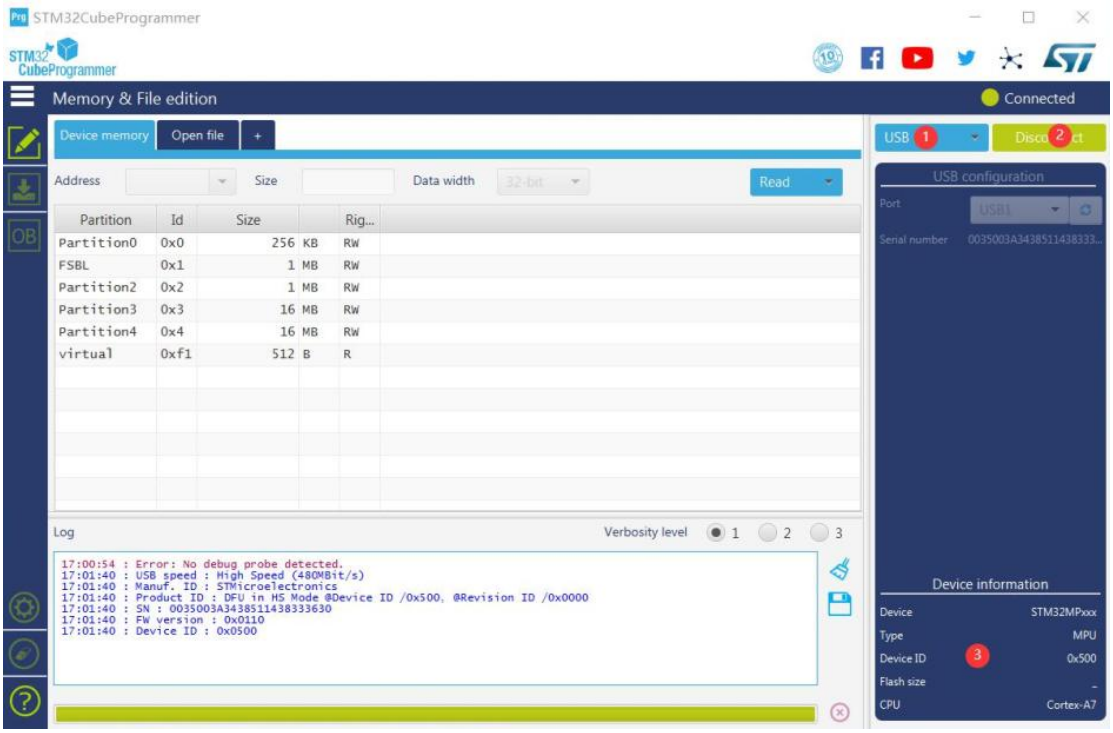
After having the above software, we connect the RX , TX, GND of the CH340 serial port cable to the TX , RX,

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

GND on the board , and then connect to the type-c interface. Note that the virtual machine should not occupy the USB port. Finally, we connect the power cord.

Set the DIP switch to Download mode (B2/B1/B0: 0 0 0).

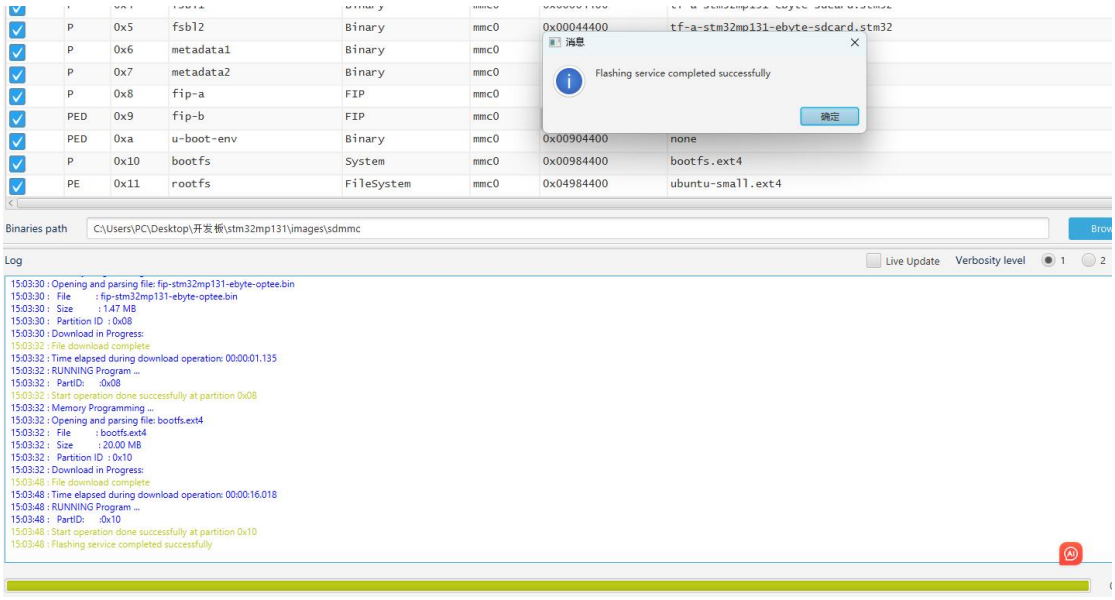
Open the STM32CubeProgrammer software that has been installed on the Windows platform. If the software is not installed, refer to Section 3.5 . Select USB burning mode. Click the connect button to connect. Check whether the information of the development board can be displayed normally, as shown in the figure below. If the log display window in the lower left corner prompts some information about the device, it means that the E CB10 has been connected normally .



2.1.3. Select Profile

Select and open 02_Images/flashlayout-nandflash.tsv and click download to burn.

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



After burning is completed.

Set the DIP switch to NandFlash mode (B2/B1/B0: 0 1 1). Press the reset button to start the image. You can use the serial port software to observe the startup process.

2.2. Create an SD card launcher

2.2.1. Prepare the Materials

SD Card

Image creation tool (04_Tools /Win32DiskImager-1.0.0-binary/ Win32DiskImager.exe)

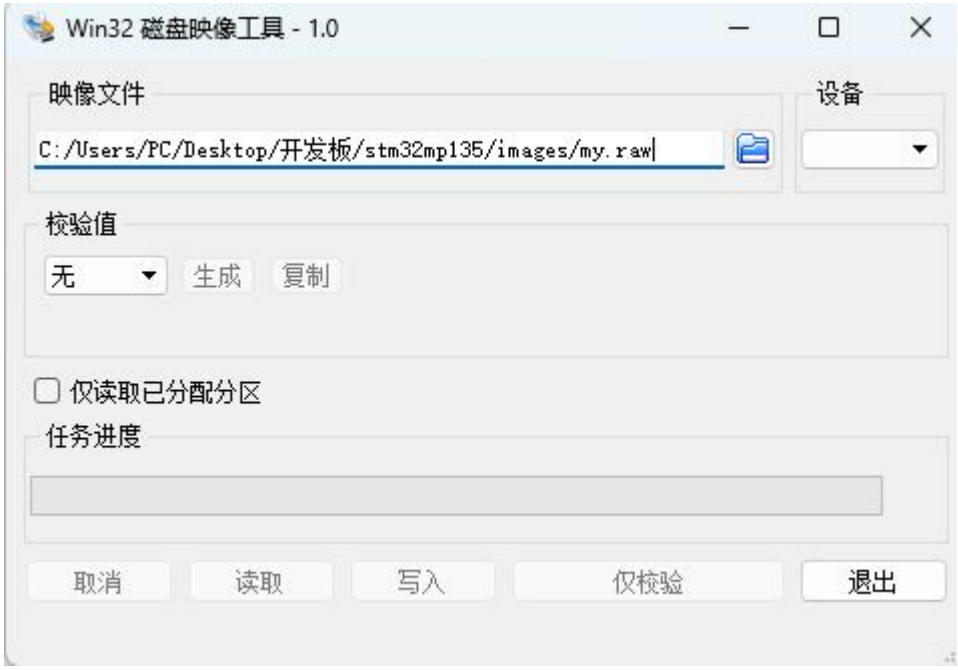
2.2.2. Make SD card bootable

Format SD card

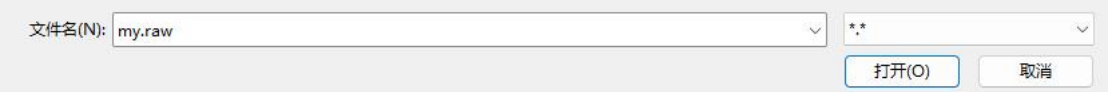
Open Win32DiskImager-1.0.0-binary

Select the image file flashlayout-sdcard.raw

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



The file type must be selected as * .* to find the burning file



Click Write to start burning and wait for writing to complete. It will take about 1 minute to complete. The speed depends on the read and write speed of the SD card.

After burning is complete, insert the SD card .

Set the DIP switch to SD Card mode (B2/B1/B0: 1 0 1). Press the reset button to start the image. You can use the serial port software to observe the startup process.

3. Development environment preparation

This section will introduce how to build a development environment suitable for the STM32MP1XX series processor platform. By reading this chapter, you will understand the installation and use of related hardware tools and software development and debugging tools. You can also quickly build a related development environment to prepare for subsequent development and debugging.

Host hardware: i7-12700 and VMware virtual machine, for reference.

Virtual machine operating system: ubuntu18.04 64bit . In the image provided by Ebyte, the path is: 0 4 _ Tools →ubuntu-18.04.5-desktop-amd64.iso.

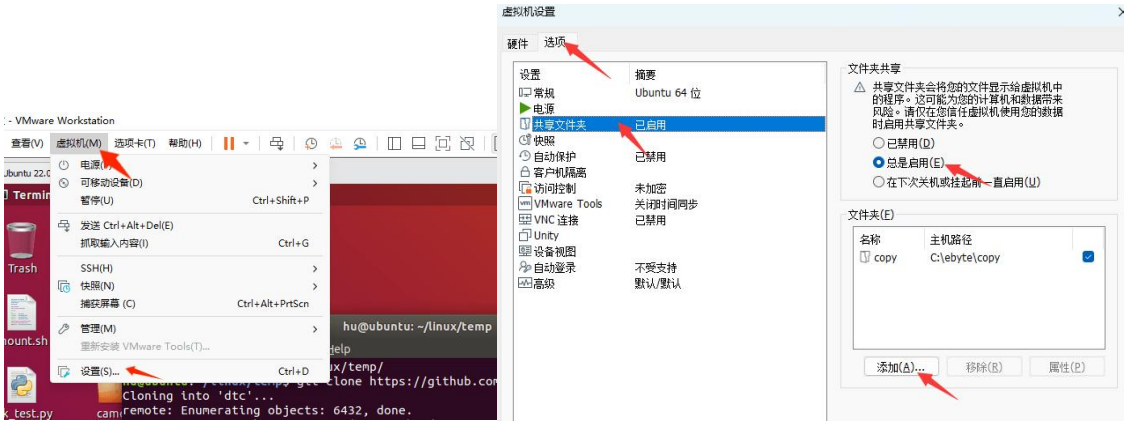
Users can use vmware and this image to build an Ubuntu virtual machine according to some online tutorials, but this article does not introduce the relevant content. After installing the virtual machine, you can use the following configuration content to configure Ubuntu.

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

The following section will explain how to install some tools for transferring files. If you do not need the following tools, you can directly jump to section 2.6 to configure Ubuntu to compile the SDK . If you need to install some file transfer tools, refer to the following section. The Ubuntu virtual machine and Windows can transfer files using the VMware shared directory method, refer to section 2.2 . The Ubuntu virtual machine and the development board can transfer files using the NFS shared directory, refer to section 2.3 . To use the serial port tool to remotely log in to Ubuntu, you need to enable the SSH service, refer to section 2.4 . The following content only verifies Ubuntu - 1804.

3.1. Vmware shared directory

By setting up a shared directory in VMware, the virtual machine and Windows can access the same folder, making it easier to transfer files.



Then add a Windows path here. Here I choose C :ebyte/copy , and the default name is copy. Next, we will verify whether this folder can be accessed in Ubuntu.

```
# cd /mnt/hgfs/copy/

# ls
```



This way you can easily transfer files between Windows and Ubuntu virtual machines. It is best not to compile in this folder, copy it to another location in the virtual machine and then compile. If you cannot find this folder after the virtual machine restarts, you need to remount it manually.

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

```
# sudo mount -t fuse.vmhgfs-fuse .host:/ /mnt/hgfs -o allow_other -o nonempty
```

3.2. NFS Service

If you need to boot the development board from the network, you must first install and enable the NFS service in Ubuntu. Use the following command to install the NFS service:

```
# sudo apt-get install nfs-kernel-server rpcbind
```

Wait for the installation to complete. After the installation is complete, create a folder named "linux" in the user's root directory, and create a new folder named "nfs" in the "linux" folder.

