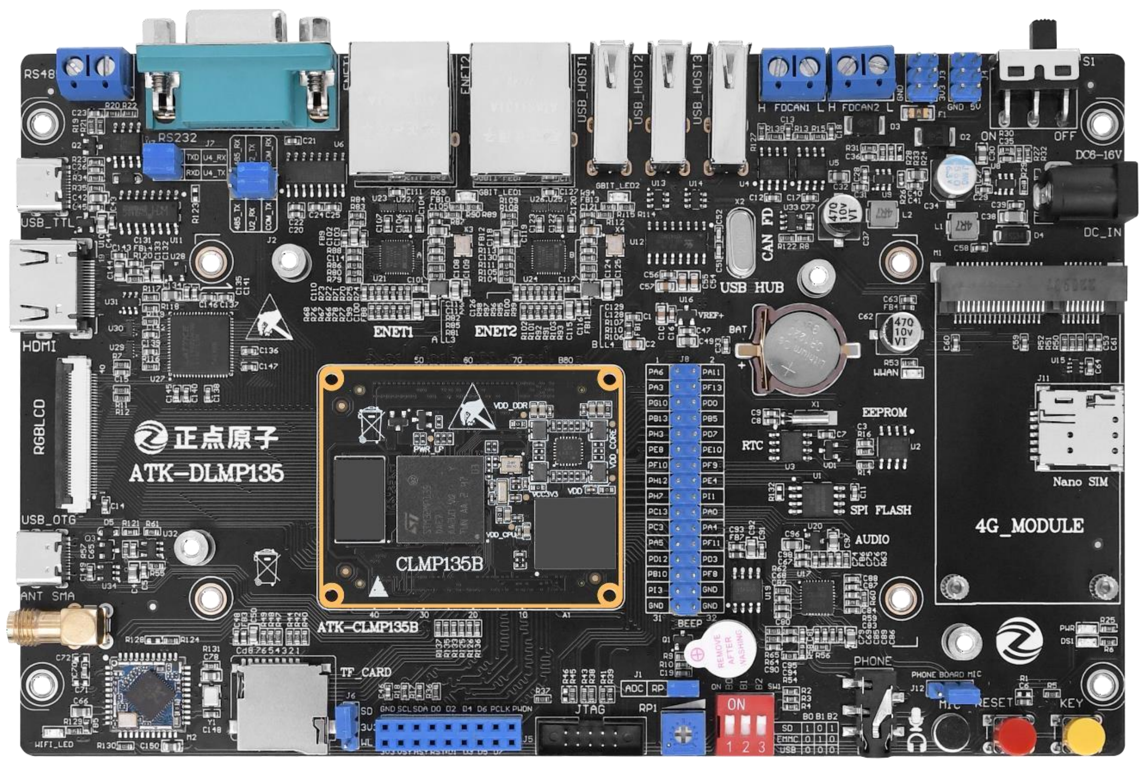
ATK-DLMP135

Factory System Source Code Usage Guide

V1.1



ALIENTEK

1. Shopping:

TMALL：https://zhengdianyuanzi.tmall.com

TAOBAO：https://openedv.taobao.com

2. Download

Address：http://www.openedv.com/docs/index.html

3. FAE

Website	:	www.alientek.com
Forum	:	http://www.openedv.com/forum.php
Videos	:	www.yuanzige.com
Fax	:	+86 - 20 - 36773971
Phone	:	+86 - 20 - 38271790

Disclaimer

Revision History:

Version	Version Update Notes	Responsible person	Proofreading	Date
V1.0	release officially	ALIENTEK	ALIENTEK	2023.04.18
V1.1	Modified the file system compilation, added the make compilation command, and installed some libraries.	ALIENTEK	ALIENTEK	2023.04.24

Catalogue

Brief

This document mainly introduces the compilation process of the factory system source code of the ALIENTEK ATK-DLMP135 development board, guiding users to master the compilation method of the factory source code of this development board, and facilitating subsequent secondary development. This document does not cover the explanation of the compilation principles of each part.

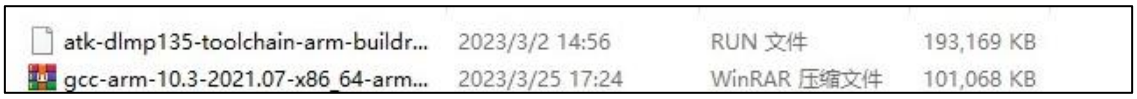
The environment used in this document:

- **Windows 10 64-bit**. It is not recommended to use 32-bit Windows for development. The memory size supported by 32-bit Windows is limited, and the system performance is also limited. This document is introduced based on the 64-bit operating system.
- **Ubuntu 18.04**. Ubuntu recommends using 18.04; otherwise, errors may occur due to different installation environments. Readers are required to know how to use FileZilla or WinSCP to transfer files between Ubuntu and Windows.
- This document is written for the default factory system source code of the ALIENTEK ATK-DLMP135 development board. If users use other source code versions or other versions of cross-compilers, it may cause compilation errors due to differences in source code configuration, compiler supported instructions, etc. Self-resolution is required. The compilation instructions in this document are for reference only. For convenience of development, please use the default factory system source code.

Chapter 1. Install the ARM cross-compilation toolchain

The system source code of the ATK-DLMP135 development board needs to be compiled using the ARM cross-compilation toolchain, and finally generate executable binary files to be burned onto the development board. This section will introduce the installation method of the cross-compilation toolchain.

