Software Development Guide
ECB10-PGD



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

Chengdu Ebyte Electronic Technology Co.,Ltd.

Contents

Disclaimer 2
1. Overview 3
1.1. Software Resources 3
2. How to burn the system image 3
2.1. Programming using STM32CubeProg 3
2.2. Create an SD card launcher 5
2.3. Modify the startup configuration 6
3. Development environment preparation 7
3.1. Vmware shared directory 8
3.2. NFS service 9
3.3. SSH service 11
3.4. FTP service 11
3.5. Windows installation software development tools 12
3.6. U Ubuntu configuration development environment 14
4. How to build an image 16
4.1. TF-A 17
4.2. Optee-os 21
4.3. U-boot 22
4.4. Kernel 29
4.5. summary 30
5. How to adapt to different hardware platforms 30
5.1. Adding hardware resources to the device tree 30
5.2. Adding device tree 31
5.3. How to configure CPU function pins according to hardware 33
6. Making a file system 37
6.1. Buildroot to create a minimal system 37
6.2. Use ubuntu -base to create a system 40
7. References 44
8. Revision Notes 44
9. About Us 45

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

Disclaimer

EBYTE reserves all rights to this document and the information contained herein. Products, names, logos and designs described herein may in whole or in part be subject to intellectual property rights. Reproduction, use, modification or disclosure to third parties of this document or any part thereof without the express permission of EBYTE is strictly prohibited.

The information contained herein is provided “as is” and EBYTE assumes no liability for the use of the information. No warranty, either express or implied, is given, including but not limited, with respect to the accuracy, correctness, reliability and fitness for a particular purpose of the information. This document may be revised by EBYTE at any time. For most recent documents, visit www.cdebyte.com.

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

1. Overview

This guide describes the complete process of customizing an embedded Linux system based on the Ebyte core board, including preparing the development environment, obtaining code, porting the bootloader and kernel, and customizing the rootfs filesystem to suit your application needs. We first explain how to build a system image for the ECB10 development board based on the provided source code and how to burn the built image to the board. For users developing projects based on the ECK10 core board, we focus on methods and key points for porting this system to their hardware platforms. Through practical porting examples and rootfs customization examples, we enable users to quickly customize a system image tailored to their hardware.

1.1. Software Resources

The ECB10-TB13X runs an operating system based on the Linux 5.15 kernel. The development board comes with the cross-compilation toolchain required for embedded Linux system development, along with TF-A source code, Optee-OS source code, U-boot source code, the source code for the Linux kernel and various driver modules, as well as various development and debugging tools for both Windows and Linux desktop environments, and application development examples.

2. How to burn the system image

Use the stm 32cubeprog rammer to burn to the emmc memory on the E CB10 board. Alternatively, you can create an SD card bootloader to boot the E CB10 from an SD card .

2.1. Programming using STM32CubeProg

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 serial port software mobaexterm and STM32CubeProgrammer software is explained in Chapter 3.5 .

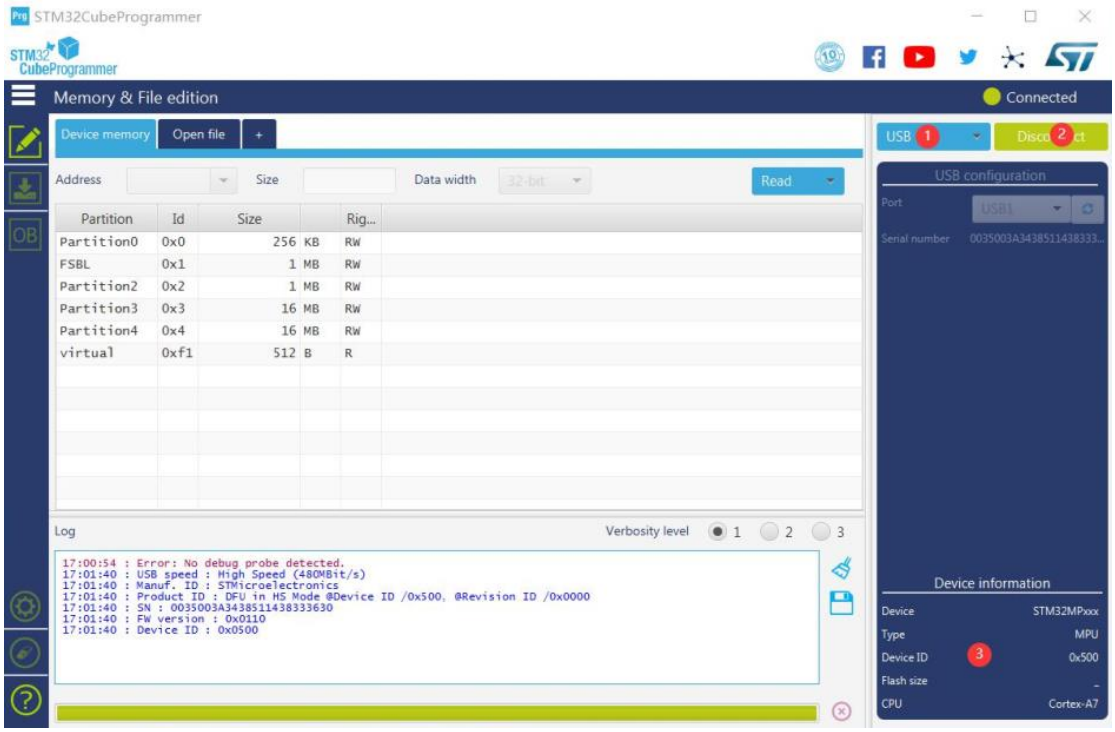
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.1.2. Setting up the hardware and programmer

After having the above software, we connect the RX , TX, GND of the CH340 serial cable to the TX , RX, GND on the board , and then connect to the Type-C interface. Here, be careful not to occupy the USB port by the virtual machine. Finally, we connect the power cord.

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

Open the STM32CubeProgrammer software, which has already been installed on your Windows platform. If the software is not installed, refer to Section 3.5 . Select the USB programming method. Click the Connect button to connect. Verify that the development board information is displayed normally, as shown below. If the log display window in the lower left corner displays some device information, then the E CB10 is connected properly .



2.1.3. Select Profile

ECB10-PGD has four configurations of core boards, namely 1 31-D2 , 1 31-D2E8 , 1 35-D2 and 1 35-D5E8 .

1 31-D2 : cpu: 1 31 memory: 2 56M emmc: no (can only boot from sd card)

1 31-D2E8 : cpu: 1 31 memory: 2 56M emmc: 8 G (boot from sd card or emmc)

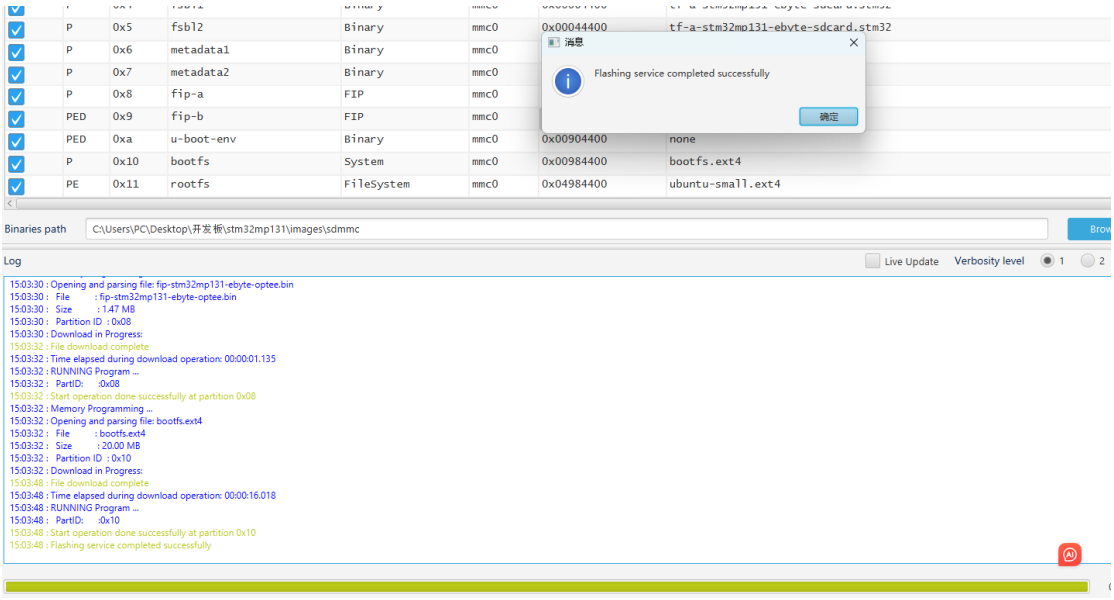
1 35-D2 : cpu: 1 35 memory: 2 56M emmc: no (can only boot from sd card)

1 35-D5E8 : cpu: 1 35 memory: 512M emmc: 8 G (boot from sd card or emmc)

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

Let's take 135-D5E8 as an example. The Cube Program burning tool can only download to the flash on the board, so please use 135-D5E8 with eMMC to burn.