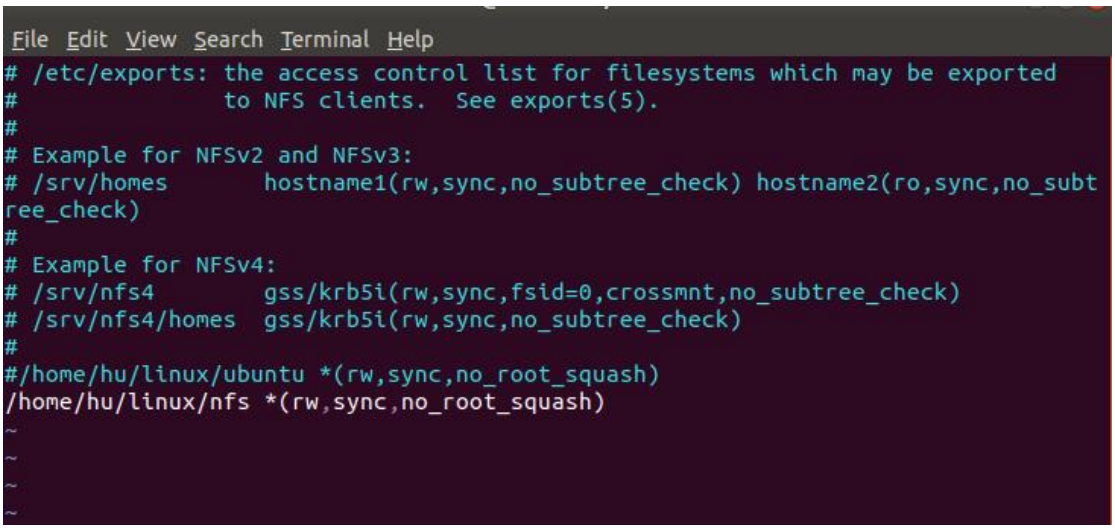
```
# sudo mkdir -p /home/hu/linux/nfs
```

The created nfs folder is used by the nfs server. Later, we can access the nfs folder through the network file system on the development board. To configure nfs first, use the following command to open the nfs configuration file /etc/exports:

```
# sudo vi /etc/exports
```

Open /etc/exports and add the following content at the end:

```
/home/hu/linux/nfs *(rw,sync,no_root_squash)
```



Restart the NFS service using the following command:

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

```
# sudo /etc/init.d/nfs-kernel-server restart
```

3.3. SSH service

we can use terminal software to log in to Ubuntu under Windwos, such as using Moba X term. Use the following command to install the SSH service under Ubuntu:

```
# sudo apt-get install openssh-server
```

The above command installs the ssh service. The ssh configuration file is /etc/ssh/sshd_config. Use the default configuration. Users can use SSH to remotely log in to the virtual machine. It is necessary to ensure that the host can ping the development board.

3.4. TFTP Service

By enabling Ubuntu's TFTP service, we can start the development board from the network. In essence, the development board accesses the file path on the virtual machine through the network. After obtaining the kernel file, it can be started.

```
# sudo apt-get install vsftpd
```

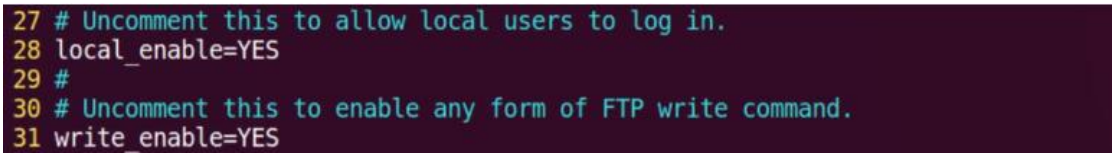
Wait for the software to be automatically installed. After the installation is complete, use the following VI command to open /etc/vsftpd.conf:

```
# sudo vi /etc/vsftpd.conf
```

Open the vstfpd.conf file and find the following two lines:

```
local_enable=YES
write_enable=YES
```

Make sure there is no "#" in front of the two lines above. If there is, cancel it. When completed, it will look like the following figure:



After modifying vsftpd.conf, save and exit, and restart the FTP service using the following command:

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

```
# sudo /etc/init.d/vsftpd restart
```

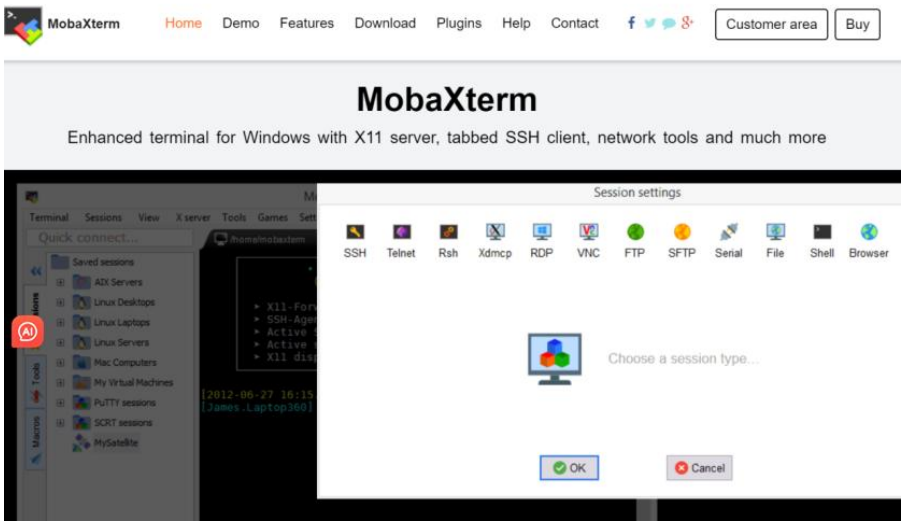
3.5. Windows installation software development tools

3.5.1. Moba X term

This software is used for communication with the debug serial port of the development board. Users can use the debug serial port to interact with the development board through Linux commands.

3.5.1.1. 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 at https://mobaxterm.mobatek.net/, as shown in the figure:

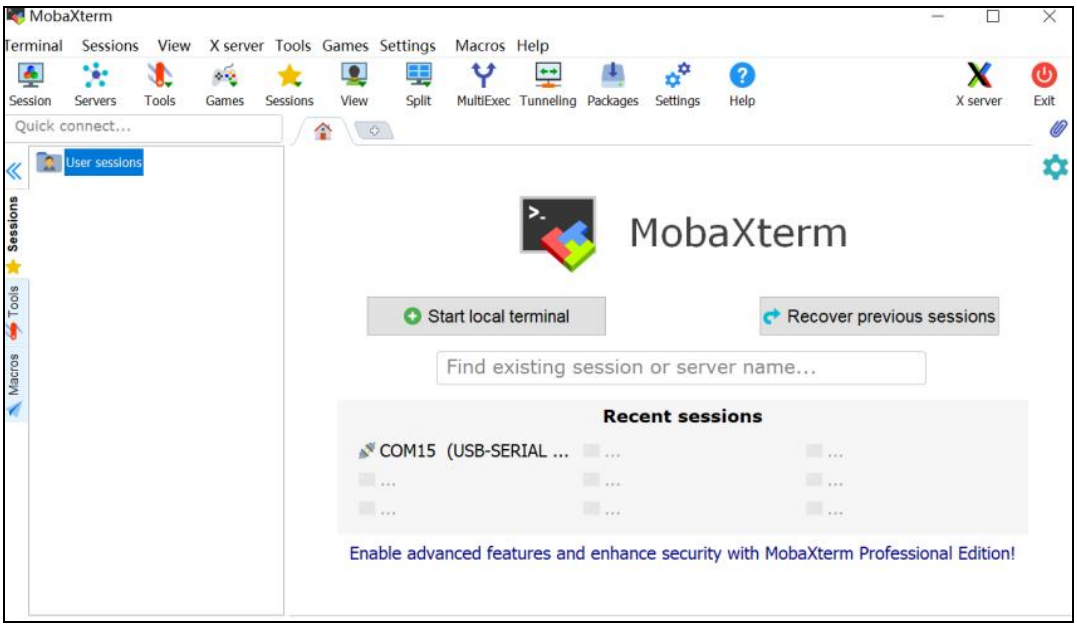


The installation package has been placed in the development board, the path is: 04_Tools -> MobaXterm_ Portable.zip . Open this compressed package.

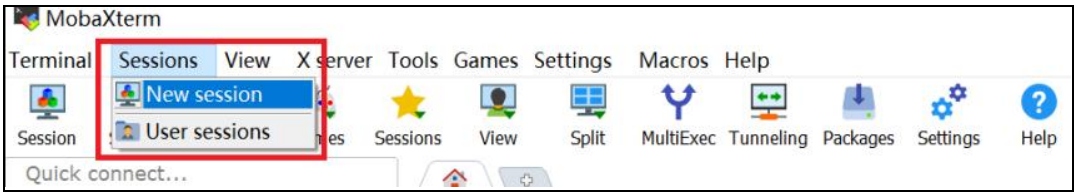
3.5.1.2. Software Usage

Double-click the MobXterm icon to open the software. The software interface is as shown below:

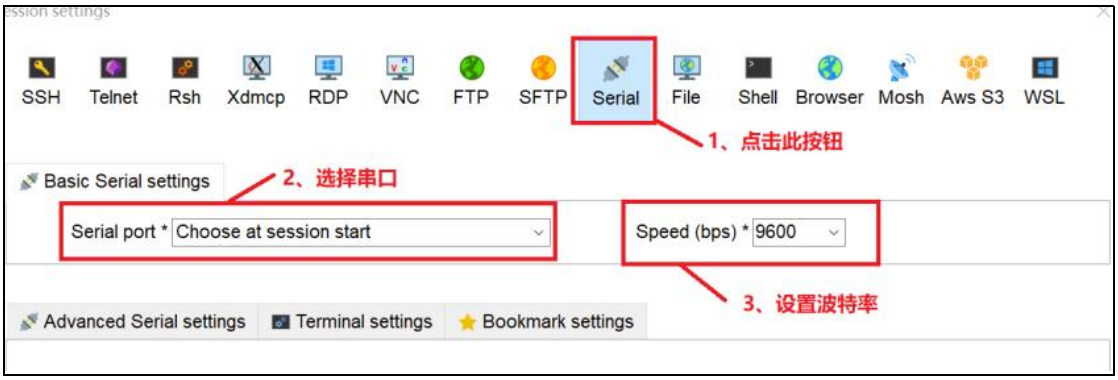
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



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, and then set the baud rate to 115200 (set according to your actual needs).

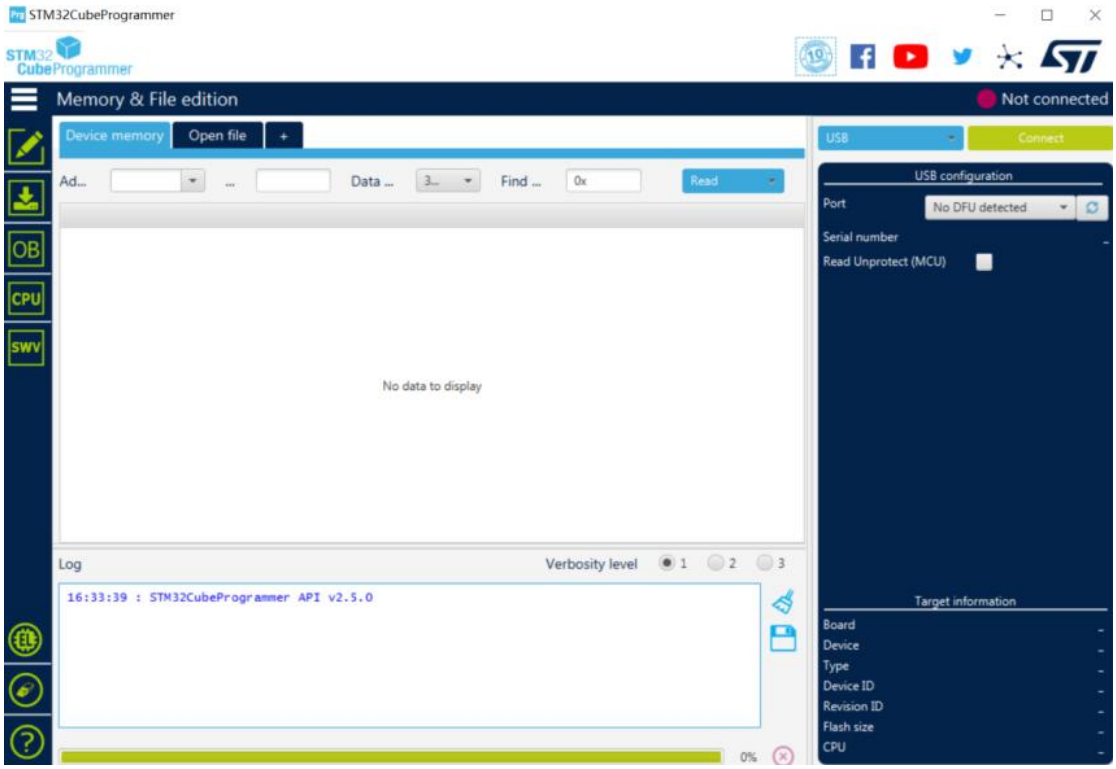
3.5.2. STM32CubeProg

ST introduced the highly integrated programming tool STM32CubeProgrammer, which can partition unpartitioned storage devices during the burning process. Once partitioned, the compiled binary file can be used to update a partition

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

separately. Users can choose to use the appropriate version according to their needs. The download address is as follows: https://www.st.com/zh/development-tools/stm32cubeprog.html. The path is 04_Tools→en.stm32cubeprog_v2-5-0.zip . This software is mainly used to burn our development board using USB .