The ARM cross-compilation toolchain provided by the ATK-DLMP135 development board is located in the network disk data path: STM32MP135 development board →Development board CD A - Basic data→05_tools→01_Cross compiler.



```
atk-dlmp135-toolchain-arm-buildr...  2023/3/2 14:56   RUN 文件          193,169 KB
gcc-arm-10.3-2021.07-x86_64-arm...   2023/3/25 17:24  WinRAR 压缩文件    101,068 KB
```

Figure 1.1-1 ATK-DLMP135 Cross Compilation Toolchain

This toolchain, atk-dlmp135-toolchain-arm-buildroot-linux-gnueabihf-x86_64-xxxxxxxx-v1.x.x.run, was generated by ALIENTEK based on the buildroot file system and is a cross toolchain mainly used for compiling commands and programs related to file systems.

This toolchain, gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz, is a cross toolchain downloaded from the arm Developer website (address), and after testing, it is found that compiling source code (TF-A, OP-TEE, U-Boot and Linux) for the MP135 chip requires a version of GCC 10.3 or above.

1.1 Installation of Cross-Compilation Toolchain for Source Code

Before installation, first copy it to the Ubuntu virtual machine. Please note that the ATK-DLMP135 development board is developed using Ubuntu 18.04. It is recommended that users uniformly use Ubuntu 18.04. The user environment should be consistent with the author's environment to facilitate troubleshooting in the future.

Copy gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz to the user-defined directory of Ubuntu.

Create the directory: /usr/local/arm in Ubuntu. The command is as follows:

```
sudo mkdir /usr/local/arm
```

After the creation is completed, copy the newly copied cross-compiler to the directory /usr/local/arm. In the terminal, use the command "cd" to enter the directory where the cross-compiler is stored. For example, if I copied the cross-compiler to the directory "/home/alientek/135/toolchain" earlier, then use the following command to copy the cross-compiler to /usr/local/arm:

```
sudo cp gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz /usr/local/arm/ -f
```

After the copying is completed, the cross-compilation tools should be decompressed in the /usr/local/arm directory. The decompression command is as follows:

```
sudo tar -vxf gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf.tar.xz
```

Wait for the decompression to complete. After the decompression is finished, a folder named "gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf" will be generated. This folder contains our cross-compiler toolchain.

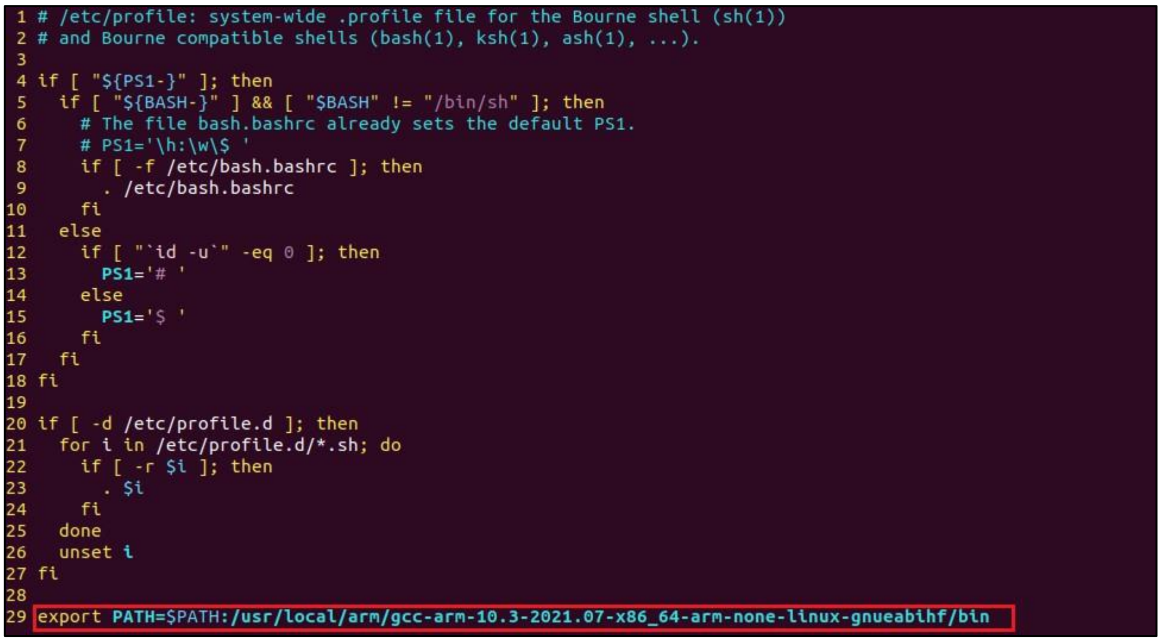
Modify the environment variables. Open the /etc/profile file and use the following command:

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

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

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

The /etc/profile file after the addition is shown in Figure 1.1.1:



```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

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

Figure 1.1-1 Add environment variables

After making the modifications, save and exit. Restart the Ubuntu system, and the cross-compiler toolchain (compiler) will be successfully installed.

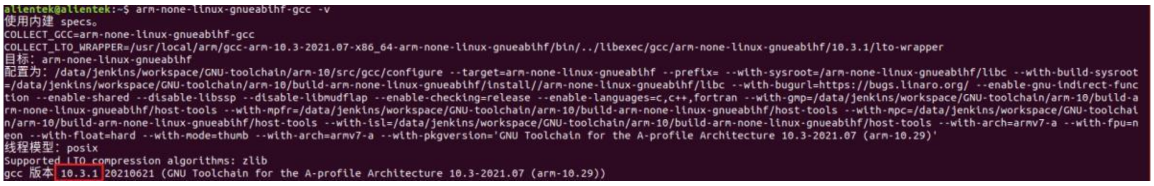
1.2 Query the cross-compiler version

After installing the cross-compiler toolchain, we can verify whether the installation is successful by querying the GCC version included in the toolchain.

Query the GCC version of the cross-compiler.

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

The running result is shown in the following figure:



```
alientek@alientek:~$ arm-none-linux-gnueabihf-gcc -v
使用内建 specs。
COLLECT_GCC=arm-none-linux-gnueabihf-gcc
COLLECT_LTO_WRAPPER=/usr/local/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf/bin/../libexec/gcc/arm-none-linux-gnueabihf/10.3.1/lto-wrapper
目标：arm-none-linux-gnueabihf
配置为：/data/jenkins/workspace/GNU-toolchain/arm-10/src/gcc/configure --target=arm-none-linux-gnueabihf --prefix= --with-sysroot=/arm-none-linux-gnueabihf/libc --with-build-sysroot
=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabihf/install//arm-none-linux-gnueabihf/libc --with-bugurl=https://bugs.linaro.org/ --enable-gnu-indirect-func
tion --enable-shared --disable-libssp --disable-libmudflap --enable-checking=release --enable-languages=c,c++,fortran --with-gmp=/data/jenkins/workspace/GNU-toolchain/arm-10/build-a
rm-none-linux-gnueabihf/host-tools --with-mpfr=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabihf/host-tools --with-mpc=/data/jenkins/workspace/GNU-toolchai
n/arm-10/build-arm-none-linux-gnueabihf/host-tools --with-isl=/data/jenkins/workspace/GNU-toolchain/arm-10/build-arm-none-linux-gnueabihf/host-tools --with-arch=armv7-a --with-fpu=n
eon --with-float=hard --with-mode=thumb --with-arch=armv7-a --with-pkgversion='GNU Toolchain for the A-profile Architecture 10.3-2021.07 (arm-10.29)'
线程模型：posix
Supported LTO compression algorithms: zlib
gcc 版本 10.3.1 20210621 (GNU Toolchain for the A-profile Architecture 10.3-2021.07 (arm-10.29))
```

Figure 1.2-1 Query the version of the cross-compiler

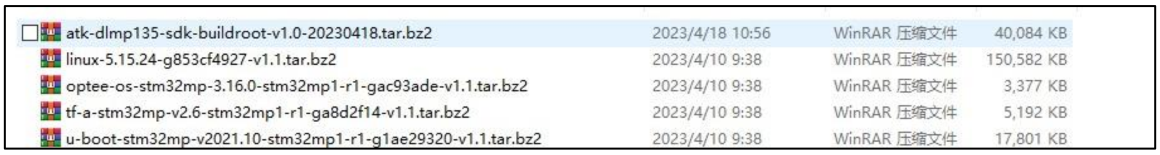
As shown in the above figure, the version of the cross-compiler GCC is 10.3.1, which proves that the ARM cross-compiler toolchain for the development board has been installed successfully.

Chapter 2. Compilation of the factory system source code

The factory system source code of the ALIENTEK ATK-DLMP135 development board is composed of four parts of source code: TF-A, OP-T EE, U-Boot, Linux kernel source code and file system.

The factory system source code was developed by ALIENTEK to adapt to the hardware resources of the ATK-DLMP135 development board, and was transplanted and modified based on the original source code provided by ST. Users can conduct secondary development and debugging based on this factory system source code. For example, if the hardware resources of the baseboard are different from those of the development board, they can modify some parts of the source code of the factory system and directly compile and debug.

The factory system source code is located at the network disk data path: STM32MP135 development board→development board CD A - basic materials→01_codes→01_Linux_factory system source code



atk-dlmp135-sdk-buildroot-v1.0-20230418.tar.bz2	2023/4/18 10:56	WinRAR 压缩文件	40,084 KB
linux-5.15.24-g853cf4927-v1.1.tar.bz2	2023/4/10 9:38	WinRAR 压缩文件	150,582 KB
optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2	2023/4/10 9:38	WinRAR 压缩文件	3,377 KB
tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2	2023/4/10 9:38	WinRAR 压缩文件	5,192 KB
u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2	2023/4/10 9:38	WinRAR 压缩文件	17,801 KB

Figure 1.2-1 Example of Factory System Source Code

As shown in the above figure, the factory source code for different parts of the development board is managed through Git. Therefore, the source code package names come with information such as Git Commit ID and factory version number. The factory source code may undergo version iterations when necessary, and the version number and other information may differ from the example in the figure. For users, it can be used directly. For the convenience of explanation in this document, v1.1 version is taken as an example here. The specific source code version is subject to the latest release. Please note to change the source code extraction instructions involving the source code version below.

The following will introduce the compilation methods of each part's source code in the system firmware startup sequence: TF-A -> OP-TEE -> U-Boot -> Linux kernel.

Before compiling, please create a directory named "ATK-DLMP135" in any directory under the Ubuntu environment. All compilation operations of the factory source code will be carried out in the ATK-DLMP135 directory.

Additionally, the following tools need to be installed in the Ubuntu 18.04 environment to avoid prompts of missing tool libraries during compilation:

2.1 Compile TF-A Source Code

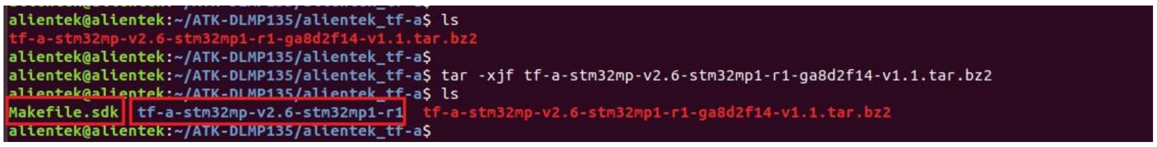
First, install the device tree compiler tool for the Ubuntu environment by entering the following command:

```
sudo apt-get install device-tree-compiler
```

Create a directory named "alientek_tf-a" in the ATK-DLMP135 directory of Ubuntu. Then, copy the factory TF-A source code package tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2 to this directory. The decompression command is as follows: (Please confirm the version of the source code you downloaded. Do not directly copy the following decompression command)

```
tar  -xjf  tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2
```

The running result is shown in the following figure:



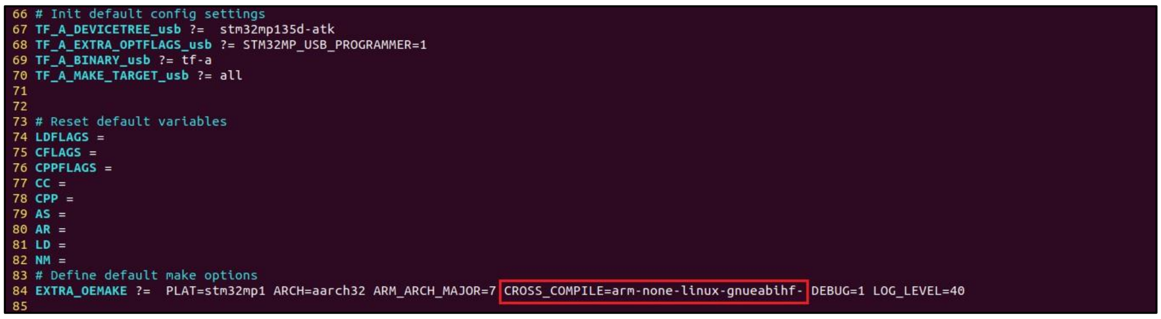
```
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls
tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ tar -xjf tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls
Makefile.sdk  tf-a-stm32mp-v2.6-stm32mp1-r1  tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$
```

Figure 2.1-1 The decompressed TF-A source code

As shown in the above figure, after decompression, a Makefile.sdk file and the TF-A source code tf-a-stm32mp-v2.6-stm32mp1-r1 appear.

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

The source code after modification is specified in Makefile.sdk to use the arm-none-linux-gnueabi hf cross-compiler toolchain installed in Chapter 1, and to compile the "stm32mp135d-atk" device tree file, as well as some other configurations, etc.



```
# Init default config settings
TF_A_DEVICETREE_usb ?=  stm32mp135d-atk
TF_A_EXTRA_OPTFLAGS_usb ?= STM32MP_USB_PROGRAMMER=1
TF_A_BINARY_usb ?= tf-a
TF_A_MAKE_TARGET_usb ?= all


# Reset default variables
LDFLAGS =
CFLAGS =
CPPFLAGS =
CC =
CPP =
AS =
AR =
LD =
NM =
# Define default make options
EXTRA_OEMAKE  ?=   PLAT=stm32mp1 ARCH=aarch32 ARM_ARCH_MAJOR=7 CROSS_COMPILE=arm-none-linux-gnueabihf- DEBUG=1 LOG_LEVEL=40

```

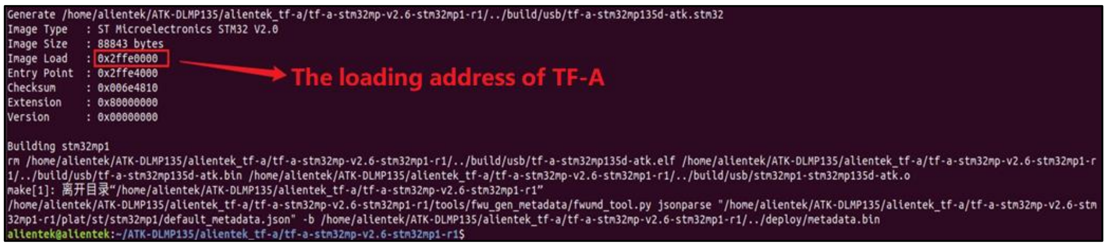
Figure 2.1-2 Makefile.sdk specifies an example of cross-compilation toolchain

When compiling the factory source code, users do not need to modify any content. The TF-A source code device tree path adapted to the hardware resources of the ATK-DLMP135 development board is fdts/, and the device tree files are stm32mp135d-atk.dts, stm32mp13-pinctrl-atk.dtsi, and st m32mp135d-atk-fw-config.dts.

Enter the TF-A source code directory tf-a-stm32mp-v2.6-stm32mp1-r1, and then execute the compilation command. The compilation result is as follows.