Select 02_Images/ FlashLayout_emmc_ebyte-stm32mp135-D5E8-optee.tsv and click Download to start burning.



After the burning is completed.

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

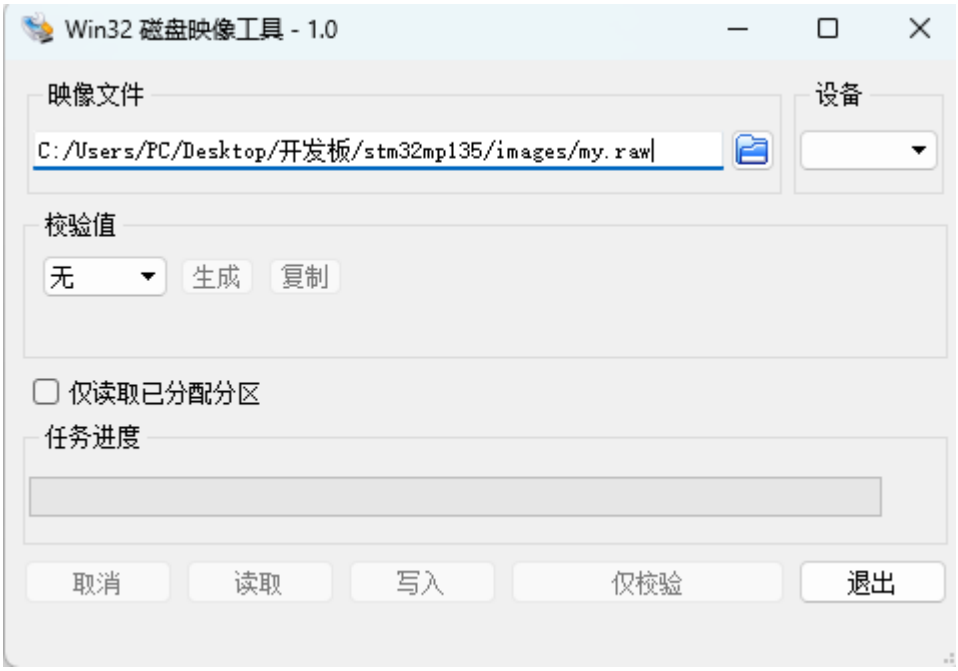
Open Win32DiskImager-1.0.0-binary

Select the image file FlashLayout_sdcard_ebyte-stm32mp135-D5E8-optee.raw .

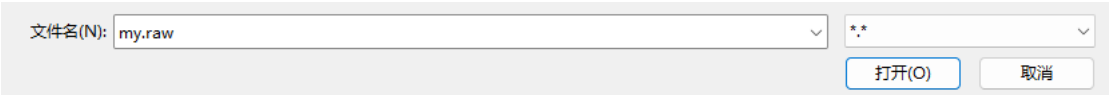
135-D5E8 model is provided here for burning SD cards. If you need images for other models, copy the s tm32mp13x-PGD.tar.gz file from Sources to the virtual machine, unzip it, and go to the FIP_artifacts folder. Then, use the image creation script and the corresponding tsv file to create the image. An example of creating an SD card image for the stm 32mp 131 -D2 is shown below.

```
# cd FIP_artifacts/script/
# ./create_sdcard_from_flashlayout_13x.sh ../FlashLayout_sdcard_ebyte-stm32mp131-D2-optee.tsv
```

You will get FlashLayout_sdcard_ebyte-stm32mp131-D2-optee.raw . Then you can burn it according to the subsequent methods.



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



Click Write to start burning and wait for writing to complete, which takes about 1 minute. 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 boot the image. You can use the serial port software to observe the boot process.

2.3.Modify the startup configuration

Because different core boards support different hardware features and memory capacities, you need to select the device tree file to load based on the core board model. If you need to modify the default configuration, you need to modify the configuration file in bootfs .ext4 . You can modify bootfs .ext4 before burning (recommended, more secure) or after starting the

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

development board.

```
Hit any key to stop autoboot:  0
Boot over mmc0!
switch to partitions #0, OK
mmc0 is current device
Scanning mmc 0:8...
Found /mmc0_extlinux/extlinux.conf
Retrieving file: /mmc0_extlinux/extlinux.conf
983 bytes read in 11 ms (86.9 KiB/s)
Retrieving file: /splash_landscape.bmp
7802 bytes read in 12 ms (634.8 KiB/s)
Select the boot mode
1:      ebyte-stm32mp135-D5E8
2:      ebyte-stm32mp135-D2
3:      ebyte-stm32mp131-D2E8
4:      ebyte-stm32mp131-D2
Enter choice: 1:        ebyte-stm32mp135-D5E8
```

MMC 0 corresponds to SD card , and MMC 1 corresponds to eMMC.

If you want to modify the data after booting the development board, you need to directly access the mounted /media/mmcblk0p8/ and then make the modification.

```
#cd /media/mmcblk0p8/
#ls
boot.scr.uimg                ebyte-stm32mp135-D5E8.dtb   splash_landscape.bmp
ebyte-stm32mp131-D2.dtb      lost+found                  splash_portrait.bmp
ebyte-stm32mp131-D2E8.dtb    mmc0_extlinux               st-image-resize-initrd
ebyte-stm32mp135-D2.dtb      mmc1_extlinux               uImage
```

Here, mmc0_extlinux and mmc1_extlinux represent the SD card and E MMC respectively . Since we boot from the SD card, we need to modify the extlinux.conf configuration file under mmc0_extlinux .

```
# vi mmc0_extlinux/extlinux.conf
```

D EFALT to any of the L ABEL below .

If you need to modify it before burning, follow the above method to mount the bootfs .ext4 in the provided Image , and then modify the corresponding position. Finally, use CubeProgrammer to burn it to eMMC, or create a raw file and burn it to an SD card.

3. Development environment preparation

This section describes how to build a development environment for the STM32MP1XX

series processor platform. By reading this chapter, you will learn how to install and use relevant hardware tools and software development and debugging tools. This will help you quickly build a development environment and prepare for subsequent development and debugging.

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

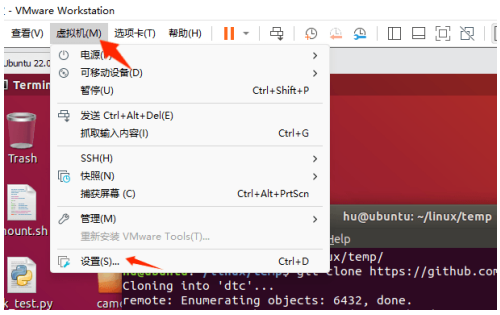
Virtual machine operating system: Ubuntu 18.04 64bit . In the image provided by Ebyte, the path is: 0 4 _ Tools → ubuntu-18.04.5-desktop-amd64.iso.