Unzip the STM32CubeProgrammer installation package in the development board, and then double-click the unzipped "SetupSTM32CubeProgrammer-2.5.0.exe". The installation process is very simple. Just follow the prompts to install it. Double-click the icon to open STM32CubeProgrammer , as shown in the figure



1.1.1. J AVA Environment

Before installing STM32CbeMX and STM32CubeIDE, we must first install the Java environment. The Java runtime environment version must be V1.7 or above, otherwise the above two applications will not be usable. Find and download the latest 64-bit Java software at https://www.java.com/zh-CN/download/manual.jsp . The path is 04_Tools→jre-8u271-windows-x64.exe . STM32CbeMX can generate some device tree files. If you do not need to modify them, you can skip them.

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



After installing the Java runtime environment, in order to check whether it is installed normally, we can open the Windows cmd command input box and enter the following command:

java -version //Command to query the Java version

If the installation is successful, the Java version number will be printed out, as shown in the figure:

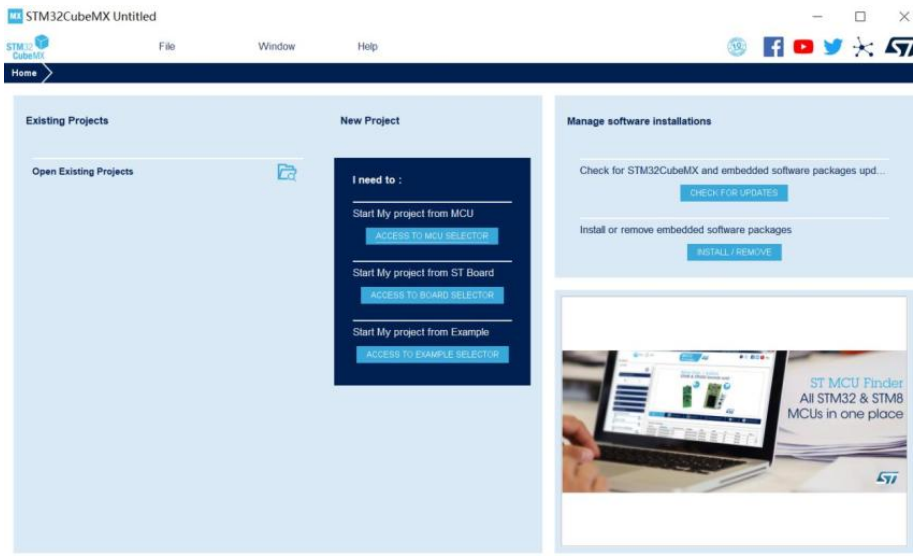
```
C:\Users\PC>java -version
java version "1.8.0_271"
Java(TM) SE Runtime Environment (build 1.8.0_271-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.271-b09, mixed mode)
```

3.5.3. STM32CubeMX

STM32CubeMX is a code generation software launched by ST specifically for generating HAL code for STM32. It uses a visual interface to configure various resources such as STM32 clock, timer, DMA, serial port, GPIO, etc. At present, ST has added the STM32MPU series CPU to STM32CubeMX. We can also use this tool to configure the device tree and clock of TF-A, Optee, U-boot and Kernel, so it can be used to generate device tree and clock when developing Cortex A7. The download address is as follows: https://www.st.com/zh/development-tools/stm32cubemx.html . The path is 04_Tools →en.stm32cubemx_v6-0-1.zip .

Double-click the CubeMX icon and it will open as shown below:

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

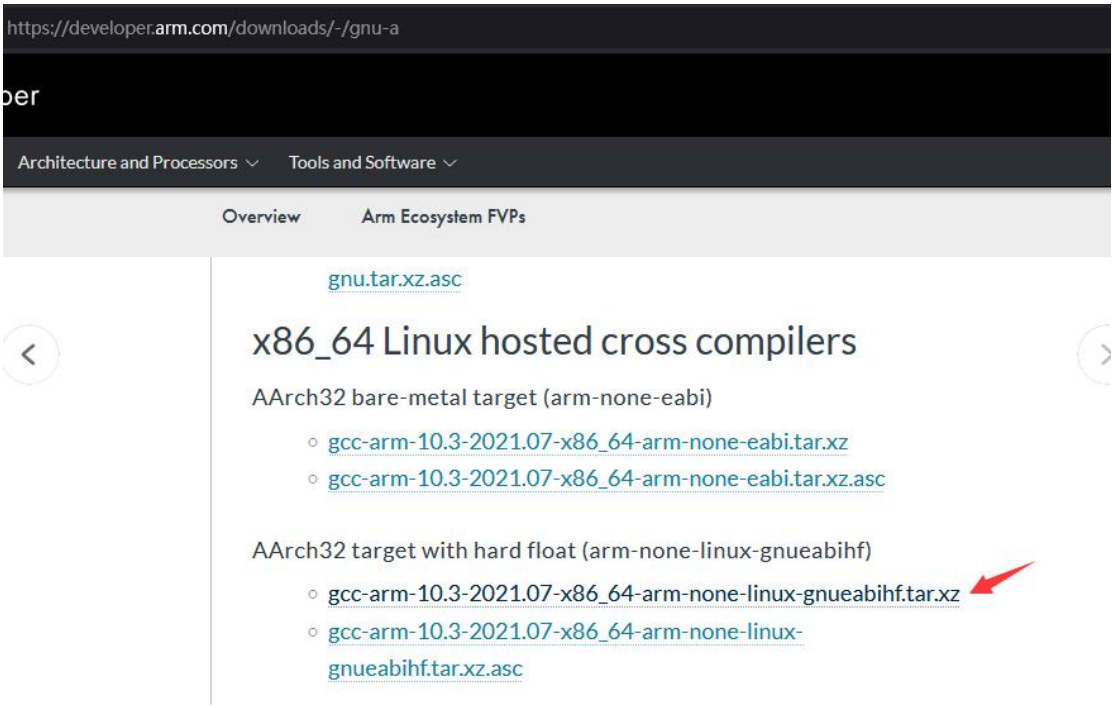


3.6. Ubuntu configuration development environment

3.6.1. Install the cross-compilation toolchain

Users can directly use this cross-compilation tool chain to compile Bootloader, Kernel or their own applications. The specific process will be introduced in detail in the following chapters. Here we first introduce the installation steps of the cross-compilation tool chain.

Use the cross compiler officially produced by ARM . The compiler download address is as follows: https://developer.arm.com/tools-and-software/open-source-software/developertools/gnu-toolchain/gnu-a/downloads , open it as shown in the figure.



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

paths provided are :

04_Tools→ gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz .

We use the vmware shared directory to copy this tool chain to the virtual machine. First put this file in the copy directory set in section 3.1 , and then open the virtual machine.

In Ubuntu, create the directory: /usr/local/arm

Place the cross compiler in the directory above and unzip it

```
# sudo mkdir /usr/local/arm
# cd /mnt/hgfs/copy
# sudo cp gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz /usr/local/arm/ -f
# cd /usr/local/arm
# sudo tar -vxf gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz
```

Wait for the decompression to be completed. After the decompression is completed, a folder named " gcc-arm-10.3-2021.07-x86_64-arm-none-linux- gnueabihf" will be generated. This folder contains our cross-compilation tool chain.

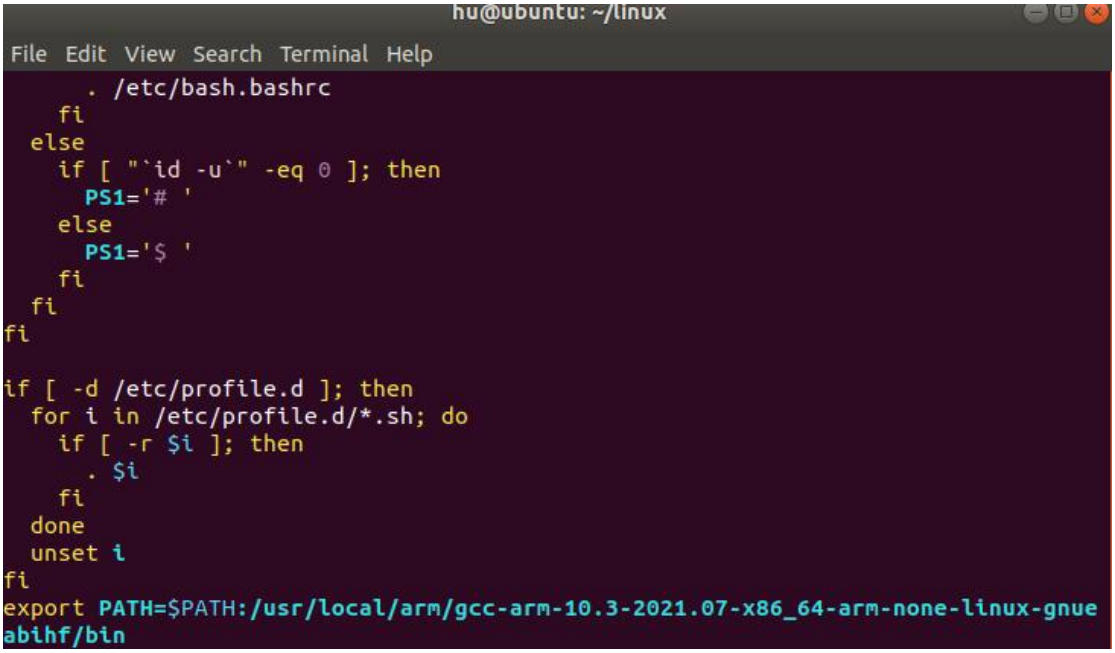
To modify the environment variables, open the /etc/profile file with the following command:

```
# sudo vi /etc/profile
```

After opening /etc/profile, enter the following at the end:

```
export PATH=$PATH:/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-
gnueabihf/bin
```

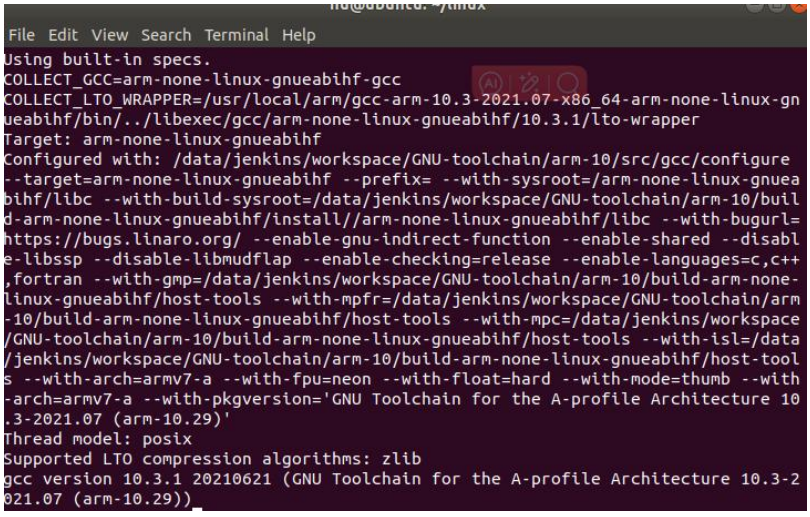

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



After modification, save and exit, restart the Ubuntu system, and the cross-compilation tool chain (compiler) will be installed successfully.

Next, we verify whether the installation is successful. The following version information indicates that the installation is successful.

```
# arm-none-linux-gnueabihf-gcc -v
```



The meaning of "arm-none-linuxgnueabihf-gcc" is as follows:

1. arm means this is a compiler for compiling arm architecture code.

2. none indicates the manufacturer. Generally, semiconductor manufacturers will modify the universal cross-compiler. Here is the name of the semiconductor manufacturer. The cross-compilation done by ARM itself is none,

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

indicating that there is no manufacturer.

3. linux means running in Linux environment.

4. gnueabihf stands for Embedded Binary Interface. The following hf is the abbreviation of hard float, which means hardware floating-point operation. It means that this cross-compilation tool chain supports hardware floating-point operation.

5. gcc means the gcc tool.

3.6.2. Install related libraries

We need to rely on many libraries when compiling, so we install them here.

```
# sudo apt-get update
# sudo apt-get install lsb-core lib32stdc++6 u-boot-tools libyaml-dev bison flex sed wget curl git-core coreutils unzip texi2html texinfo docbook-utils gawk cvs subversion python-pysqlite2 diffstat help2man make gcc build-essential g++ chrpath libxml2-utils xmlto docbook bsdmainutils iputils-ping cpio python-wand python-pycryptopp python-crypto
# sudo apt-get install libsdl1.2-dev xterm corkscrew nfs-common nfs-kernel-server mercurial u-boot-tools libarchive-zip-perl
# sudo apt-get install ncurses-dev bc linux-headers-generic gcc-multilib libncurses5-dev libncursesw5-dev lrzsz dos2unix lib32ncurses5 repo libssl-dev
```

The default dtc version of ubuntu -1804 is too low, so we manually install device -tree-compile r. Get the latest source code from github, or use the source code we have downloaded.