```
cd tf-a-stm32mp-v2.6-stm32mp1-r1/
./build.sh
```

The running result is shown in the following figure:



```
Generate /home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/../build/usb/tf-a-stm32mp135d-atk.stm32
Image Type   : ST Microelectronics STM32 V2.0
Image Size   : 88843 bytes
Image Load   : 0x2ffe0000
Entry Point  : 0x2ffe4000
Checksum     : 0x006e4810
Extension    : 0x80000000
Version      : 0x00000000

Building stm32mp1
rm /home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/../build/usb/tf-a-stm32mp135d-atk.elf /home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r
1/../build/usb/tf-a-stm32mp135d-atk.bin /home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/../build/usb/stm32mp1-stm32mp135d-atk.o
make[1]: 离开目录“/home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1”
/home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fwu_gen_metadata/fwumd_tool.py jsonparse "/home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm
32mp1-r1/plat/st/stm32mp1/default_metadata.json" -b /home/alientek/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/../deploy/metadata.bin
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1$
```

Figure 2.1-3 TF-A source code compilation result

After the compilation is completed, two directories "build" and "deploy" will be generated in the upper-level directory, namely the "alienek_tf-a" directory, as shown in the following figure.

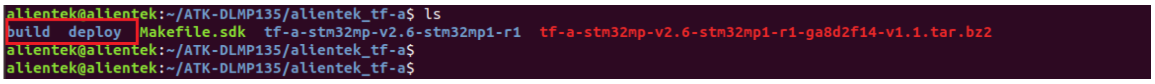

```
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls
build  deploy  Makefile.sdk  tf-a-stm32mp-v2.6-stm32mp1-r1  tf-a-stm32mp-v2.6-stm32mp1-r1-ga8d2f14-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$
```

Figure 2.1-4 TF-A source code compilation output directory

The "build" directory contains all the files generated during the TF-A source code compilation process, such as .a, .d, .o, .map, .bin, etc. formats. Most of these files are not accessible to users.

However, the files in the "deploy" directory are very important. They are processed by the Makefile.sdk script on the binary files generated in the "build" directory and then copied to the "deploy" directory. It includes 5 files: metadata.bin, tf-a-stm32mp135d-atk-emmc.stm32, tf-a-stm32mp135d-atk-sdcard.stm32, tf-a-stm32mp135d-atk-usb.stm32, and the subdirectory fwconfig with stm32mp135d-atk-fw-config-optee.dtb. These 5 files are the final products of compiling the TF-A source code for the factory, which is the TF-A firmware for the ATK-DLMP135 development board. Later, we will introduce how to use these TF-A firmware files to create a system image for burning.

Here is a reminder: If the user modifies the TF-A source code, device tree, etc., then when recompiling, they need to first delete the "build" directory and the "deploy" directory, and then re-execute the compilation command.

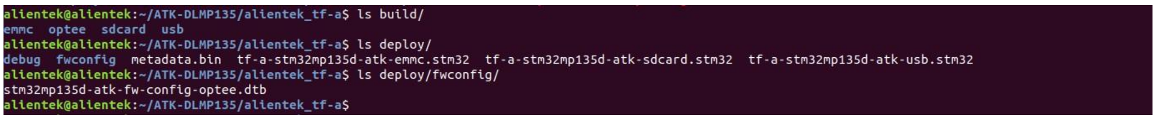

```
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls build/
emmc  optee  sdcard  usb
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls deploy/
debug  fwconfig  metadata.bin  tf-a-stm32mp135d-atk-emmc.stm32  tf-a-stm32mp135d-atk-sdcard.stm32  tf-a-stm32mp135d-atk-usb.stm32
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$ ls deploy/fwconfig/
stm32mp135d-atk-fw-config-optee.dtb
alientek@alientek:~/ATK-DLMP135/alientek_tf-a$
```

Figure 2.1-5 TF-A source code compilation result

The factory-compiled TF-A source code has been completed. Now, let's introduce the compilation process of OP-TEE source code.

2.2 Compile OP-TEE source code

OP-TEE is an open-source trusted execution environment (TEE) for implementing Arm TrustZone technology, which is related to the security field. We will not elaborate on the working principle of OP-TEE here. Interested parties can refer to relevant materials for further study.

Create a directory named "alientek_optee" in the ATK-DLMP135 directory of ubuntu. Then, copy the ALIENTEK factory OP-TEE source code package "optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2" to the "alientek_optee" directory. The decompression command is as follows: (Please confirm the version of the source code you downloaded and do not directly copy the following decompression command)

```
tar  -xjf  optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2
```

The running result is shown in the following figure:

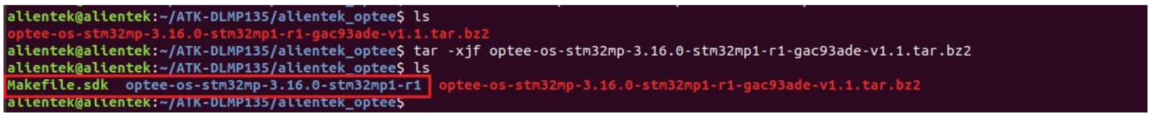
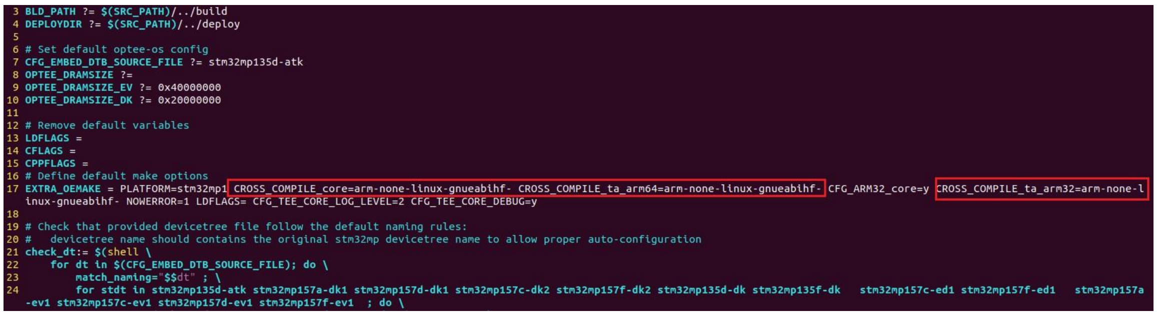