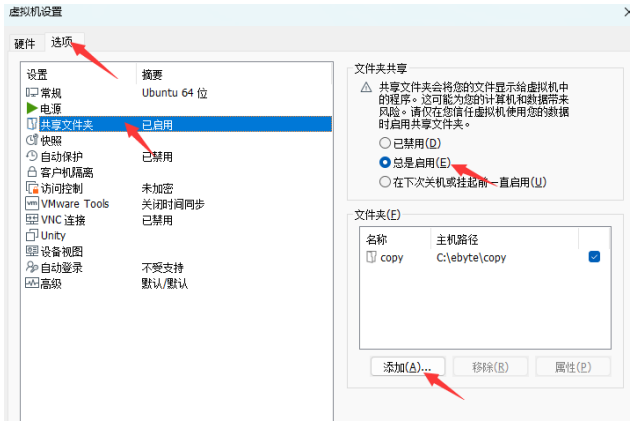
You can follow some online tutorials to build a Ubuntu virtual machine using VMware and this image, but this article won't cover that. Once the virtual machine is installed, you can configure Ubuntu using the following configuration instructions.

The following sections describe the installation of some file transfer tools. If you don't need these tools, you can skip directly to Section 2.6, Configuring Ubuntu to Compile the SDK . If you need to install other file transfer tools, refer to the following sections. To transfer files between a Ubuntu virtual machine and Windows, you can use VMware shared directories (see Section 2.2). To transfer files between a Ubuntu virtual machine and a development board, you can use NFS shared directories (see Section 2.3). To remotely log in to Ubuntu using a serial port tool, you need to enable the SSH service (see Section 2.4). The following sections only verify the Ubuntu - 1804 system .

3.1.Vmware shared directory

By setting up a shared directory in VMware, you can enable the virtual machine and Windows to access the same folder, thereby facilitating file transfer.





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

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



This allows you to easily transfer files between Windows and Ubuntu virtual machines. It's best not to compile files in this folder; copy them to another location in the virtual machine before compiling. If you can't find this folder after restarting the virtual machine, you'll need to manually remount it.

```
# 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.

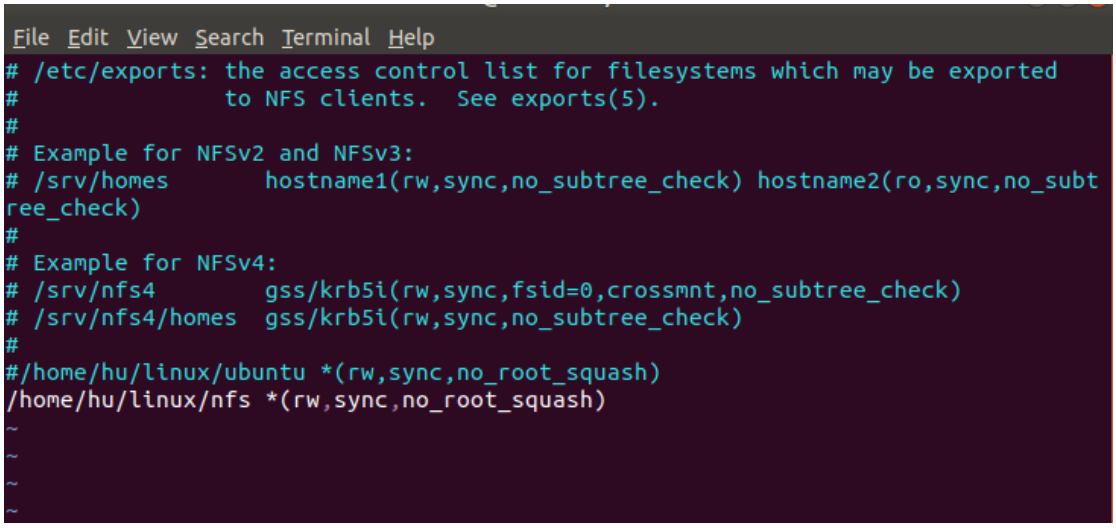
```
# sudo mkdir -p /home/hu/linux/nfs
```

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:

```
# 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, so you can use the default settings. You can now use SSH to remotely log in to the virtual machine. Make sure the host can ping the development board.

3.4.FTP service

TFTP service of Ubuntu , 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 installed automatically. 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 above two lines. If there is, cancel it. When completed, it will look like the picture below:

```
27 # Uncomment this to allow local users to log in.
28 local_enable=YES
29 #
30 # Uncomment this to enable any form of FTP write command.
31 write_enable=YES
```

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

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

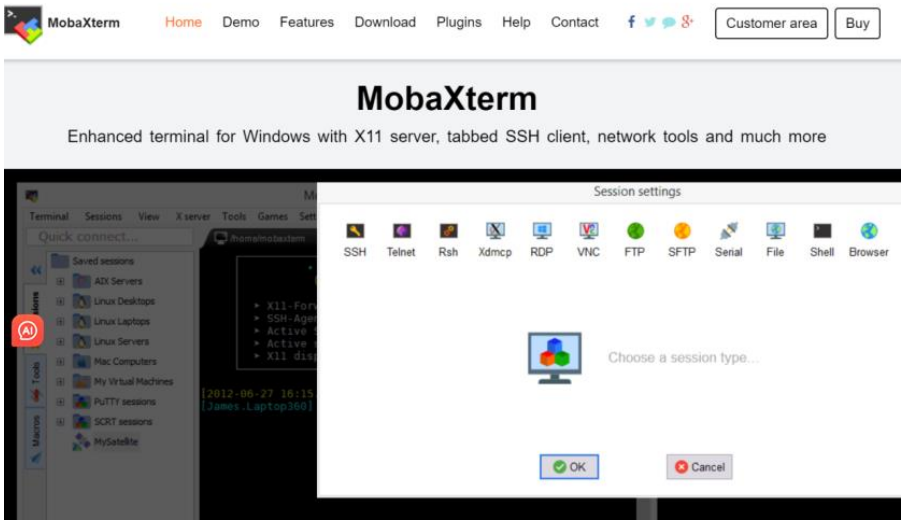
3.5.Windows installation software development tools

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

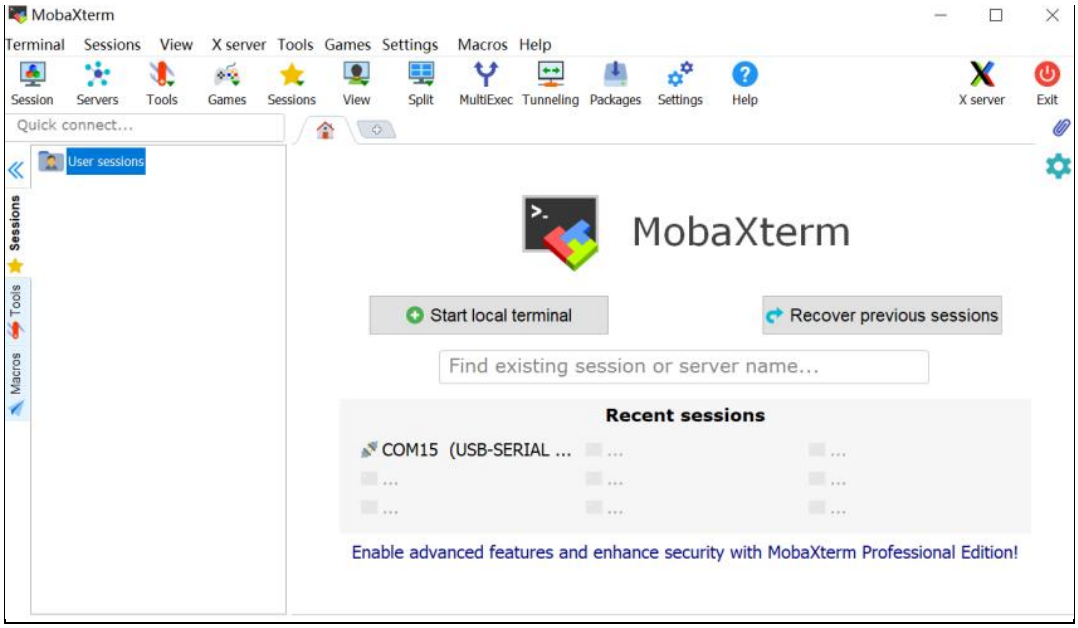
MobaXterm is a terminal software that is powerful and free. It is recommended to use this software as terminal debugging software. MobaXterm software can be downloaded from its official website at https://mobaxterm.mobatek.net/ , as shown in the figure:



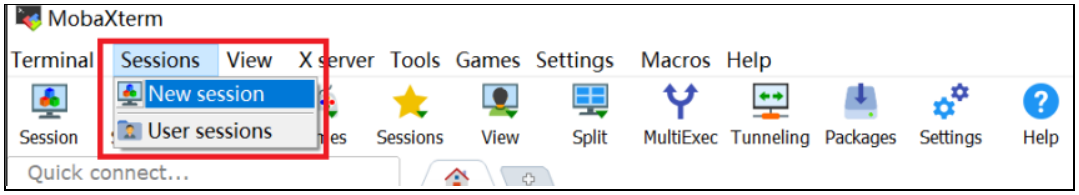
The installation package has been placed on the development board in the following path: 04_Tools -> MobaXterm_ Portable.zip . Open this compressed package.