04_Tools→ dtc.tar.gz .

After copying to the virtual machine, unzip, compile, and add it to the system directory.

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

```
# sudo apt-get install pkg-config

# tar -xvzf dtc.tar.gz

# cd dtc

# make

# sudo cp dtc /usr/bin
```

Finally, we verify that we can see the correct dtc version.



4. How to build an image

The previous chapters have already described the complete process of burning the image to the development board. Since many pins of the ECB10-TB13X core board have the characteristics of multiple function reuse, there will always be some differences between the baseboard designed based on the ECB10-TB13X core board and the ECK10-135A5M1G-I in the actual project . These differences may be removing the display, adding more GPIO, or adding more serial ports, and it is also possible to expand some peripherals through SPI, I2C, USB, etc. This chapter describes the specific process of developing and customizing your own system from the perspective of a system developer . If the user does not have special needs, he only needs to modify the kernel part, and the other images can use the images compiled by Ebyte.

4.1. TF-A

When booting using Trusted Boot Chain (for more information about Boot Chain, please see https://wiki.st.com/stm32mpu/wiki/Boot_chain_overview), TFA is used as FSBL (First Stage Boot Loader), which is mainly used to initialize core resources such as DDR and Clock. Hardware designed based on ECB10-TB13X core board generally does not need to modify TFA. If in special circumstances, a clock needs to be modified or early control of a pin is required, you can refer to the following method for transplantation.

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

4.1.1. stm32wrapper4dbg tool installation

When compiling TF-A or Uboot, you need to use the tool stm32wrapper4dbg, otherwise the compilation will report an error. ST provides the source code of this tool. We need to compile and install this source code under Ubuntu. The download address of the source code is: https://github.com/STMicroelectronics/stm32wrapper4dbg . We have downloaded it and put it in the development board. The path is: 04_Tools →stm32wrapper4dbg-master.zip . Copy the source code compressed package to Ubuntu and then decompress it. The command is as follows:

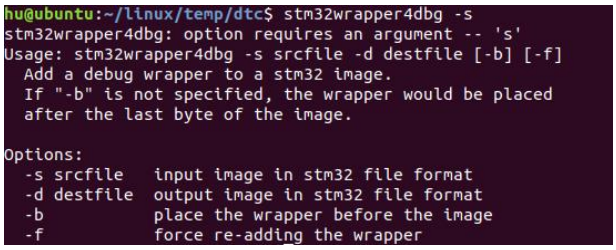
```
# unzip stm32wrapper4dbg-master.zip
# cd stm32wrapper4dbg-master
# make
```

After the compilation is complete, you will get a tool called "stm32wrapper4dbg", copy it to the /usr/bin directory of Ubuntu

```
# sudo cp stm32wrapper4dbg /usr/bin
```

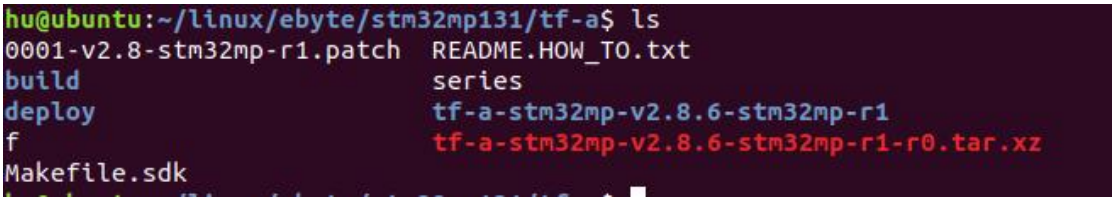
Verify that the installation was successful in the terminal

```
# stm32wrapper4dbg -s
```



4.1.2. Get the source code and compile

After obtaining the source code, decompress it. The source code path is 0 4_Source/tf-a.tar.gz . After decompression, it is as shown in the figure



The Makefile.sdk file mainly defines some compilation properties, such as the cross compiler to be used, some compilation options, etc. Makefile.sdk will eventually call the Makefile inside TF-A to compile the TF-A source code.

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

The factory source code has been modified, and the arm-none-linux-gnueabihf cross-compilation tool chain installed in Chapter 1 is specified in Makefile.sdk, and the "stm32mp13 1- ebyte , stm32mp135-ebyte " device tree file is specified to be compiled, as well as some other configurations.

```
File  Edit  View  Search  Terminal  Help
#TF_A_EXTRA_OPTFLAGS_trusted ?= AARCH32_SP=sp_min
#TF_A_BINARY_trusted ?= tf-a
#TF_A_MAKE_TARGET_trusted ?= bl32 dtbs

# Init default config settings
TF_A_DEVICETREE_emmc ?=  stm32mp131-ebyte stm32mp135-ebyte
TF_A_EXTRA_OPTFLAGS_emmc ?= STM32MP_EMMC=1 PSA_FWU_SUPPORT=1
TF_A_BINARY_emmc ?= tf-a
TF_A_MAKE_TARGET_emmc ?= all

# Init default config settings
TF_A_DEVICETREE_nand ?=  stm32mp131-ebyte stm32mp135-ebyte
TF_A_EXTRA_OPTFLAGS_nand ?= STM32MP_RAW_NAND=1 STM32MP_FORCE_MTD_START_OFFSET=0x
00200000
TF_A_BINARY_nand ?= tf-a
TF_A_MAKE_TARGET_nand ?= all

# Init default config settings
#TF_A_DEVICETREE_nor ?=  stm32mp157a-ev1 stm32mp157c-ev1 stm32mp157d-ev1 stm32mp
157f-ev1
#TF_A_EXTRA_OPTFLAGS_nor ?= STM32MP_SPI_NOR=1 PSA_FWU_SUPPORT=1 STM32MP_FORCE_MT
D_START_OFFSET=0x00080000
#TF_A_BINARY_nor ?= tf-a
```

After opening the source code root directory, execute the compilation script. If you do not have executable permissions, you need to add permissions first. Then compile. The compilation script is shown in the figure below.

```
# chmod+x ./build.sh

# chmod +x ./tools/stm32image/stm32image

# chmod +x ./tools/fwu_gen_metadata/fwumd_tool.py

# ./build.sh
```

```
#!/bin/sh
# script-name: build.sh
# version: 1.0
# date: 2024-04-08
# author: ebyte
#本脚本用作编译TF-A
#

#删除上层目录的编译文件
rm ../build/ ../deploy/ -rf

#运行编译
make -f ../Makefile.sdk all
~
```

After the compilation is completed, two directories "build" and "deploy" will be generated in the upper directory,

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

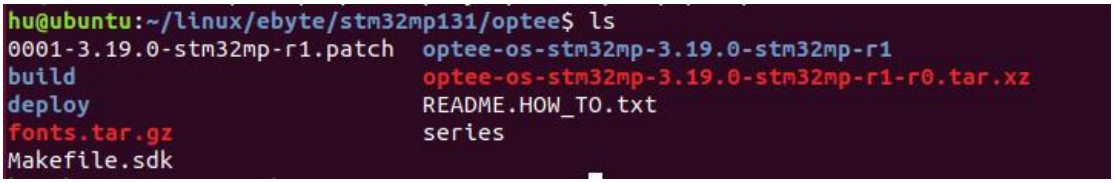
that is, the tf-a directory, as shown in the figure below. The compilation file will be generated in the deploy directory of the upper directory

```
├── fwconfig
│   ├── stm32mp131-ebyte-fw-config-optee.dtb
│   └── stm32mp135-ebyte-fw-config-optee.dtb
├── metadata.bin
├── tf-a-stm32mp131-ebyte-nand.stm32
├── tf-a-stm32mp131-ebyte-sdcard.stm32
├── tf-a-stm32mp131-ebyte-usb.stm32
├── tf-a-stm32mp135-ebyte-nand.stm32
├── tf-a-stm32mp135-ebyte-sdcard.stm32
└── tf-a-stm32mp135-ebyte-usb.stm32
```

4.2. Optee-os

OP-TEE is an open source trusted execution environment (TEE) that implements Arm TrustZone technology and is relevant to the security field.

code directory is in 04_Source / optee.tar.gz . After decompression, it is as shown in the figure below.



As can be seen from the above figure, similar to the TF-A source code, a Makefile.sdk file and an OP-TEE source code directory appear after decompression. The Makefile.sdk file mainly defines some compilation properties, such as the cross compiler to be used, some compilation options, etc. Makefile.sdk will eventually call the Makefile inside OP-TEE to compile the OP-TEE source code. The factory source code has been modified, and the arm-buildroot-linux-gnueabihf cross-compilation tool chain installed in Chapter 1 is specified in Makefile.sdk, and the "stm32mp135 - ebyte" device tree file is specified to be compiled, as well as some other configurations.

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

```
# Set default path
SRC_PATH ?= $(PWD)
BLD_PATH ?= $(SRC_PATH)/../build
DEPLOYDIR ?= $(SRC_PATH)/../deploy

# Set default optee-os config
CFG_EMBED_DTB_SOURCE_FILE ?= stm32mp135-ebyte
OPTEE_DRAMSIZE ?= 0x20000000
OPTEE_DRAMSIZE_EV ?= 0x40000000
OPTEE_DRAMSIZE_DK ?= 0x20000000

# Remove default variables
LDFLAGS =
CFLAGS =
CPPFLAGS =
# Define default make options
EXTRA_OEMAKE = PLATFORM=stm32mp1 CROSS_COMPILE_core=arm-none-linux-gnueabihf- CR
OSS_COMPILE_ta_arm64=arm-none-linux-gnueabihf- CFG_ARM32_core=y CROSS_COMPILE_ta
_arm32=arm-none-linux-gnueabihf- NOWERROR=1 LDFLAGS= CFG_TEE_CORE_LOG_LEVEL=2 CF
G_TEE_CORE_DEBUG=y
```

When compiling the factory source code, users do not need to modify anything. The OPTEE source code device tree path adapted to the ECK10-135A5M1G-I development board hardware resources is core/arch/arm/dts/, and the device tree files are stm32mp135-ebyte.dts and stm32mp13-pinctrl- ebyte .dtsi.

After getting the source code and decompressing it, executing the compilation script will generate an image file

```
# ./build.sh
```

```
#!/bin/sh
# script-name: build.sh
# version: 1.0
# date: 2023-04-08
# author: ebyte
#本脚本用作编译OP-TEE
#

#删除上层目录的编译文件
rm ../build/ ../deploy/ -rf/

#运行编译
make -f ../Makefile.sdk -j8 all
~
~
```

Similar to the TF-A compilation result, after the OP-TEE source code is compiled, two directories "build" and "deploy" will be generated in the upper directory, that is, the optee directory, as shown in the figure below.

```
hu@ubuntu:~/linux/ebyte/stm32mp131/optee$ ls
0001-3.19.0-stm32mp-r1.patch  optee-os-stm32mp-3.19.0-stm32mp-r1
build                         optee-os-stm32mp-3.19.0-stm32mp-r1-r0.tar.xz
deploy                        README.HOW_TO.txt
fonts.tar.gz                  series
Makefile.sdk
```

The build directory contains all the files generated by the compilation process, such as .a, .o and other format files. Most of these files do not need to be touched by users.

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

are very important. They are the binary files generated in the build directory and copied to the deploy directory by the Makefile.sdk script. It contains three files: tee-header_v2-stm32mp135-ebyte.bin , tee-pageable_v2-stm32mp135-ebyte.bin and tee-pager_v2-stm32mp135-ebyte.bin . These three files are the final products of compiling the factory OP-TEE source code.

~ /linux/ebyte/stm32mp131/optee/deploy$ ls

tee-header_v2-stm32mp135-ebyte.bin

tee-pageable_v2-stm32mp135-ebyte.bin

tee-pager_v2-stm32mp135-ebyte.bin

The factory OP-TEE source code has been compiled. The following describes the U-Boot source code compilation.

4.3. U-boot

The full name of uboot is Universal Boot Loader. uboot is an open source software that complies with the GPL protocol. uboot is a bare metal code and can be regarded as a bare metal comprehensive routine. Now uboot already supports advanced functions such as LCD screen, network, USB, etc. The official website of uboot is http://www.denx.de/wiki/U-Boot/ .