```
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls
optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_optee$ tar -xjf optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls
Makefile.sdk  optee-os-stm32mp-3.16.0-stm32mp1-r1  optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_optee$
```

Figure 2.2-1 The decompressed OP-TEE source code

As shown in the above figure, similar to the TF-A source code, after decompression, there appears a Makefile.sdk file and the OP-TEE source code directory optee-os-stm32mp-3.16.0-stm32mp1-r1.

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 internal Makefile of OP-TEE to compile the OP-TEE source code.

The factory source code has been modified. In the Makefile.sdk, it specifies to use the arm-buildroot-linux-gnue abihf cross-compiler toolchain installed in Chapter 1, and specifies to compile the "stm32mp135d-atk" device tree file, as well as some other configurations, etc.



```
BLD_PATH ?= $(SRC_PATH)/../build
DEPLOYDIR ?= $(SRC_PATH)/../deploy

# Set default optee-os config
CFG_EMBED_DTB_SOURCE_FILE ?= stm32mp135d-atk
OPTEE_DRAMSIZE ?=
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- CROSS_COMPILE_ta_arm64=arm-none-linux-gnueabihf- CFG_ARM32_core=y CROSS_COMPILE_ta_arm32=arm-none-l
inux-gnueabihf- NOWERROR=1 LDFLAGS= CFG_TEE_CORE_LOG_LEVEL=2 CFG_TEE_CORE_DEBUG=y

# Check that provided devicetree file follow the default naming rules:
#   devicetree name should contains the original stm32mp devicetree name to allow proper auto-configuration
check_dt:= $(shell \
    for dt in $(CFG_EMBED_DTB_SOURCE_FILE); do \
        match_naming="$$dt" ; \
        for stdt in stm32mp135d-atk stm32mp157a-dk1 stm32mp157d-dk1 stm32mp157c-dk2 stm32mp157f-dk2 stm32mp135d-dk stm32mp135f-dk   stm32mp157c-ed1 stm32mp157f-ed1   stm32mp157a
-ev1 stm32mp157c-ev1 stm32mp157d-ev1 stm32mp157f-ev1  ; do \
```

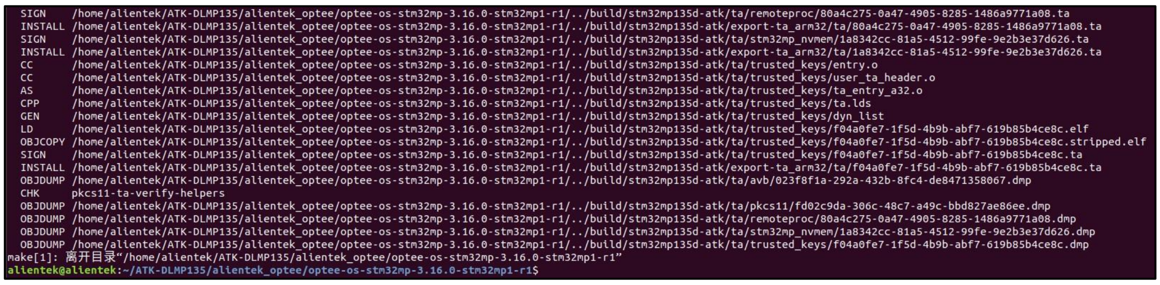
Figure 2.2-2 Makefile.sdk specifies an example of cross-compilation toolchain

When compiling the factory source code, users do not need to modify any content. The device tree path for the OPTEE source code adapted to the hardware resources of the ATK-DLMP135 development board is core/arch/arm/dts/, and the device tree files are stm32mp135d-atk.dts and stm32mp13-pin ctrl-atk.dtsi.

Enter the source code directory of OPTEE optee-os-stm32mp-3.16.0-stm32mp1-r1, export the cross-compilation toolchain environment variables first, and then execute the compilation command. The compilation result is as follows.

```
cd optee-os-stm32mp-3.16.0-stm32mp1-r1/
./build.sh
```

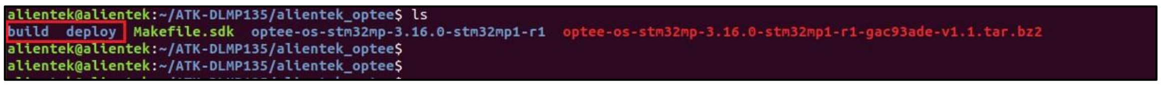
The compilation results are shown in the following figure:



```
  SIGN    /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/remoteproc/80a4c275-0a47-4905-8285-1486a9771a08.ta
  INSTALL /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/export-ta_arm32/ta/80a4c275-0a47-4905-8285-1486a9771a08.ta
  SIGN    /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/stm32mp_nvmem/1a8342cc-81a5-4512-99fe-9e2b3e37d626.ta
  INSTALL /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/export-ta_arm32/ta/1a8342cc-81a5-4512-99fe-9e2b3e37d626.ta
  CC      /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/entry.o
  CC      /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/user_ta_header.o
  AS      /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/ta_entry_a32.o
  CPP     /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/ta.lds
  GEN     /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/dyn_list
  LD      /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.elf
  OBJCOPY /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.stripped.elf
  SIGN    /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
  INSTALL /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/export-ta_arm32/ta/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
  OBJDUMP /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/avb/023f8f1a-292a-432b-8fc4-de8471358067.dmp
  CHK     pkcs11-ta-verify-helpers
  OBJDUMP /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/pkcs11/fd02c9da-306c-48c7-a49c-bbd827ae86ee.dmp
  OBJDUMP /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/remoteproc/80a4c275-0a47-4905-8285-1486a9771a08.dmp
  OBJDUMP /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/stm32mp_nvmem/1a8342cc-81a5-4512-99fe-9e2b3e37d626.dmp
  OBJDUMP /home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1/../build/stm32mp135d-atk/ta/trusted_keys/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.dmp
make[1]: 离开目录“/home/alientek/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1”
alientek@alientek:~/ATK-DLMP135/alientek_optee/optee-os-stm32mp-3.16.0-stm32mp1-r1$
```

Figure 2.2-3 The OP-TEE source code has been successfully compiled.

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



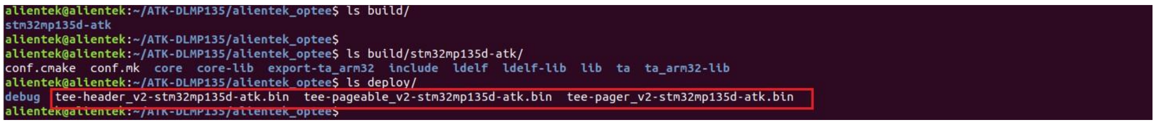
```
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls
build  deploy  Makefile.sdk  optee-os-stm32mp-3.16.0-stm32mp1-r1  optee-os-stm32mp-3.16.0-stm32mp1-r1-gac93ade-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_optee$
alientek@alientek:~/ATK-DLMP135/alientek_optee$
```

Figure 2.2-4 OP-TEE source code compilation output directory

The "build" directory contains all the files generated during the compilation process, such as .a, .o, etc. formats. Most of these files are not accessible to users.

However, the files in the "deploy" directory are very important. They are processed by the Makefile.sdk script on the binary files generated in the "build" directory and then copied to the "deploy" directory. It contains three files: tee-header_v2-stm32mp135d-atk.bin, tee-pageable_v2-stm32mp135d-atk.bin, and tee-pager_v2-stm32mp135d-atk.bin. These three files are the final products of compiling the OP-TEE source code for the factory and are the OP-TEE firmware for the ATK-DLMP135 development board. Later, we will introduce how to use these OP-TEE firmware files to create a system image for burning.

Please note that if the user modifies the OP-TEE source code, device tree, etc., then when recompiling, they need to first delete the "build" directory and "deploy" directory, and then re-execute the compilation command.



```
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls build/
stm32mp135d-atk
alientek@alientek:~/ATK-DLMP135/alientek_optee$
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls build/stm32mp135d-atk/
conf.cmake  conf.mk  core  core-lib  export-ta_arm32  include  ldelf  ldelf-lib  lib  ta  ta_arm32-lib
alientek@alientek:~/ATK-DLMP135/alientek_optee$ ls deploy/
debug  tee-header_v2-stm32mp135d-atk.bin  tee-pageable_v2-stm32mp135d-atk.bin  tee-pager_v2-stm32mp135d-atk.bin
alientek@alientek:~/ATK-DLMP135/alientek_optee$
```

Figure 2.2-5 OP-TEE source code compilation result

The factory-provided OP-TEE source code has been compiled. Now, let's introduce the compilation of U-Boot source code.

2.3 Compile U-Boot source code

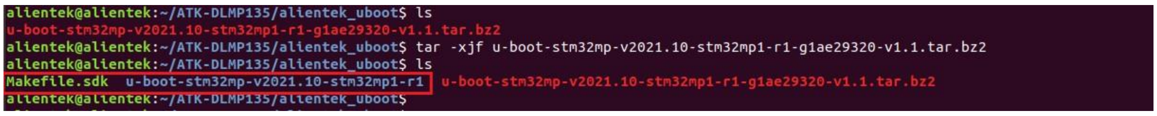
To compile uboot, some libraries need to be installed. Enter the following command:

```
sudo apt-get install bison flex
```

Create a directory named "alientek_uboot" in the ATK-DLMP135 directory of Ubuntu. Then, copy the ALIENTEK factory U-Boot source code package "u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2" to this directory. The decompression command is as follows: (Please confirm the version of the source code you have downloaded. Do not directly copy the following decompression command)

```
tar -xjf u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2
```

The running result is shown in the following figure:

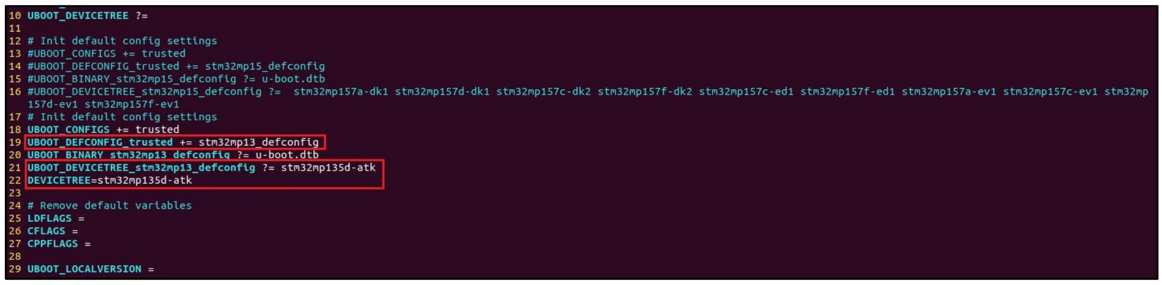