3.5.1.2. Software Usage

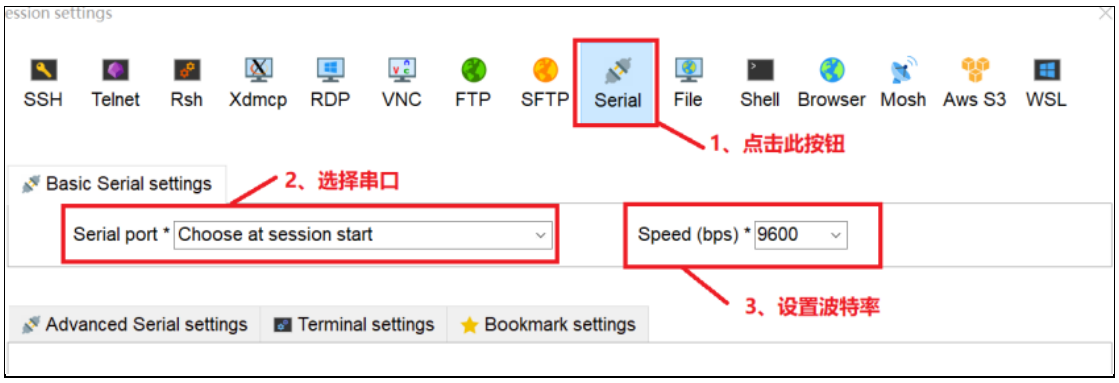
Double-click the MobXterm icon to open the software. The software interface is as shown below:



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 terminal. Click the "Serial" button in the figure to open the serial port setting interface, as shown in the figure:



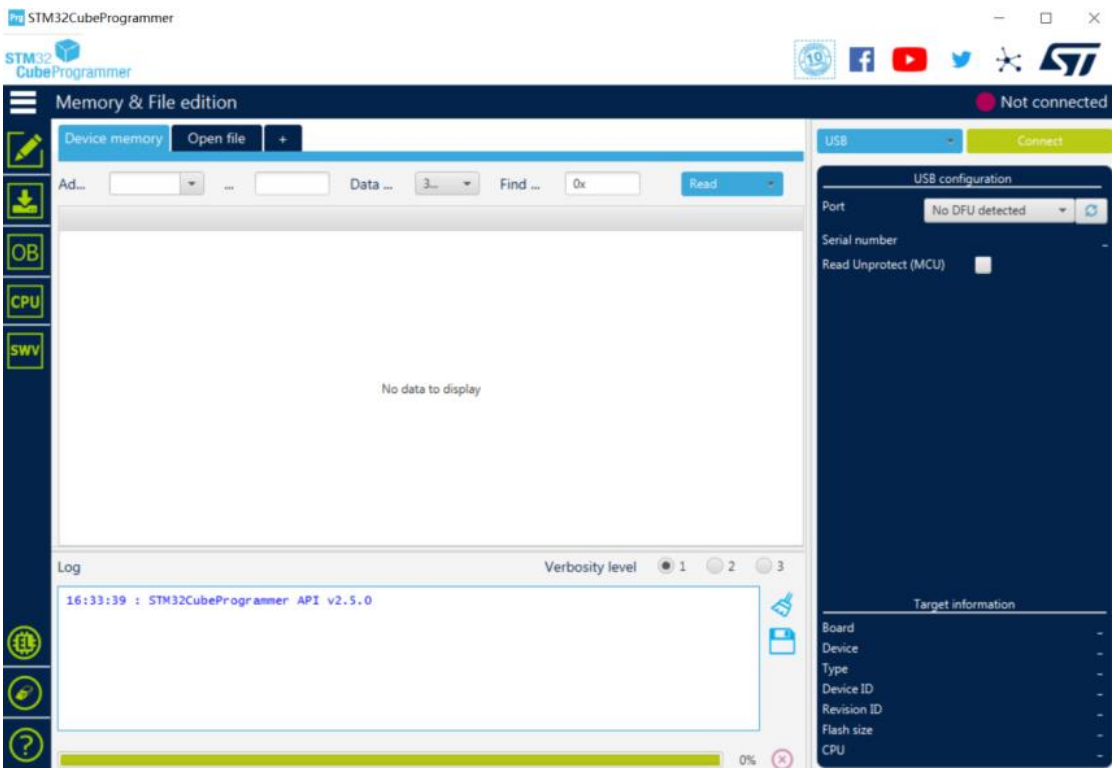
After opening the serial port setting window, first select the serial port number to be set. Therefore, you must first connect the development board to the computer with a serial cable, and then set the baud rate to 115200 (set according to your actual needs).

3.5.2. STM32CubeProg

ST's highly integrated STM32CubeProgrammer programming tool can partition unpartitioned storage devices during the flashing process. Once partitioned, individual partitions can be updated using pre-compiled binary files. Users can choose the appropriate version based on their needs. The download address is: https://www.st.com/zh/development-tools/stm32cubeprog.html . The path is 04_Tools → en.stm32cubeprog_v2-5-0.zip . This software is primarily used for flashing development boards via 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

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



3.6.U Ubuntu configuration development environment

3.6.1. Install the cross-compilation toolchain

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

The cross compiler path provided is 04_Tools→ SDK-x86_64-stm32mp1-openstlinux.tar.gz .

Put the cross compiler into the virtual machine and decompress it

tar -xvzf SDK-x86_64-stm32mp1-openstlinux.tar.gz

Wait for the decompression to complete. After the decompression is completed, a stm32mp1-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23 folder will be generated.

Enter the directory to install the tool chain

cd stm32mp1-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/sdk/

./st-image-weston-openstlinux-weston-stm32mp1-x86_64-toolchain-4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23.s h

Follow the prompts to press enter and enter y, and finally enter the root password to install it

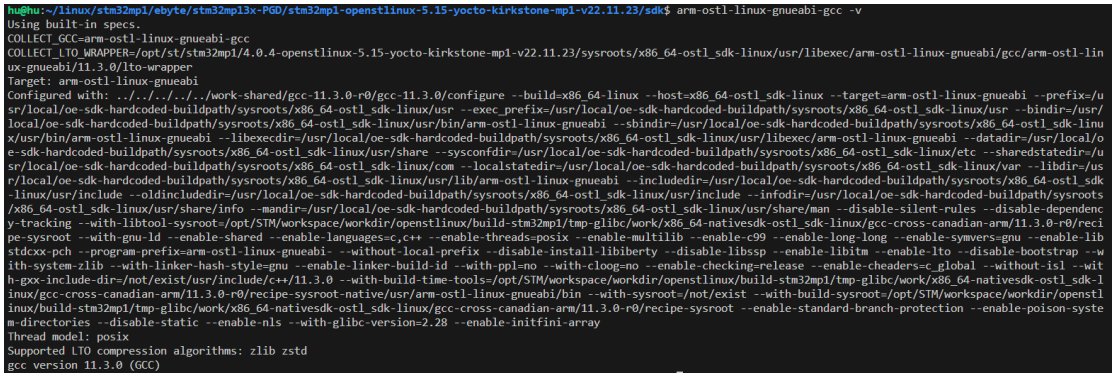
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



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. If the following version information appears, it means the installation is successful.

```
#source  /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# arm-ostl-linux-gnueabi-gcc -v
```



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 already

downloaded.