4.3.1. Get the source code and compile

Create a directory named "uboot" in the ubuntu directory, and then copy the Ebyte factory U-Boot source package 04 _ Source/uboot.tar.gz to the uboot directory. Then decompress it, and the decompression will be as shown in the figure.

```
hu@ubuntu:~/linux/ebyte/stm32mp131/uboot$ ls
0001-v2022.10-stm32mp-r1-MACHINE.patch
0002-v2022.10-stm32mp-r1-BOARD.patch
0003-v2022.10-stm32mp-r1-MISC-DRIVERS.patch
0004-v2022.10-stm32mp-r1-DEVICETREE.patch
0005-v2022.10-stm32mp-r1-CONFIG.patch
0099-Add-external-var-to-allow-build-of-new-devicetree-fi.patch
build
deploy
Makefile.sdk
README.HOW_TO.txt
series
u-boot-stm32mp-v2022.10-stm32mp-r1
u-boot-stm32mp-v2022.10-stm32mp-r1-r0.tar.xz
```

As can be seen from the figure above, similar to the previous source code, a Makefile.sdk file and U-Boot source code u-boot-stm32mp-v2022.10-stm32mp-r1 directory appear after decompression.

Here, the Makefile.sdk file mainly defines some compilation properties, such as U-Boot default source code configuration, specifying the name of the compilation device tree, some compilation options, etc., as shown in the figure

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

below. Makefile.sdk will eventually call the Makefile inside U-Boot to compile the UBoot source code.

```
# Set default path
SRC_PATH ?= $(PWD)
BLD_PATH ?= $(SRC_PATH)/../build
DEPLOYDIR ?= $(SRC_PATH)/../deploy

# Default U-Boot overall settings to null
UBOOT_CONFIG ?=
UBOOT_DEFCONFIG ?=
UBOOT_BINARY ?=
UBOOT_DEVICETREE ?=

# Init default config settings
# UBOOT_CONFIGS += trusted
# UBOOT_DEFCONFIG_trusted += stm32mp15_defconfig
# UBOOT_BINARY_stm32mp15_defconfig ?= u-boot.dtb
# UBOOT_DEVICETREE_stm32mp15_defconfig ?=  stm32mp157c-ed1 stm32mp157f-ed1 stm32
mp157a-ev1 stm32mp157c-ev1 stm32mp157d-ev1 stm32mp157f-ev1 stm32mp157a-dk1 stm32
mp157d-dk1 stm32mp157c-dk2 stm32mp157f-dk2
# Init default config settings
UBOOT_CONFIGS += trusted
UBOOT_DEFCONFIG_trusted += stm32mp13_defconfig
UBOOT_BINARY_stm32mp13_defconfig ?= u-boot.dtb
UBOOT_DEVICETREE_stm32mp13_defconfig ?=  stm32mp135-ebyte
```

When compiling the factory source code, users do not need to modify anything. The uboot source code device tree path adapted to the ECK10-135A5M1G-I development board hardware resources is arch/arm/dts/, and the device tree files are stm32mp135-ebyte.dts and stm32mp13-pinctrl- ebyte .dtsi.

Enter the U-Boot source code directory u-boot-stm32mp-v2022.10-stm32mp-r1 and execute the ./build.sh script to generate the image file under the deploy directory of the parent directory . If the build.sh compilation script comments out r m ../build/ ../deploy/ -rf , you need to open it manually, otherwise the path error will be prompted.

```
#!/bin/sh
# script-name: build.sh
# version: 1.0
# date: 2023-04-08
# author: ebyte
#本脚本用作编译U-boot
#

#删除上层目录的编译文件
rm ../build/ ../deploy/ -rf

#运行编译
make CROSS_COMPILE=arm-none-linux-gnueabihf- -f ../Makefile.sdk all -j8
~
~
~
```

Similar to the previous source code, after the U-Boot source code is compiled, two directories "build" and "deploy" will be generated in the upper directory, that is, the uboot directory, as shown in the figure below.

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

```
hu@ubuntu:~/linux/ebyte/stm32mp131/uboot$ ls
0001-v2022.10-stm32mp-r1-MACHINE.patch
0002-v2022.10-stm32mp-r1-BOARD.patch
0003-v2022.10-stm32mp-r1-MISC-DRIVERS.patch
0004-v2022.10-stm32mp-r1-DEVICETREE.patch
0005-v2022.10-stm32mp-r1-CONFIG.patch
0099-Add-external-var-to-allow-build-of-new-devicetree-fi.patch
build
deploy
Makefile.sdk
README.HOW_TO.txt
series
u-boot-stm32mp-v2022.10-stm32mp-r1
```

The build directory contains all the files generated by the compilation process, most of which are unnecessary for users. The files in the deploy directory are very important. They are processed by the Makefile.sdk script to copy the binary files in the build directory to the deploy directory. It contains two files, u-boot-nodtb-stm32mp13.bin and u-boot-stm32mp135-ebyte-trusted.dtb . These two files are the final products of compiling the factory U-Boot source code, which is the U-Boot firmware of the ECB10-TB13X core board.

4.3.2. Set U boot kernel startup mode

There are many environment variables in U boot, and here are some environment variables that mainly affect the kernel startup mode.

4.3.2.1. bootcmd

Bootcmd stores the default uboot commands. After the uboot countdown ends, the commands in bootcmd will be executed. These commands are generally used to start the Linux kernel, such as reading the Linux kernel image file and device tree file in EMMC or NAND Flash into DRAM, and then starting the Linux kernel. You can enter the command line after uboot starts to set the value of the bootcmd environment variable.

The default value of bootcmd is CONFIG_BOOTCOMMAND, and the default value of bootargs is CONFIG_BOOTARGS. We can set the default value of bootcmd directly in the stm32mp1.h file by setting the macro CONFIG_BOOTCOMMAND.

4.3.2.2. bootargs

bootargs stores the parameters passed by uboot to the Linux kernel, such as specifying the console used by the Linux kernel , specifying the partition where the root file system is located, etc., as shown in the following bootargs environment variable value:

console=ttySTM0,115200 root=/dev/mmcblk2p3 rootwait rw

console is used to set the Linux terminal (or console), that is, what device is used to interact with Linux, is it a serial port or an LCD screen? If it is a serial port, it should be serial port number, etc. Generally, the serial port is set as the

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

Linux terminal, so that we can interact with Linux through MobaXterm on the computer. Here, console is set to ttySTM0, because after Linux starts, the device file of STM32MP1's serial port 4 in Linux is /dev/ttySTM0 . In Linux, everything is a file. There is a ",115200" after ttySTM0, which is to set the baud rate of the serial port. console=ttySTM0,115200 means to set ttySTM0 (that is, serial port 4) as the Linux terminal, and the serial port baud rate is set to 115200. Using the debugging serial port terminal, after the development board starts to uboot, the boot mode is controlled by setting bootargs and bootcmd in uboot. After the setting is completed, the environment variables need to be saved using commands. Subsequently, the kernel and file system can be automatically started from the set environment variables.

root is used to set the location of the root file system, root=/dev/mmcblk2p3 is used to indicate that the root file system is stored in partition 3 of the mmcblk2 device. After the STM32MP1 core board starts Linux, there will be files such as /dev/mmcblk1, /dev/mmcblk2, /dev/mmcblk1p1, /dev/mmcblk1p2, /dev/mmcblk2p1, /dev/mmcblk2p2 and /dev/mmcblk2p3, where /dev/mmcblkx (x=0~n) represents the mmc device, and /dev/mmcblkxpy (x=0~n, y=1~n) represents the partition y of the mmc device x. In the STM32MP1 development board, /dev/mmcblk2 represents EMMC, and /dev/mmcblk2p3 represents partition 3 of EMMC.

There is "rootwait rw" after root. rootwait means to wait for the mmc device to be initialized before mounting. Otherwise, it will be wrong to mount the root file system before the mmc device is initialized. rw means that the root file system can be read and written. Without rw, you may not be able to write in the root file system and can only read.

rootfstype This option is usually used together with root . rootfstype is used to specify the type of the root file system. If the root file system is in ext format, this option does not matter. If the root file system is yaffs , jffs or ubifs , this option needs to be set to specify the type of the root file system.

After setting the bootcmd and bootargs environment variables, you need to save them.

saveenv

```
STM32MP> saveenv
Saving Environment to MMC... Writing to MMC(0)... OK
```

Three different startup methods are introduced below.

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

4.3.2.3. Boot from SD card

```
# setenv bootargs 'console=ttySTM0,115200 root=/dev/mmcblk0p9 rootwait rw'
# setenv bootcmd 'ext4load mmc 0:8 c2000000 uImage;ext4load mmc 0:8 c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000'
```

4.3.2.4. from nandflash

```
# setenv bootcmd 'ubifsmount ubi:boot;ubifsload c2000000 uImage;ubifsload c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000;ubifsmount ubi:rootfs'
# setenv bootargs 'ubi.mtd=UBI rootfstype=ubifs root=ubi:rootfs rootwait rw console=ttySTM0,115200'
```

4.3.2.5. Boot from the network

To boot from the network, you need to set up the network card first, and configure the nfs service and tftp service of the server (virtual machine). You need to configure the IP address , MAC address , gateway, mask and IP address of the virtual machine .

```
# setenv ipaddr 192.168.0.250
# setenv ethaddr 00:04:9f:04:d2:35
# setenv gatewayip 192.168.0.1
# setenv netmask 255.255.255.0
# setenv serverip 192.168.0.130
```

Set up booting from the network

```
# setenv bootcmd 'tftp c2000000 uImage;tftp c4000000 stm32mp135-ebyte.dtb;bootm c2000000 - c4000000'
# setenv bootargs 'console=ttySTM0,115200 root=/dev/nfs nfsroot=192.168.0.130:/home/hu/linux/ubuntu/ubuntu-small,proto=tcp rw ip=192.168.0.250:192.168.0.130:192.168.0.1:255.255.255.0::eth0:off
```

4.3.2.6. Code setting auto start

SDK provided by Ebyte has been set to start automatically. The following content is for user reference.

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

include/configs/stm32mp13_st_common.h in the uboot source code: for STMicroelectronics boards for STM32MP13x.

We can change the way uboot starts the kernel here, so that the kernel can be automatically started after burning the firmware, without having to manually set the startup mode after starting uboot using the method in the above section.

```
#define ST_STM32MP13_BOOTCMD "bootcmd_stm32mp=" \
 "echo \"Boot over ${boot_device}${boot_instance}!\";" \
 "if test ${boot_device} = serial || test ${boot_device} = usb;" \
 "then stm32prog ${boot_device} ${boot_instance}; " \
 "else " \
  "run env_check;" \
  "if test ${boot_device} = mmc;" \
  "then env set bootargs \"console=ttySTM0,115200 root=/dev/mmcblk0p9 rootwait rw\";" \
  "ext4load mmc 0:8 c2000000 uImage;"\
  "ext4load mmc 0:8 c4000000 stm32mp135-ebyte.dtb;"\
  "bootm c2000000 - c4000000;"\
  "fi;" \
  "if test ${boot_device} = nand ||" \
     " test ${boot_device} = spi-nand ;" \
  "then env set bootargs \"ubi.mtd=UBI rootfstype=ubifs root=ubi:rootfs rootwait rw
console=ttySTM0,115200\";" \
  "ubifsmount ubi:boot;"\
  "ubifsload c2000000 uImage;"\
  "ubifsload c4000000 stm32mp135-ebyte.dtb;"\
  "bootm c2000000 - c4000000;"\
  "fi;" \
  "if test ${boot_device} = nor;" \
  "then env set boot_targets mmc0; fi;" \
  "run distro_bootcmd;" \
 "fi;\0"
```


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

As shown in the above code, if it is booted from sd card (mmc), the device tree, kernel and root file system are loaded from the partition of sd card. If it is booted from nand, it is loaded from nand.

At this point, the factory U-Boot source code has been compiled. The following describes how to use the fiptool tool to encapsulate OP-TEE and U-Boot firmware.

4.4. Fiptool encapsulates OP-TEE and U-Boot firmware

This section describes how to use the fiptool tool to package OP-TEE and U-Boot firmware files together into a single image file fip-stm32mp135d-atk-optee.bin, and then burn it to the development board.

FIP, the full name of Firmware Image Package, is a packaging format used by TF-A to package the bootloader image file (bootloader) and other payload files into a single binary file. Now ST recommends using FIP to start the STM32MP1 platform in the ecosystem development kit released. Only the TF-A (with STM32 header structure information) image is loaded by the ROM code, while other binary files such as OP-TEE, U-Boot and their respective device tree files are encapsulated in the form of FIP image packages. TF-A can load and authenticate FIP images when running.

Therefore, the factory system startup and operation process of the ECB10-TB13X core board can be summarized as follows:

ROM code->TF-A->FIP image (including OP-TEE and U-Boot)->Linux kernel->file system.

In the TF-A source code, a tool called fiptool is provided to implement the FIP function. Next, we enter the fiptool source code directory of TF-A and compile it to generate the fiptool tool. The compilation instructions are as follows:

```
# cd tf-a-stm32mp-v2.8.6-stm32mp-r1 / //Enter your TF-A source directory.
# cd tools/fiptool //Enter fiptool source code.
# make //Compile fiptool source code.
```

```
hu@ubuntu:~/linux/ebyte/stm32mp131/tf-a/tf-a-stm32mp-v2.8.6-stm32mp-r1/tools/fip
tool$ make
  HOSTCC  fiptool.c
  HOSTCC  tbbr_config.c
  HOSTLD  fiptool

Built fiptool successfully

hu@ubuntu:~/linux/ebyte/stm32mp131/tf-a/tf-a-stm32mp-v2.8.6-stm32mp-r1/tools/fip
tool$ ls
fiptool     fiptool.o           Makefile.msvc  tbbr_config.o
fiptool.c   fiptool_platform.h  tbbr_config.c  win_posix.c
fiptool.h   Makefile            tbbr_config.h  win_posix.h
```

Copy the fiptool tool compiled above to the /usr/bin/ directory of ubuntu

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

```
# sudo cp fiptool /usr/bin/
```

```
hu@ubuntu:/usr/bin$ ls fiptool
fiptool
```

Execute the following command to view the fiptool tool help information;

```
# fiptool help
```

```
hu@ubuntu:/usr/bin$ fiptool help
usage: fiptool [--verbose] <command> [<args>]
Global options supported:
  --verbose     Enable verbose output for all commands.

Commands supported:
  info          List images contained in FIP.
  create        Create a new FIP with the given images.
  update        Update an existing FIP with the given images.
  unpack        Unpack images from FIP.
  remove        Remove images from FIP.
  version       Show fiptool version.
  help          Show help for given command.
```

Next, we use the fiptool tool to package binary image files such as OP-TEE and U-Boot , and encapsulate the FIP image package fip-stm32mp135-ebyte-optee.bin .

First, we create a directory named "fip" in the ubuntu directory, and copy the six image files generated after compiling TF-A, OP-TEE and U-Boot to the fip directory. The required files are shown in the following table.

Source code	**Directory**	**file name**
TFA	TF-A/ deploy / fwconfig	stm32mp135-ebyte-fw-config-optee.dtb
Optee	Opti / deploy	tee-header_v2-stm32mp135-ebyte.bin tee-pager_v2-stm32mp135-ebyte.bin tee-pageable_v2-stm32mp135-ebyte.bin
U - Boot	U boot / deploy	u-boot-stm32mp135-ebyte-trusted.dtb u-boot-nodtb-stm32mp13.bin

Make sure the above files exist in the fip directory

fip$ tree -l

├── stm32mp135-ebyte-fw-config-optee.dtb

├── tee-header_v2-stm32mp135-ebyte.bin

├── tee-pageable_v2-stm32mp135-ebyte.bin

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

├── tee-pager_v2-stm32mp135-ebyte.bin

├── u-boot-nodtb-stm32mp13.bin

└── u-boot-stm32mp135-ebyte-trusted.dtb

Then enter the fip directory and execute the following command

```
# fiptool create --nt-fw u-boot-nodtb-stm32mp13.bin --hw-config u-boot-stm32mp135-ebyte-trusted.dtb --fw-config stm32mp135-ebyte-fw-config-optee.dtb --tos-fw tee-header_v2-stm32mp135-ebyte.bin --tos-fw-extra1 tee-pager_v2-stm32mp135-ebyte.bin --tos-fw-extra2 tee-pageable_v2-stm32mp135-ebyte.bin fip-stm32mp135-ebyte-optee.bin
```

The fip-stm32mp135-ebyte-optee.bin file will be generated. This binary file contains the OP-TEE firmware and U-Boot firmware. When the development board is started up, TF-A will load and authenticate the firmware, and then run the next stage of OP-TEE and U-Boot.

```
hu@ubuntu:~/linux/ebyte/stm32mp131/fip$ ls
fip-stm32mp135-ebyte-optee.bin      tee-pager_v2-stm32mp135-ebyte.bin
stm32mp135-ebyte-fw-config-optee.dtb  u-boot-nodtb-stm32mp13.bin
tee-header_v2-stm32mp135-ebyte.bin    u-boot-stm32mp135-ebyte-trusted.dtb
tee-pageable_v2-stm32mp135-ebyte.bin
```

4.5. Kernel

4.5.1. Get the source code and compile

Create a directory named "linux" in the ubuntu directory, and then copy the factory linux source package 04_Source/ linux- 6.1.28.tar.gz to the linux directory.

```
hu@ubuntu:~/linux/ebyte/stm32mp131/linux-6.1.28$ ls
arch             include      modules.builtin          System.map
block            init         modules.builtin.modinfo  tmp
build.sh         io_uring     modules.order            tools
built-in.a       ipc          Module.symvers           usr
certs            Kbuild       net                      virt
CONTRIBUTING.md  Kconfig      README                   vmlinux
COPYING          kernel       rust                     vmlinux.a
CREDITS          lib          samples                  vmlinux.o
crypto           LICENSES     scripts                  vmlinux.symvers
Documentation    MAINTAINERS  security
drivers          Makefile     SECURITY.md
fs               mm           sound
```

When compiling the factory source code, users do not need to modify anything. The uboot source code device tree path adapted to the hardware resources of the ECK10-135A5M1G-I development board is arch/arm / boot/dts/, and the device tree files are stm32mp135-ebyte.dts and stm32mp13-pinctrl- ebyte.dtsi . To facilitate your compilation, a

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

compilation script is provided. You can directly run the script compilation when compiling for the first time. The script is shown in the figure below:

```
make distclean #清除编译      You, 1秒钟前 • Uncommitted changes
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- stm32mp1_ebyte_defconfig

# make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- menuconfig
#编译内核
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- uImage vmlinux dtbs LOADADDR=0xC2000040 -j8

#编译内核模块
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- modules -j16

#在当前目录下新建一个tmp目录，用于存放编译后的目标文件
if [ ! -e "./tmp" ]; then
    mkdir tmp
fi
rm -rf tmp/*

#将编译好的模块安装到tmp目录，通过INSTALL_MOD_STRIP=1移除模块调试信息
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- modules_install INSTALL_MOD_PATH=tmp INSTALL_MOD_STRIP=1
#删除模块目录下的source目录
rm -rf tmp/lib/modules/6.1.28/source
#删除模块的目录下的build目录
rm -rf tmp/lib/modules/6.1.28/build

#跳转到模块目录
cd tmp/lib/modules
#压缩内核模块
tar -jcvf ../../modules.tar.bz2 .
cd -
rm -rf tmp/lib

#拷贝uImage到tmp目录
cp arch/arm/boot/uImage tmp

#拷贝所有编译的设备树文件到当前的tmp目录下
cp arch/arm/boot/dts/stm32mp135-ebyte.dtb tmp
```

Then execute the ./build . sh compilation script. The first compilation may be slow. After compilation, the u Image image and stm 32mp135-ebyte.dtb device tree file will be generated in the tmp folder under the root directory.

4.5.2. System image packaging

4.5.2.1. new ext4 format disk

First, create a disk file in ext4 format, then mount the disk in ext4 format, and copy stm32mp1 35 -ebyte.dtb and uImage to the ext4 disk.

Create a bootfs folder and then enter the following command to create an ext4 disk:

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

```
# mkdir -p bootfs
# dd if=/dev/zero of=bootfs.ext4 bs=1M count=10
# mkfs.ext4 -L bootfs bootfs.ext4
```

The above command creates a 10M ext4 disk .

4.5.2.2. Copy the system image to the ext4 disk

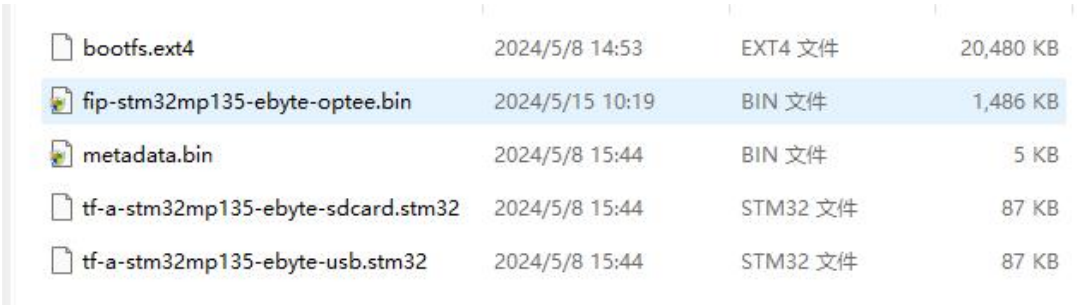
First, create a directory to mount the bootfs.ext4 created earlier. For example, I create the directory /mnt/bootfs here. The command is as follows:

```
sudo mkdir /mnt/bootfs
Then mount the bootfs.ext4 created above to bootfs
sudo mount bootfs.ext4 /mnt/bootfs/
Then copy u Image and stm32mp135-ebyte.dtb to bootfs, and then cancel the mount
sudo cp uImage stm32mp135-ebyte.dtb /mnt/bootfs/
sudo umount /mnt/bootfs.
```

In this way, we have successfully packaged the kernel's device tree and uImage.

4.6. summary

Through the above steps, we get the image file of all source code parts, and take out the files we need for burning, as shown in the figure below.



Among them , metadata.bin , tf -a-stm32mp135-ebyte-sdcard.stm32 and tf -a-stm32mp135-ebyte- usb.stm32 are from the deploy directory of tf -a . Then follow the burning steps in Chapter 3 to burn. If you do not need to create a file system yourself, you can use the file system provided by Ebyte to burn.

5. How to adapt to different hardware platforms

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

5.1. Adding hardware resources to the device tree

have a detailed understanding of the CPU chip manual, the ECB10-TB13X core board product manual, and the pin definitions in order to correctly configure and use these pins according to their actual functions.

Users can create their own device tree in the BSP source code. Generally, there is no need to modify TF-a, optee and u-boot in the Bootloader part. Users only need to make appropriate adjustments to the Linux kernel device tree according to the actual hardware resources. Here is a list of device trees in various parts of the BSP of ECB10-TB13X for user development.

project	Device Tree	illustrate
TF - A	s tm32mp13x-base.dts	Resource Device Tree
	s tm32mp131-ebyte.dts stm32mp131-ebyte-fw-config.dts s tm32mp135-ebyte.dts stm32mp135-ebyte-fw-config.dts	Core board resource description
	stm32mp13-ddr3-1x2Gb-1066- binF .dtsi stm32mp13-ddr3-1x4Gb-1066- binF .dtsi	Single chip 256MB DDR3 Description Single chip 512 MB DDR3 Description
	stm32mp13-pinctrl-ebyte.dtsi	Pin multiplexing description
Opti	s tm32mp13x-base.dts	Resource Device Tree
	s tm32mp131-ebyte.dts s tm32mp135-ebyte.dts	Core board resource description
	stm32mp13-pinctrl-ebyte.dtsi	Pin multiplexing description
U - Boot	stm32mp13x-base.dts	Resource Device Tree
	s tm32mp131-ebyte.dts stm32mp131-ebyte-u-boot.dtsi s tm32mp135-ebyte.dts stm32mp135-ebyte-u-boot.dtsi	Core board resource description
	stm32mp13-pinctrl-ebyte.dtsi	Pin multiplexing description

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

Kernel	stm32mp1_ebyte_defconfig	Kernel configuration files
	stm32mp13x-base.dts	Resource Device Tree
	s tm32mp131-ebyte.dts	ECB10-TB131A Device Tree
	s tm32mp135-ebyte.dts	ECB10-TB13X Device Tree
	stm32mp13-pinctrl-ebyte.dtsi	ECK10-131A2M1G-I

Users generally only need to add different hardware resources to the kernel's device tree, and then adapt the driver according to the specific hardware resources.

5.2. Adding device tree

The Linux kernel device tree is a data structure that describes the on-chip and off-chip device information in a unique syntax format. It is passed to the kernel by the BootLoader, and the kernel parses it to form a dev structure associated with the driver for use by the driver code.

Under the kernel source code, you can see a large number of platform device trees in arch/arm/boot/dts. For example, if the device tree is suitable for ECK10-135A5M1G-I , you can add a custom device tree under the current path, such as: ebyte-stm32mp135 a -xxx.dts. As shown in the figure below.

```
hu@ubuntu:~/linux/ebyte/stm32mp131/linux-6.1.28/arch/arm/boot/dts$ ls stm32mp13*
stm32mp131.dtsi       stm32mp133.dtsi       stm32mp135-ebyte.dts          stm32mp135f-dk.dtb           stm32mp13-pinctrl.dtsi  stm32mp13xc.dtsi
stm32mp131-ebyte.dtb  stm32mp135.dtsi       stm32mp135f-dk-a7-examples.dtb  stm32mp135f-dk.dts         stm32mp13xa.dtsi        stm32mp13xd.dtsi
stm32mp131-ebyte.dts  stm32mp135-ebyte.dtb  stm32mp135f-dk-a7-examples.dts  stm32mp13-ebyte-pinctrl.dtsi  stm32mp13x-base.dts     stm32mp13xf.dtsi
```

We write the resources related to the ECB10-TB13X core board into stm32mp135-base.dts. Other extended interfaces and devices can reference them as shown below (for reference only):

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

```
/dts-v1/;

#include "stm32mp13x-base.dts"

/ {
    model = "EBYTE TECH STM32MP131 Discovery Board";
    compatible = "st,stm32mp131-ebyte", "st,stm32mp135";

    memory@c0000000 {
        device_type = "memory";
        reg = <0xc0000000 0x10000000>;
    };

    reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;

        optee_framebuffer@cd000000 {
            reg = <0xcd000000 0x1000000>;
            no-map;
        };
        optee@ce000000 {
            reg = <0xce000000 0x2000000>;
            no-map;
        };
    };

```

After the user adds a new device tree source file, it is also necessary to add the device tree compilation information in the Makefile in the same directory, so that the corresponding device tree binary file can be generated when compiling the kernel.