04_Tools→ dtc.tar.gz .

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

```
# 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 provided a comprehensive overview of the process for burning an image to a development board. Because many pins on the ECB10 core board are multi-functional, there will always be some differences between a baseboard designed based on the ECK10 core board and the ECB10 in a real-world project . These differences might include removing the display, adding more GPIO, adding more serial ports, or perhaps expanding peripherals via SPI, I2C, USB, and so on. This chapter describes the detailed process of developing and customizing a system from the perspective of a system developer . If the user has no special requirements, they only need to modify the kernel; the remaining images can be used with the pre-compiled images compiled by Ebyte.

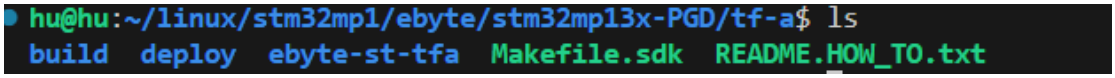
4.1.TF-A

When booting using the Trusted Boot Chain (for more information about the Boot Chain, see https://wiki.st.com/stm32mpu/wiki/Boot_chain_overview), the TFA functions as the FSBL (First Stage Boot Loader), primarily initializing core resources such as DDR and the clock. Hardware

designed based on the ECB10-TB13X core board generally does not require modification of the TFA. However, if a clock needs to be modified or early control of a pin is required, the following method can be used for porting.

4.1.1. Get the source code

After obtaining the source code, decompress it. The source code path is 02_Source/tf-a.tar.gz . The decompression result is as shown in the figure.



4.1.2. Fip tool installation

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, short for Firmware Image Package, is a packaging format used by TF-A, which packages the bootloader image and other payload files into a single binary file. ST's ecosystem development kit currently recommends using FIP to boot the STM32MP1 platform. Only the TF-A image (with STM32 header structure information) is loaded by the ROM code, while other binary files such as OP-TEE, U-Boot, and their respective device tree files are packaged as FIP images. The TF-A runtime can load and authenticate FIP images.

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.

The TF-A source code provides a tool called fiptool that implements 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 ebyte-st-tfa/tools/fiptool / // Enter your TF-A source code directory
# cd tools/fiptool // Enter the fiptool source code
# make // Compile the fiptool source code
```

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

```
# 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.
```

4.1.3. stm32wrapper4dbg tool installation

When compiling TF-A or Uboot, you need the stm32wrapper4dbg tool; otherwise, compilation errors will occur. ST provides the source code for this tool, which you need to compile and install in Ubuntu. The source code can be downloaded from: https://github.com/STMicroelectronics/stm32wrapper4dbg . We have already downloaded it and installed it on the development board. The path is: 04_Tools → stm32wrapper4dbg-master.zip . Copy the compressed source code package to Ubuntu and decompress it using the following command:

```
# 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
```

```
hu@ubuntu:~/linux/temp/dtc$ stm32wrapper4dbg -s
stm32wrapper4dbg: option requires an argument -- 's'
Usage: stm32wrapper4dbg -s srcfile -d destfile [-b] [-f]
  Add a debug wrapper to a stm32 image.
  If "-b" is not specified, the wrapper would be placed
  after the last byte of the image.

Options:
  -s srcfile   input image in stm32 file format
  -d destfile  output image in stm32 file format
  -b           place the wrapper before the image
  -f           force re-adding the wrapper
```

4.1.4. Compile

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

```
23
24  # Init default config settings
25  TF_A_DEVICETREE_optee ?= ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
26  TF_A_EXTRA_OPTFLAGS_optee ?= AARCH32_SP=optee
27  TF_A_BINARY_optee ?= tf-a
28  TF_A_MAKE_TARGET_optee ?= dtbs
29
30  # Init default config settings
31  TF_A_DEVICETREE_emmc ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
32  TF_A_EXTRA_OPTFLAGS_emmc ?= STM32MP_EMMC=1 PSA_FWU_SUPPORT=1
33  TF_A_BINARY_emmc ?= tf-a
34  TF_A_MAKE_TARGET_emmc ?= all
35
36  # Init default config settings
37  TF_A_DEVICETREE_sdcard ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
38  TF_A_EXTRA_OPTFLAGS_sdcard ?= STM32MP_SDMMC=1 PSA_FWU_SUPPORT=1
39  TF_A_BINARY_sdcard ?= tf-a
40  TF_A_MAKE_TARGET_sdcard ?= all
41
42  # Init default config settings
43  TF_A_DEVICETREE_uart ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
44  TF_A_EXTRA_OPTFLAGS_uart ?= STM32MP_UART_PROGRAMMER=1
45  TF_A_BINARY_uart ?= tf-a
46  TF_A_MAKE_TARGET_uart ?= all
47
48  # Init default config settings
49  TF_A_DEVICETREE_usb ?=  ebyte-stm32mp135-D5E8 ebyte-stm32mp135-D2
50  TF_A_EXTRA_OPTFLAGS_usb ?= STM32MP_USB_PROGRAMMER=1
```

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.

```
# tar -xvzf tf-a.tar.gz

# cd tf-a/ebyte-st-tfa/

# chmod +x ./tools/stm32image/stm32image

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

#  source  /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environm

ent-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi

# ./build.sh
```

```
#!/bin/bash
rm ../build ../deploy -rf      You, 上个月 • init
make -f ../Makefile.sdk all -j 8
cp ../deploy/tf-a-ebyte-stm32mp13* ../../FIP_artifacts/arm-trusted-firmware/
cp ../deploy/metadata.bin ../../FIP_artifacts/arm-trusted-firmware/
cp ../deploy/fwconfig/ebyte-stm32mp13* ../../FIP_artifacts/arm-trusted-firmware/fwconfig/
cp ../deploy/debug/* ../../FIP_artifacts/arm-trusted-firmware/debug/
```

After the compilation is complete, two directories, "build" and "deploy," are created in the upper directory, tf-a, as shown below. Compiled files are generated in the deploy directory of the upper directory.

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/tf-a/deploy$ ls -l
total 708
drwxrwxr-x 2 hu hu  4096  9月  8 13:29 debug
drwxrwxr-x 2 hu hu  4096  9月  8 13:29 fwconfig
-rw-rw-r-- 1 hu hu    96  9月  8 13:29 metadata.bin
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-emmc.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-sdcard.stm32
-rw-rw-r-- 1 hu hu 84723  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-uart.stm32
-rw-rw-r-- 1 hu hu 88819  9月  8 13:29 tf-a-ebyte-stm32mp135-D2-usb.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-emmc.stm32
-rw-rw-r-- 1 hu hu 88843  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-sdcard.stm32
-rw-rw-r-- 1 hu hu 84723  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-uart.stm32
-rw-rw-r-- 1 hu hu 88819  9月  8 13:29 tf-a-ebyte-stm32mp135-D5E8-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.

The source code directory is in 02_Source/ optee.tar.gz . After decompression, it is as shown in the figure below.

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/optee$ ls
build  deploy  ebyte-st-optee  Makefile.sdk  README.HOW_TO.txt
```

As shown in the image above, similar to the TF-A source code, after decompression, a

Makefile.sdk file and the OP-TEE source code directory appear. The Makefile.sdk file primarily defines compilation properties, such as the cross-compiler to use and compilation options. The Makefile.sdk file ultimately calls the Makefile within OP-TEE to compile the OP-TEE source code.