In arch/arm/boot/dts/ Makefile

stm32mp131-ebyte.dtb under the configuration item of dtb-$(CONFIG_ARCH_STM32) += \

```
stm32mp157f-ev1.dtb \
stm32mp157f-ev1-a7-examples.dtb \
stm32mp157f-ev1-m4-examples.dtb \
stm32mp131-ebyte.dtb \
stm32mp135-ebyte.dtb
```

After the addition is completed, you can add the onboard description of the device driver and compile and generate the device tree dtb file stm32mp13 1 -ebyte.dtb using the method in Section 4.5.1 .

5.3. How to configure CPU function pins according to hardware

Implementing the control of a functional pin is one of the more complex system development processes, which includes steps such as pin configuration, driver development, application implementation, etc. This section does not specifically analyze the development process of each part, but uses examples to explain the control implementation of

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

functional pins.

5.3.1. G PIO pin configuration method

GPIO: General-purpose input/output, general-purpose input/output port, is a very important resource in embedded devices. It can output high and low levels or read the status of pins through it - high or low level. STM32MP1 encapsulates a large number of peripheral controllers. These peripheral controllers and external devices are generally realized by controlling GPIO. The use of GPIO by peripheral controllers is called alternate function, which gives them more complex functions. For example, users can exchange data with external hardware through GPIO ports (such as UART), control hardware work (such as LED, buzzer, etc.), read hardware work status signals (such as interrupt signals), etc. Therefore, GPIO ports are widely used.

The GPIO pin configuration method of STM32MPU is generally configured using STM32CubeMX or using the Datasheet lookup method.

5.3.1.1. STM32CubeMX configuration method

Currently ST has added the STM32MPU series CPU to STM32CubeMX. We can also use this tool to configure TF-A, U-boot and Kernel's GPIO function device tree and peripheral clock. This section does not focus on its usage. You can get detailed development instructions from the official website.

WIKI: https://wiki.st.com/stm32mpu/wiki/STM32CubeMX

ST official: https://www.st.com/zh/development-tools/stm32cubemx.html

5.3.1.2. Query Manual

The configuration of GPIO can be found in the description file of the Datasheet compiled by Ebyte (01_Documents\Datasheet\STM32MP135A&D Datasheet.pdf), the example is as follows:

```
i2c1_pins_a: i2c1-0 {
    pins {      You, 上个月 • add ebyte board support
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
        bias-disable;
        drive-open-drain;
        slew-rate = <0>;
    };
};
```

By consulting the data sheet, we can find that the multiplexing relationship of I2C_SCL on the PB8 pin is A F4 , and the multiplexing relationship of I2C_SDA on the PD3 pin is A F5 .

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

PB7	-	TIM17_CH1N	TIM4_CH2	-	-	I2S4_CK	I2C4_SDA	-
PB8	-	TIM16_CH1	TIM4_CH3	-	I2C1_SCL	I2C3_SCL	DFSDM1_ DATIN1	-
PB9	TRACED3	-	TIM4_CH4	-	-	-	I2C4_SDA	-
PD2	TRACED4	-	TIM3_ETR	-	I2C1_SMBA	SPI3_NSS/ I2S3_WS	SAI2_D1	USART3_RX
PD3	-	-	TIM2_CH1/ TIM2_ETR	USART2_CTS/ USART2_NSS	DFSDM1_ CKOUT	I2C1_SDA	SAI1_D3	-
PD4	-	-	-	USART2_RTS/	-	SPI3_MISO/	DFSDM1_	-

5.3.2. PIO in the device tree

Configure the function pins as I2C function instances. I2C mainly uses two GPIOs as I2C_SCL and I2C_SDA . After defining the pin positions and configurations of I2C in the pinctrl subsystem , add references to them in the board-level device tree, as shown in the figure below. Here, I2C uses two pin configurations, the default state for normal use and the pin state in sleep mode, which are default and sleep respectively. The system automatically switches according to the state.

```
&i2c1 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&i2c1_pins_a>;
    pinctrl-1 = <&i2c1_sleep_pins_a>;
    i2c-scl-rising-time-ns = <100>;
    i2c-scl-falling-time-ns = <7>;
    status = "okay";
    /delete-property/dmas;
    /delete-property/dma-names;
```

I2C pins in the pinctrl system is shown in the figure below.

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

```
i2c1_pins_a: i2c1-0 {       You, 上个月 • add ebyte board support
    pins {
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
        bias-disable;
        drive-open-drain;
        slew-rate = <0>;
    };
};

i2c1_sleep_pins_a: i2c1-sleep-0 {
    pins {
        pinmux = <STM32_PINMUX('B', 8, AF4)>, /* I2C1_SCL */
        <STM32_PINMUX('D', 3, AF5)>; /* I2C1_SDA */
    };
};
```

5.3.3. How to use the pins you configure

5.3.3.1. Using GPIO pins in U-boot

Uboot can use commands directly in the terminal to control the GPIO settings. For example, to set GPIO B6 , use the following command.

```
STM32MP> gpio set GPIOB6
gpio: pin GPIOB6 (gpio 22) value is 1
STM32MP> gpio clear GPIOB6
gpio: pin GPIOB6 (gpio 22) value is 0
```

5.3.3.2. User space uses GPIO pins

The architecture of the Linux operating system is divided into user state and kernel state (or user space and kernel). User state is the activity space of upper-level applications. The execution of applications must rely on the resources provided by the kernel, including CPU resources, storage resources, I/O resources, etc. In order for upper-level applications to access these resources, the kernel must provide an access interface for upper-level applications: that is, system calls.

Shell is a special application, commonly known as the command line, which is essentially a command interpreter. It communicates with system calls and various applications. Using Shell scripts, a few short lines of Shell scripts can usually achieve a very large function. The reason is that these Shell statements usually encapsulate the system calls. In order to facilitate the interaction between users and the system.

This section describes how to use the three basic methods of controlling GPIO pins in user mode.

5.3.3.2.1. Shell commands

The GPIO test is implemented through the file system sysfs interface. The following content uses PB6 as an

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

example to illustrate the use of GPIO.

pin number is defined as #define PIN_NO(port, line) (((port) - 'A') * 0x10 + (line)), where port is the gpio port, line is the corresponding pin of the gpio, and ((port)-'A') represents the subtraction of ASCII codes. For example, the pin number corresponding to PB6 is: PIN_NO(' B ' , 6) = (0x42-0x41)*0x10+ 6 = (66-65)*16+ 6 = 22 .

1） Export G PIO

echo 94 > /sys/class/gpio/export

PB6 directory will be generated in the /sys/class/gpio/ directory .

2） Set/view GPIO direction

Setting up input

echo "in" > /sys/class/gpio/PB6/direction

Setting up output

echo " out " > /sys/class/gpio/PB6/direction

Check GPIO direction

cat /sys/class/gpio/PB6/direction

o ut

3） Set/view GPIO value

Set output low

echo "0" > /sys/class/gpio/PB6/value

Set output high

echo "1" > /sys/class/gpio/PB6/value

Check the value of G PIO

cat /sys/class/gpio/PB6/value

It can be seen that PB6 outputs a high level. You can use a multimeter to measure the PB6 pin of the extended IO and you can see that the voltage is around 3.3V.

In addition, if users need to control GPIO in the application, they can refer to the wiki:

https://wiki.st.com/stm32mpu/wiki/How_to_control_a_GPIO_in_userspace

5.3.3.2.2. Library Functions

Starting from Linux 4.8, Linux introduced a new gpio operation mode, GPIO character device. It is no longer recommended to use the previous SYSFS mode to operate GPIO in the "/sys/class/gpio" directory, but based on the "file descriptor" character device, each GPIO group has a corresponding gpiochip file under "/dev", for example,

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

"/dev/gpiochip0 corresponds to GPIOA, /dev/gpiochip1 corresponds to GPIOB" and so on.

Libgpiod library functions implement gpiochip-based methods, providing some tools and a simpler C API interface. Libgpiod (Library General Purpose Input/Output device) provides a complete API for developers, and also provides some user space applications to operate GPIO.

Libgpiod common basic interface description:

gpiodetect - Lists all gpiochips present in the system, their names, labels and GPIO line numbers.

gpioinfo - Lists all rows for the specified gpiochips, their name, user, direction, activity state, and additional flags.

gpioget - Reads the value of a specified GPIO line.

gpioset - Sets the specified GPIO line values, potentially keeping those lines exported and waiting for a timeout, user input, or a signal.

gpiofind - Find the gpiochip name and line offset for a given line name.

gpiomon - Waits for events on a GPIO line, specifying which events to watch for, how many events to process before exiting, or whether events should be reported to the console.

6. Making a file system

This is the last step of Linux system transplantation. After the root file system is built, it means that we have a complete and runnable minimum system. In the future, we will write and test Linux drivers on this minimum system, transplant some third-party components, and gradually improve this minimum system. Finally, we will get a relatively complete operating system with complete functions and complete drivers.

6.1. Buildroot creates a minimal system

6.1.1. Get the source code and compile

the buildroot source code buildroot-2020.02.6.tar.bz 2 to Ubuntu and decompress it after the copy is complete. Ebyte provides an example of a buildroot file system that can be burned directly.

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

```
# tar -vxjf buildroot-2020.02.6.tar.bz2
```

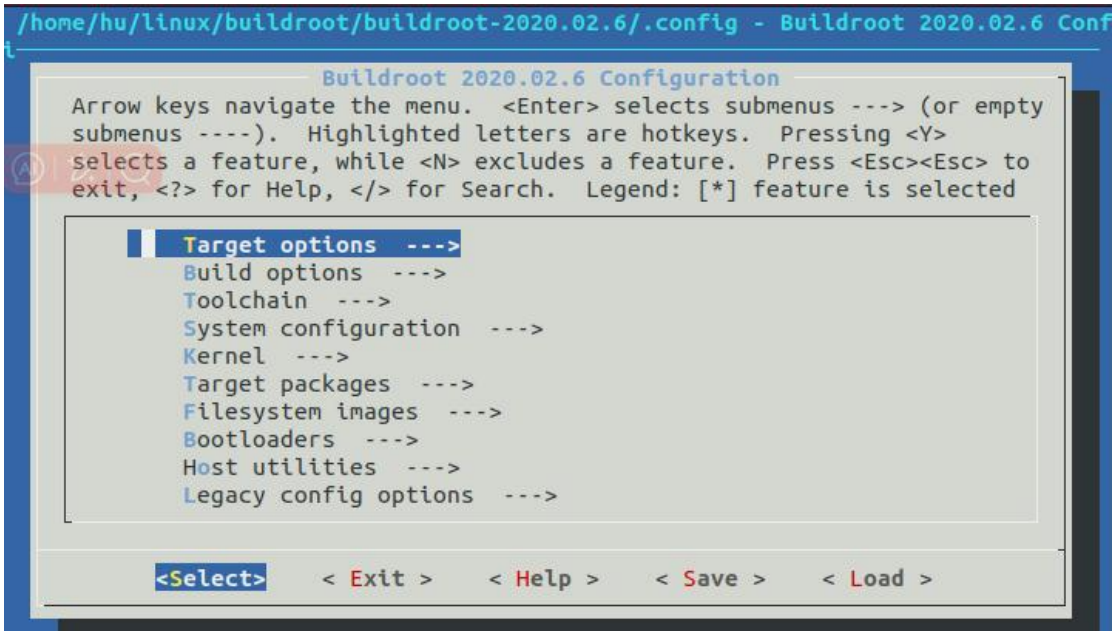
After decompression, it is as shown below

```
hu@ubuntu:~/linux/buildroot$ cd buildroot-2020.02.6/
hu@ubuntu:~/linux/buildroot/buildroot-2020.02.6$ ls
arch              configs     linux            stm32mp1_ebyte_defconfig
board             COPYING     Makefile         stm32mp1_ebyte_defconfig.old
boot              DEVELOPERS  Makefile.legacy  support
CHANGES           dl          output           system
Config.in         docs        package          toolchain
Config.in.legacy  fs          README           utils
```

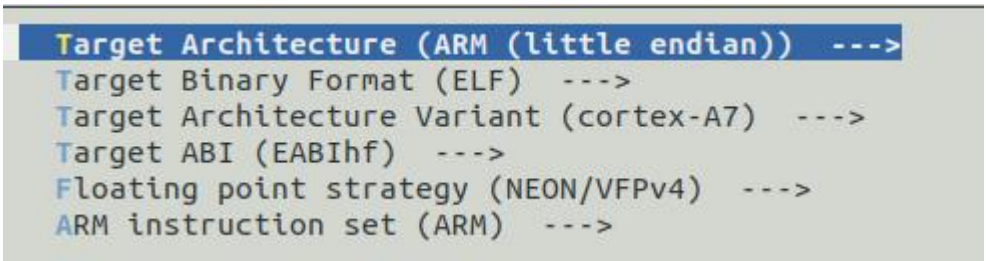
6.1.2. Configure buildroot

We use the graphical interface to configure buildroot and use the following command in the root directory:

```
# make menuconfig
```



Configuring targets options, the configuration results are as follows.



Configure the toolchain. The configuration results are as follows:

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

```
    Toolchain type (External toolchain)  --->
    *** Toolchain External Options ***
    Toolchain (Custom toolchain)  --->
    Toolchain origin (Pre-installed toolchain)  --->
(/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf) Toolchain path
($(ARCH)-none-linux-gnueabihf) Toolchain prefix
    External toolchain gcc version (9.x)  --->
    External toolchain kernel headers series (4.20.x)  --->
    External toolchain C library (glibc/eglibc)  --->
[*] Toolchain has SSP support?
[*]   Toolchain has SSP strong support?
[*] Toolchain has RPC support?
[*] Toolchain has C++ support?
[ ] Toolchain has D support?
[ ] Toolchain has Fortran support?
[ ] Toolchain has OpenMP support?
[ ] Copy gdb server to the Target
    *** Host GDB Options ***
[ ] Build cross gdb for the host
    *** Toolchain Generic Options ***
[ ] Copy gconv libraries
()  Extra toolchain libraries to be copied to target
[*] Enable MMU support
()  Target Optimizations
()  Target linker options
[ ] Register toolchain within Eclipse Buildroot plug-in
```

Configure system configuration

-> System hostname = stm32mp1 //Platform name, set by yourself

-> System banner = Welcome to STM32MP1 35 //Welcome message

-> Init system = BusyBox //Use busybox

->/dev management = Dynamic using devtmpfs + mdev //Use mdev

-> [*] Enable root login with password (NEW) // Enable login password

-> Root password = 123456 //The login password is 123456

Configuring Filesystem images

-> [*] ext2/3/4 root filesystem //If it is EMMC or SD card, use ext3/ext4

-> ext2/3/4 variant = ext4 //Select ext4 format

-> exact size = 1G //ext4 format root file system 1GB (modify according to actual situation)

-> [*] ubi image containing an ubifs root filesystem //If using NAND, use ubifs

Configure the kernel to support UBIFS

Device Drivers --->Memory Technology Device (MTD) support --->UBI - Unsorted block images --->Enable UBI

Configure mtd to support UBI interface

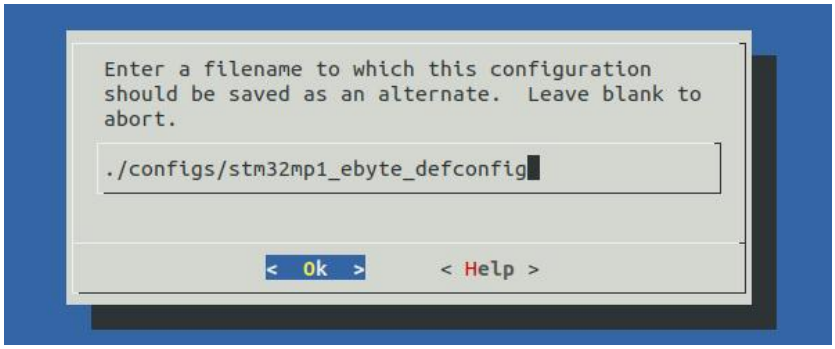
File systems --->Miscellaneous filesystems --->UBIFS file system support

Save configuration items

and kernel, it is best to save the configuration items after configuring buildroot through the graphical interface to

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

prevent the configuration items from being deleted after clearing the project. The default configuration items of buildroot are saved in the configs directory. After the configuration is completed, select <Save> and then enter the name of the configuration item to be set, as shown in the figure:



Compile buildroot

Just use make -j 8 to compile.

6.2. Use ubuntu -base to create a system

ubuntu-base is the Ubuntu minimum file system built by Ubuntu, including the debain package manager. The size of the basic package is usually only tens of megabytes, and it is supported by the entire ubuntu software source. Ubuntu software is generally stable. Linux software can be installed on demand based on ubuntu-base, and it is deeply customizable. It is often used for embedded rootfs construction. Ebyte provides an example of an ubuntu file system that can be burned directly.

6.2.1. Get the source code

Official website address:

http://cdimage.ubuntu.com/ubuntu-base/releases/

Or get 04_Tools /ubuntu-base-18.04.5-base-armhf.tar.gz and unzip it

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



6.2.2. Preparing the chroot environment

Install the emulator

```
# sudo apt-get install qemu-user-static

# sudo cp /usr/bin/qemu-arm-static ./ubuntu-1804/usr/bin/

Add DNS configuration, and you can directly use the network update package later

# echo "nameserver 8.8.8.8" > ./ubuntu-1804/etc/resolv.conf
```

Create a mount script

Copy the following script to the ch-mount.sh file and change the permission to executable.

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

```
#!/bin/bash

function mnt() {
        echo "MOUNTING"
        sudo mount -t proc /proc ${2}/proc
        sudo mount -t sysfs /sys ${2}/sys
        sudo mount -o bind /dev ${2}/dev
        sudo chroot ${2}
}
function umnt(){
        echo "UNMOUNTING"
        sudo umount ${2}/proc
        sudo umount ${2}/sys
        sudo umount ${2}/dev
}
if [ "$1" == "-m" ] && [ -n "$2" ] ;
then
        mnt $1 $2
elif [ "$1" == "-u" ] && [ -n "$2" ];
then
        umnt $1 $2
else
        echo ""
        echo "Either 1'st, 2'nd or both parameters were missing"
        echo ""
        echo "1'st parameter can be one of these: -m(mount) OR -u(umount)"
        echo "2'nd parameter is the full path of rootfs directory(with trailing '/')"
        echo ""
        echo "For example: ch-mount -m /media/sdcard/"
        echo ""
        echo 1st parameter : ${1}
        echo 2nd parameter : ${2}
fi
```

6.2.3. Installation package file

Mounting system

```
# ./ch-mount.sh -m ubuntu-1804
```

```
hu@ubuntu:~/linux/ubuntu$ ./ch-mount.sh -m ubuntu-1804
MOUNTING
[sudo] password for hu:
root@ubuntu:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ubuntu:/#
```

After the mount is successful, you can configure the Ubuntu file system and install some necessary software.

Basic package installation

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

```
# apt update
# apt install sudo
# apt install language-pack-en-base
# apt install vim
# apt install ssh
# apt install net-tools
# apt install ethtool
# apt install ifupdown
# apt install iputils-ping
# apt install rsyslog
# apt install htop
Add log, user debugging Ubuntu system debugging
#touch /var/log/rsyslog
#chown syslog:adm /var/log/rsyslog
#chmod 666 /var/log/rsyslog
#systemctl unmask rsyslog
#systemctl enable rsyslog
Install network and language pack support
#apt-get install synaptic
#apt-get install network-manager network-manager-gnome
#apt-get install language-pack-zh-hant language-pack-zh-hans
#apt-get install rfkill
#apt install -y --force-yes --no-install-recommends fonts-wqy-microhei
#apt install -y --force-yes --no-install-recommends ttf-wqy-zenhei
Desktop system installation
Xfce4 desktop system installation
#apt-get install xinit
#apt-get install xfce4
```


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

Set root password

#passwd root

Create a user named: ebyte

#adduser ebyte

Set login serial port

#systemctl enable getty@ttySTM0.service

6.2.4. Uninstall System

After completing the above steps, you can uninstall the system. Simply enter exit in the system to exit the system and use the command to uninstall.

```
root@ubuntu:/# exit
exit
hu@ubuntu:~/linux/ubuntu$ ./ch-mount.sh -u ubuntu-1804/
UNMOUNTING
```

7. Create an image file that can be burned to N and Flash

After we use buildroot or ubuntu to build, we get a folder. The one built by buildroot is in the output directory, and Ubuntu is our ubuntu-base folder. Nandflash cannot directly recognize the image in ext 4 format, and it needs to be packaged into ubi volume format before burning. We use the following steps to make a ubi format image.

First, confirm the Nand parameters, which can be queried through the datasheet and uboot commands.

mtdlist

ubiinfo

```
UBI: MTD device name:            "UBI"
UBI: MTD device size:            1018 MiB
UBI: physical eraseblock size:   262144 bytes (256 KiB)
UBI: logical eraseblock size:    253952 bytes
UBI: number of good PEBs:        4068
UBI: number of bad PEBs:         4
UBI: smallest flash I/O unit:    4096
UBI: VID header offset:          4096 (aligned 4096)
UBI: data offset:                8192
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     4
UBI: available PEBs:             0
UBI: total number of reserved PEBs: 4068
UBI: number of PEBs reserved for bad PEB handling: 76
UBI: max/mean erase counter: 2/0
```


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

According to ST 's requirements, the ubi image contains uboot_config, uboot_config_r, boot.img, and rootfs.img.

We create the boot , rootfs, uboot_config and uboot _config _ r folders, and put all kernel files into the boot directory and the file system into the rootfs directory. Be careful not to nest folders. uboot _config and uboot _config_r can be empty.

Fill in the command according to the parameters obtained above to generate 4 img files

```
# mkfs.ubifs -F -v -r /home/hu/linux/ubi_flash/boot -o boot.img -m 4096 -e 253952 -c 4072
# mkfs.ubifs -F -v -r /home/hu/linux/ubi_flash/rootfs -o rootfs.img -m 4096 -e 253952 -c 4072
# mkfs.ubifs -F -v -r /home/hu/linux/ubi_flash/uboot_config -o uboot_config.img -m 4096 -e 253952 -c 4072
# mkfs.ubifs -F -v -r /home/hu/linux/ubi_flash/uboot_config -o uboot_config_r.img -m 4096 -e 253952 -c 4072
```

The meanings of the parameters here are as follows.

```
-m, --min-io-size=SIZE      minimum I/O unit size
-e, --leb-size=SIZE         logical erase block size
-c, --max-leb-cnt=COUNT     maximum logical erase block count
-o, --output=FILE           output to FILE
-F, --space-fixup           file-system free space has to be fixed up on first mount
-v, --verbose               verbose operation
-r, -d, --root=DIR          build file system from directory DIR
```

First create volumes, then divide the created images into volumes and put them into ubifs. You need to create the configuration file ubinize.cfg for each partition table first , and then execute the ubinize command.

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

```
ubinize -v -o ebyte_135.ubi -m 4096 -p 256KiB -s 4096 ubinize.cfg
```

The parameters of Ubinize are as follows

```
-o, --output=<file name>          output file name
-v, --verbose                      be verbose
-m, --min-io-size=<bytes>         minimum input/output unit size of the flash
                                       in bytes
-s, --sub-page-size=<bytes>     minimum input/output unit used for UBI
                                       headers, e.g. sub-page size in case of NAND
                                       flash (equivalent to the minimum input/output
                                       unit size by default)
-p, --peb-size=<bytes>            size of the physical eraseblock of the flash
                                       this UBI image is created for in bytes,
                                       kilobytes (KiB), or megabytes (MiB)
                                       (mandatory parameter)
```

ubinize.cfg configuration file is as follows.

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

```
[uboot_config]
mode=ubi
image=uboot_config.img #File generated by mkfs.ubifs
vol_id=0 #Volume serial number
vol_size=4MiB #Volume size
vol_name=uboot_config #Volume name
vol_type=static

[uboot_config_r]
mode=ubi
image=uboot_config_r.img #File generated by mkfs.ubifs
vol_id=1 #Volume serial number
vol_size=4MiB #Volume size
vol_name=uboot_config_r #Volume name
vol_type=static

[boot]
mode=ubi
image=boot.img #File generated by mkfs.ubifs
vol_id=2 #Volume serial number
vol_size=12MiB #Volume size
vol_type=dynamic #Dynamic volume
vol_alignment=1
vol_name=boot #Volume name
```


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

```
[rootfs]
mode=ubi
image=rootfs.img #File generated by mkfs.ubifs
vol_id=3 #Volume serial number
vol_size=10MiB #Volume size
vol_type=dynamic #Dynamic volume
vol_alignment=1
vol_name=rootfs #Volume name
vol_flags=autoresize
```

The parameters of the above files, such as the volume size, need to be set according to the actual size of the generated img file, which is generally larger than the actual size.

8. References

- Linux kernel open source community:

  https://www.kernel.org/

- STM32MPU Development Community:

  https://wiki.st.com/stm32mpu/wiki/Development_zone

- STM32MP131 Datasheet
- STM32MP13 5 Data Sheet

9. Revision Notes

Revision Notes Table

Version	Modifications	Modification time	prepared by	Proofreading	Approval
V1.0	First Draft	24-0 6 - 10	HSL	WYQ	WFX
V 1.1	Updated several issues	2 5-3-10	HSL	WYQ	WFX

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

10. 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.

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