```
# 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 ?=   ebyte-stm32mp135-D5E8
#CFG_EMBED_DTB_SOURCE_FILE ?=   ebyte-stm32mp135-D2

# Remove default variables
LDFLAGS =
CFLAGS =
CPPFLAGS =
# Define default make options
```

When compiling the factory source code, users do not need to modify anything. The OPTEE source code device tree path adapted to the development board's hardware resources is core/arch/arm/dts/, and the device tree files are ebyte-stm32mp135-D5E8 and ebyte-stm32mp135-D2 .

CFG_EMBED_DTB_SOURCE_FILE according to the actual core board model they have .

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

```
# cd ebyte-st-optee
# source /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

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.

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

The files in the deploy directory are crucial. The Makefile.sdk script processes the binary files generated in the build directory and copies them to the deploy directory. They contain three files: tee-header_v2-ebyte-stm32mp135-D5E8.bin , tee-pageable_v2-ebyte-stm32mp135-D5E8.bin , and tee-pager_v2-ebyte-stm32mp135-D5E8.bin . These three files are the final product of compiling the factory-installed OP-TEE source code.

the ebyte-stm32mp135-D2 configuration is selected , tee-header_v2-ebyte-stm32mp135-D2.bin , tee-pageable_v2-ebyte-stm32mp135-D2.bin , and tee-pager_v2-ebyte-stm32mp135-D2.bin are generated .

The OP-TEE source code has been compiled before delivery. The following describes how to compile the U-Boot source code.

4.3. U-boot

Uboot, short for Universal Boot Loader, is open source software licensed under the GPL. It consists of bare-metal code and can be considered a comprehensive bare-metal program. Uboot now supports advanced features such as LCD screens, networking, and USB. The official Uboot website 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 code package 02_Source/uboot.tar.gz to the uboot directory. Then decompress it, and the decompression result will be as shown in the figure.



As can be seen from the figure above, similar to the previous source code, a Makefile.sdk file and the U-Boot source code ebyte -st-uboot directory appear after decompression.

The Makefile.sdk file here mainly defines some compilation properties, such as U-Boot default source code configuration, specifying the compilation device tree name, some compilation options, etc., as shown in the figure 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 += stm32mp13_defconfig
#UBOOT_BINARY_stm32mp13_defconfig ?= u-boot.dtb
#UBOOT_DEVICETREE_stm32mp13_defconfig ?=
# Init default config settings
UBOOT_CONFIGS += trusted
#UBOOT_DEFCONFIG_trusted += ebyte_stm32mp135_D2_defconfig
#UBOOT_BINARY_ebyte_stm32mp135_D2_defconfig ?= u-boot.dtb
#UBOOT_DEVICETREE_ebyte_stm32mp135_D2_defconfig ?= ebyte-stm32mp135-D2

UBOOT_DEFCONFIG_trusted += ebyte_stm32mp135_D5E8_defconfig
UBOOT_BINARY_ebyte_stm32mp135_D5E8_defconfig ?= u-boot.dtb
UBOOT_DEVICETREE_ebyte_stm32mp135_D5E8_defconfig ?= ebyte-stm32mp135-D5E8
```

When compiling the factory source code, users do not need to modify anything.

135-D2 selects the three lines of configuration corresponding to ebyte_stm32mp135_D2_defconfig

1 35-D5E8 selects the three lines of configuration corresponding to ebyte_stm32mp135_D5E8_defconfig

```
# cd ebyte-st-uboot
# source /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

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.

The build directory contains all the files generated during the compilation process. Most of these files are not required by the user. The files in the deploy directory are crucial. The Makefile.sdk script processes the binary files in the build directory and copies them to the deploy

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

directory. It contains two files: u-boot-ebyte-stm32mp135-D5E8.dtb and u-boot-nodtb-stm32mp13.bin . These two files are the final products of compiling the factory-installed U-Boot source code, representing the U-Boot firmware for the ECB10-135D5E8 core board. If the D2 configuration is selected , u-boot-ebyte-stm32mp135-D2.dtb and u-boot-nodtb-stm32mp13.bin are generated .

4.3.2. Set U boot kernel startup mode

U- boot. These mainly affect the kernel startup method. If you do not have special requirements, you generally do not need to modify the commands in this section and can skip it.

4.3.2.1. bootcmd

The bootcmd variable stores the default uboot commands. After the uboot countdown ends, the commands in bootcmd are executed. These commands are generally used to boot the Linux kernel, for example, reading the Linux kernel image file and device tree file from EMMC or NAND Flash into DRAM and then booting the Linux kernel. You can set the value of the bootcmd environment variable by entering the command line after uboot boots.

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

The console is used to set the Linux terminal (or console). Specifically, what device is used to interact with Linux? Is it a serial port or an LCD screen? If it's a serial port, it should be serial port number, etc. Generally, setting a serial port as the Linux terminal allows us to interact with Linux on our computer through MobaXterm. Here, console is set to ttySTM0 because, after Linux boots, the STM32MP1's serial port 4 is represented by the device file /dev/ttySTM0 in Linux. In Linux, everything is a file. ",115200" follows ttySTM0 to set the serial port's baud rate. "console=ttySTM0,115200" essentially sets ttySTM0 (serial port 4) as the Linux terminal and sets

the baud rate to 115200. Using the debug serial terminal, after the development board boots into uboot, control the boot method by setting bootargs and bootcmd in uboot. After setting these settings, you need to save the environment variables using the command. 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" indicates that the root file system is stored in partition 3 of the mmcblk2 device. After the STM32MP1 core board boots Linux, files such as /dev/mmcblk1, /dev/mmcblk2, /dev/mmcblk1p1, /dev/mmcblk1p2, /dev/mmcblk2p1, /dev/mmcblk2p2, and /dev/mmcblk2p3 will exist. /dev/mmcblkx (x=0-n) represents the MMC device, and /dev/mmcblkxpy (x=0-n, y=1-n) represents partition y of MMC device x. On the STM32MP1 development board, /dev/mmcblk2 represents the EMMC, and /dev/mmcblk2p3 represents partition 3 of the EMMC.

"rootwait rw" follows "root". "rootwait" indicates that the root file system should be mounted after the MMC device is initialized. Otherwise, mounting the root file system before the MMC device is initialized will result in an error. "rw" indicates that the root file system is readable and writable. Without "rw", write operations may be disabled in the root file system, and only read operations may be possible.

This option , rootfstype , is typically used in conjunction with root to specify the root file system type. If the root file system is in ext format, this option is irrelevant. If the root file system is yaffs , jffs , or ubifs , this option must be set to specify the root file system type.

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
```

The following introduces three different startup methods.

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. Boot 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 and configure the NFS and TFTP services on the server (virtual machine). You also need to configure the IP address , MAC address , gateway, netmask, 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.

include/configs/stm32mp13_st_common.h in the uboot source code: for STMicroelectronics boards using the 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 start uboot using the method in the above

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

section and then manually set the startup mode.

```
#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"
```

As shown in the above code, if the boot is from SD card (MMC), the device tree, kernel and root file system are loaded from the SD card partition. If the boot is from NAND, the kernel and root file system are loaded from NAND.

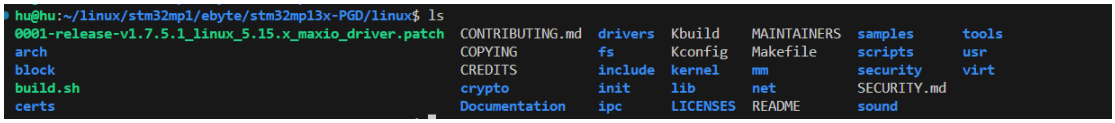
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

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.Kernel

4.4.1. Get the source code and compile

Create a directory named "linux" in the ubuntu directory, and then copy the factory linux source package 02_Source /linux.tar.gz to the linux directory.



When compiling the factory source code, users do not need to modify anything. The uboot source code device tree path for the ECK10-135A5M1G-I development board hardware resources is arch/arm / boot/dts/, and the device tree files are stm32mp135-ebyte.dts and stm32mp13-pinctrl-ebyte.dtsi . To facilitate compilation, a compilation script is provided; you can directly run the script for the first compilation. The script is shown below:

```
#!/bin/bash
rm ../build -rf      You, 1秒钟前 • Uncommitted changes
make ARCH=arm O="$PWD/../build" ebyte_stm32mp135_D5E8_defconfig
# make ARCH=arm O="$PWD/../build" menuconfig
make ARCH=arm uImage vmlinux dtbs LOADADDR=0xC2000040 O="$PWD/../build" -j 8
make ARCH=arm modules O="$PWD/../build" -j 8
make ARCH=arm INSTALL_MOD_PATH="$PWD/../build/install_artifact" modules_install O="$PWD/../build" -j 8
mkdir -p $PWD/../build/install_artifact/boot/
cp $PWD/../build/arch/arm/boot/uImage $PWD/../build/install_artifact/boot/
cp $PWD/../build/arch/arm/boot/dts/ebyte-stm32mp13* $PWD/../build/install_artifact/boot/
sudo cp ../build/install_artifact/boot/uImage /mnt/bootfs
sudo cp ../build/install_artifact/boot/ebyte-stm32mp13* /mnt/bootfs
```

We use the provided bootfs .ext4 to mount it to / mnt/bootfs , and the compilation script will automatically copy the compiled device tree and kernel image to / mnt/bootfs .

```
# cd FIP_artifacts/
# mkdir /mnt/bootfs
# sudo mount bootfs.ext4 /mnt/bootfs
# cd ../linux/
#  source  /opt/st/stm32mp1/4.0.4-openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
# ./build.sh
```

Then execute the ./build.sh compilation script . The first compilation may be slow. After

compilation, the u Image image, ebyte-stm32mp135-D5E8.dtb, and ebyte-stm32mp135-D2.dtb device tree files will be generated in the / mnt / bootfs folder .

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/FIP_artifacts$ ls /mnt/bootfs
boot.scr.uimg            ebyte-stm32mp135-D5E8.dtb  mmc0_extlinux  splash_landscape.bmp  st-image-resize-initrd
ebyte-stm32mp135-D2.dtb  lost+found                 mmc1_extlinux  splash_portrait.bmp   uImage
```

Finally, cancel the mount

```
# sudo umount /mnt/bootfs
```

4.5.summary

Through the above steps, we get the image files of all source code parts, as shown in the figure below.

```
hu@hu:~/linux/stm32mp1/ebyte/stm32mp13x-PGD/FIP_artifacts$ ls
arm-trusted-firmware                          FlashLayout_sdcard_ebyte-stm32mp135-D5E8-optee.how_to_update.txt
bootfs.ext4                                   FlashLayout_sdcard_ebyte-stm32mp135-D5E8-optee.raw
clean.sh                                      FlashLayout_sdcard_ebyte-stm32mp135-D5E8-optee.tsv
fip                                           optee
FlashLayout_emmc_ebyte-stm32mp135-D5E8-optee.tsv  rootfs.ext4
FlashLayout_sdcard_ebyte-stm32mp135-D2-optee.tsv  u-boot
```

5. How to adapt to different hardware platforms

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 within the BSP source code. Generally, there's no need to modify TF-a, optee, or u-boot in the Bootloader section. Users only need to adjust the Linux kernel device tree appropriately based on their actual hardware resources. Here's a list of the device trees in each section of the ECB10-TB13X BSP for user development convenience.

project	Device Tree	illustrate
TF - A	ebyte-stm32mp13x-base.dts	Resource Device Tree
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D2-fw-config.dts ebyte-stm32mp135-D5E8.dts ebyte-stm32mp135-D5E8-fw-config.dts	Core board resource description
	stm32mp13-ddr3-1x2Gb-1066- binF .dtsi	Single chip 256MB DDR3

	stm32mp13-ddr3-1x4Gb-1066- binF .dtsi	Description Single chip 512 MB DDR3 Description
	ebyte-stm32mp13-pinctrl.dtsi	Pin multiplexing description
Optee	ebyte-stm32mp13x-base.dts	Resource Device Tree
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D5E8.dts	Core board resource description
	ebyte-stm32mp13-pinctrl.dtsi	Pin multiplexing description
U - Boot	ebyte-stm32mp13x-base.dts	Resource Device Tree
	ebyte-stm32mp135-D2.dts ebyte-stm32mp135-D2-u-boot.dtsi ebyte-stm32mp135-D5E8.dts ebyte-stm32mp135-D5E8-u-boot.dtsi	Core board resource description
	ebyte-stm32mp13-pinctrl.dtsi	Pin multiplexing description
	ebyte_stm32mp135_D2_defconfig	U boot configuration file
	ebyte_stm32mp135_D5E8_defconfig	
Kernel	ebyte_stm32mp135_D5E8_defconfig	Kernel configuration files
	ebyte-stm32mp13x-base.dts	Resource Device Tree
	ebyte-stm32mp135-D2.dts	ECK10-135D2-C device tree
	ebyte-stm32mp135-D5E8.dts	ECK10-135D5E8-C device tree
	ebyte-stm32mp13-pinctrl.dtsi	Pin multiplexing

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 on-chip and off-chip device information using a unique syntax format. It is passed to the kernel by the Bootloader, which parses it to create a dev structure associated with the driver for use by the driver code.

Under the kernel source code, you can find a large number of platform device trees in the arch/arm/boot/dts directory. For example, if you need a device tree for the ECK10-135A5M1G-I ,

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

you can add a custom device tree to the current directory, such as ebyte-stm32mp135a - xxx.dts. This is 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 have compiled the ECB10-TB13X core board related resources into stm32mp135-base.dts. Other extended interfaces and devices can reference them as shown below (for reference only):

```
/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;
        };
    };

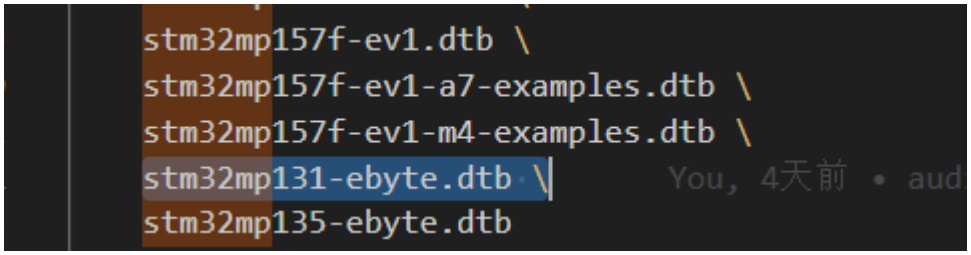
```

After the user adds a new device tree source file, he needs 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) += \

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



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, and application implementation. This section does not specifically analyze the development process of each part, but instead uses examples to explain the control implementation of functional pins.

5.3.1. G PIO pin configuration method

GPIO (General-purpose input/output) is a crucial resource in embedded devices. It can output high or low levels or read the pin's status—high or low. The STM32MP1 incorporates a large number of peripheral controllers. These peripheral controllers typically interact with external devices by controlling GPIOs. Alternate function multiplexing of GPIOs by peripheral controllers allows them to provide more complex functionality. For example, users can use GPIOs to exchange data with external hardware (such as UARTs), control hardware (such as LEDs and buzzers), and read hardware status signals (such as interrupts). Therefore, GPIOs are widely used.

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

5.3.1.1. STM32CubeMX configuration method

ST has added the STM32MPU series CPU to STM32CubeMX. This tool can also be used to configure TF-A, U-boot, and the kernel's GPIO function device tree and peripheral clocks. This section does not focus on its usage; detailed development instructions can be obtained from the official website.

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

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

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

5.3.1.2. Query manual method

The GPIO configuration 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 .

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 pins for the I2C function instance. I2C primarily uses two GPIO pins , I2C_SCL and I2C_SDA . After defining the I2C pin locations and configurations in the pinctrl subsystem , add references to them in the board device tree, as shown below. I2C uses two pin configurations: the default state for normal use and the sleep state for sleep mode. The system automatically switches between these two states.

```
&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;
```

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

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

```
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 your own configured pins

5.3.3.1. Using GPIO pins in U-boot

Uboot can use commands directly in the terminal to control 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 Linux operating system architecture is divided into user state and kernel state (or user space and kernel). User state is the active space for upper-level applications. Application execution relies on resources provided by the kernel, including CPU resources, storage resources, and I/O resources. To enable upper-level applications to access these resources, the kernel must provide an access interface: the system call.

A shell is a special application, commonly known as a command line. Essentially, it's a command interpreter that communicates with system calls and various applications. Using a shell script, a few lines of code can often implement significant functionality. This is because these shell statements typically encapsulate system calls, facilitating user interaction with the system.

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

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

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 example to illustrate the use of GPIO.

Pin numbers are defined as #define PIN_NO(port, line) (((port) - 'A') * 0x10 + (line)), where port is the GPIO port, line is the corresponding pin, and ((port) - 'A') represents ASCII code subtraction. 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 with Linux 4.8, Linux introduced a new GPIO operation method: GPIO character devices. The previous SYSFS method of operating GPIO in the "/sys/class/gpio" directory is no longer recommended. Instead, character devices based on "file descriptors" are used. Each GPIO group has a corresponding gpiochip file in "/dev", for example, "/dev/gpiochip0 corresponds to GPIOA, /dev/gpiochip1 corresponds to GPIOB", and so on.

The libgpiod library implements GPIO-based functions, providing 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 for operating 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 final step in the Linux port. Once the root file system is built, we have a complete, working minimum system. We will then write and test Linux drivers on this minimum system, port some third-party components, and gradually refine it. Ultimately, we'll have a relatively complete operating system with comprehensive functionality and a full range of drivers.

6.1. Buildroot to create a minimal system

6.1.1. Get the source code and compile

Ebyte provides an example of a buildroot file system that can be directly burned. If you need to modify it, copy the buildroot source code ebyte-mp135-sdk-buildroot to the Ubuntu virtual

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

machine and decompress it after copying.

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

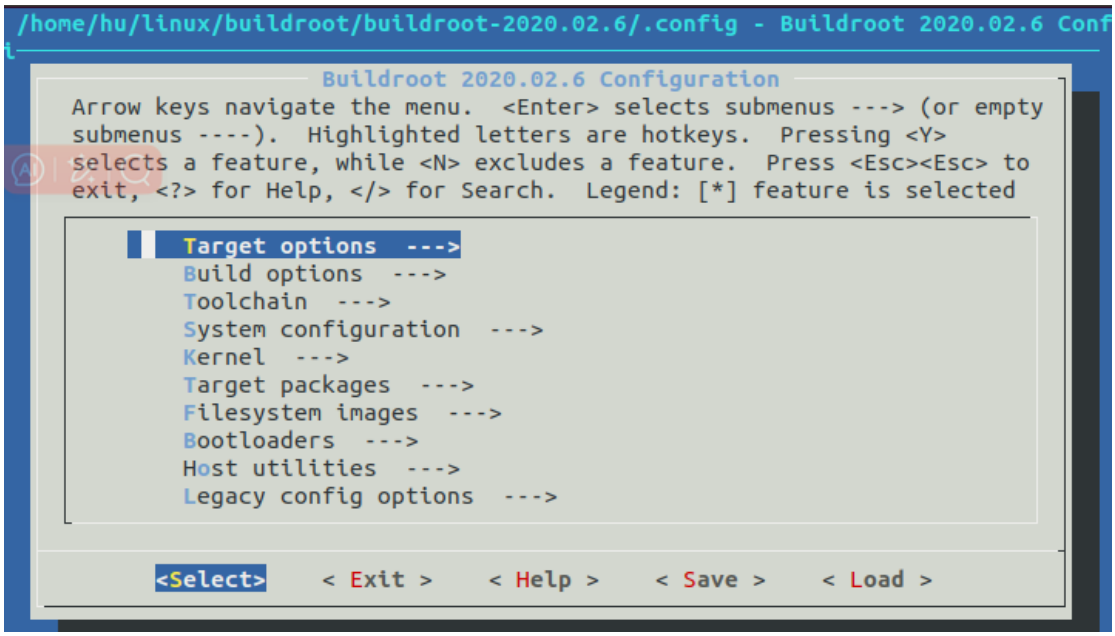
After decompression, it is as shown below

```
hu@hu:~/linux/buildroot/stm32mp135/ebyte-mp135-sdk-buildroot$ ls
arch    boot     Config.in         configs  DEVELOPERS  docs  linux     Makefile.legacy  package  support  toolchain
board   CHANGES  Config.in.legacy  COPYING  dl          fs    Makefile  output           README   system   utils
```

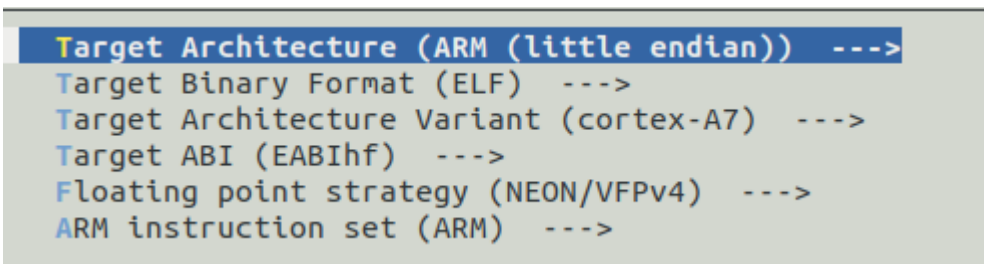
6.1.2. Configure buildroot

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

```
# make menuconfig
```



Configure target options, the configuration results are as follows.



Configure the toolchain. The configuration results are as follows:

```
    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

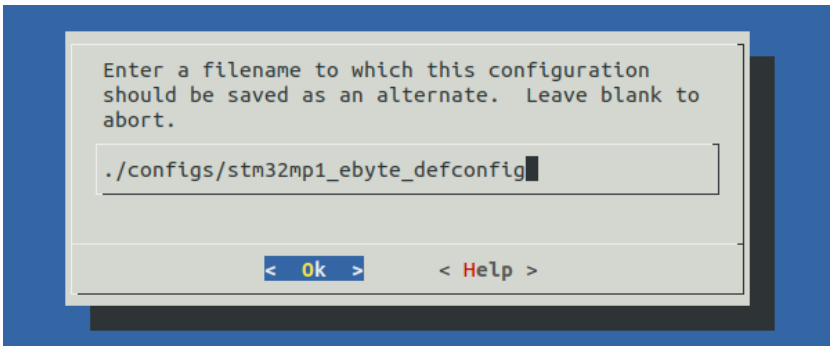
Configure mtd to support UBI interface

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

Save configuration items

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

Just like uboot and kernel, it is best to save the configuration items after configuring buildroot through the graphical interface to prevent them from being deleted after clearing the project. The default configuration items of buildroot are saved in the configs directory. After the configuration is complete, 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

Ebyte provides an example of an Ubuntu file system that can be burned directly.

If customization is required, refer to the following. ubuntu-base is the official minimal Ubuntu file system, including the Debian package manager. The base package size is typically only tens of megabytes and is backed by the entire Ubuntu software repository. Ubuntu software is generally stable. Ubuntu-base allows for on-demand installation of Linux software, offering deep customization and is often used for building embedded rootfs.

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



6.2.2. Prepare 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, one can directly use the network update package

```
# 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 permissions to executable.

```
#!/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

```
# 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
Installing 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
```

Set the root password

#passwd root

Create a username: ebyte

#adduser ebyte

Set the login serial port

#systemctl enable getty@ttySTM0.service

6.2.4. Uninstall the 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. 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 Datasheet

8. Revision Notes

Version	Modifications	Modification time	Prepared by	Proofreading	Approval
V1.0	First Draft	2025-09-08	HSL	WQ	WFX

9. About Us

Technical support: support@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

---

Official hotline:028-61399028

Web: www.cdebyte.com

Address: , Building B5, Mould Industrial Park, 199# Xiqu Ave, High-tech Zone, Chengdu, 611731, Sichuan, China

Chengdu Ebyte Electronic Technology Co.,Ltd.