```
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls
u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ tar -xjf u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls
Makefile.sdk  u-boot-stm32mp-v2021.10-stm32mp1-r1  u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_uboot$
```

Figure 2.3-1 The decompressed U-Boot source code

As can be seen from the above figure, similar to the previous source code, after decompression, there appears a Makefile.sdk file and the U-Boot source code directory u-boot-st m32mp-v2021.10-stm32mp1-r1.

The Makefile.sdk file here mainly defines some compilation properties, such as the default source code configuration of U-Boot, specifying the name of the compiled device tree, some compilation options, etc. Eventually, the Makefile.sdk will call the internal Makefile of U-Boot to compile the U-Boot source code.



```
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 ?=  stm32mp157a-dk1 stm32mp157d-dk1 stm32mp157c-dk2 stm32mp157f-dk2 stm32mp157c-ed1 stm32mp157f-ed1 stm32mp157a-ev1 stm32mp157c-ev1 stm32mp157d-ev1 stm32mp157f-ev1
# Init default config settings
UBOOT_CONFIGS += trusted
UBOOT_DEFCONFIG_trusted += stm32mp13_defconfig
UBOOT_BINARY_stm32mp13_defconfig ?= u-boot.dtb
UBOOT_DEVICETREE_stm32mp13_defconfig ?= stm32mp135d-atk
DEVICETREE=stm32mp135d-atk

# Remove default variables
LDFLAGS =
CFLAGS =
CPPFLAGS =

UBOOT_LOCALVERSION =
```

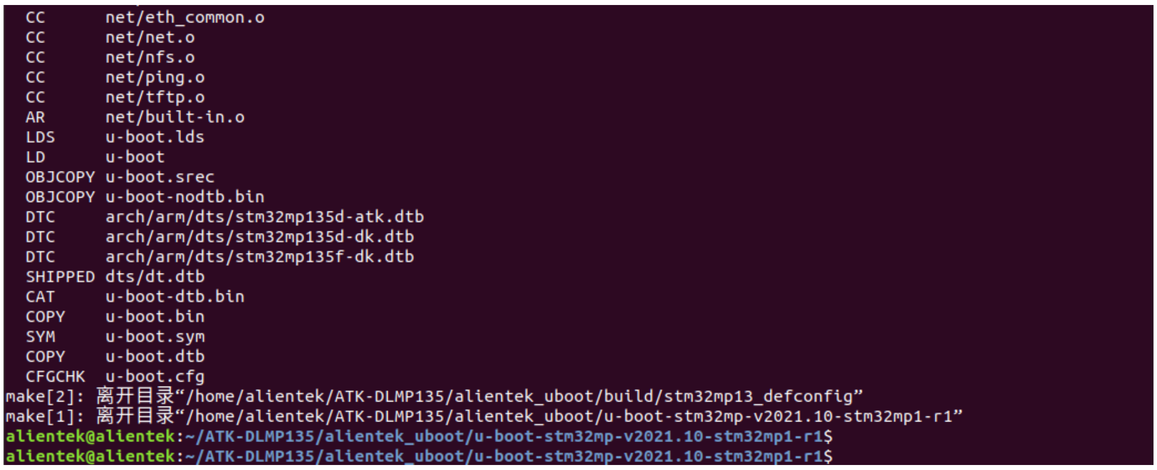
Figure 2.3-2 Makefile.sdk specifies the compilation configuration example

When compiling the factory source code, users do not need to modify any content. The U-Boot source code device tree path adapted to the hardware resources of the ATK-DLMP135 development board is arch/arm/dts/, and the device tree files are stm32mp135d-atk.dts, stm32mp135d-atk-u-boot.dtsi, and stm32mp13-pinctrl-atk.dtsi.

Enter the U-Boot source code directory u-boot-stm32mp-v2021.10-stm32mp1-r1, export the cross-compiler toolchain environment variables, and then execute the compilation command. The compilation result is as follows.

```
cd u-boot-stm32mp-v2021.10-stm32mp1-r1/
./build.sh
```

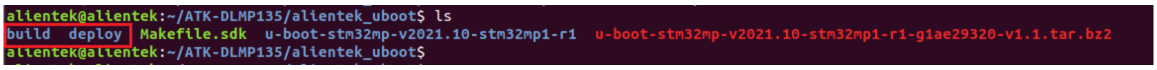
The compilation results are shown in the following figure:



```
  CC      net/eth_common.o
  CC      net/net.o
  CC      net/nfs.o
  CC      net/ping.o
  CC      net/tftp.o
  AR      net/built-in.o
  LDS     u-boot.lds
  LD      u-boot
  OBJCOPY u-boot.srec
  OBJCOPY u-boot-nodtb.bin
  DTC     arch/arm/dts/stm32mp135d-atk.dtb
  DTC     arch/arm/dts/stm32mp135d-dk.dtb
  DTC     arch/arm/dts/stm32mp135f-dk.dtb
  SHIPPED dts/dt.dtb
  CAT     u-boot-dtb.bin
  COPY    u-boot.bin
  SYM     u-boot.sym
  COPY    u-boot.dtb
  CFGCHK  u-boot.cfg
make[2]: 离开目录"/home/alientek/ATK-DLMP135/alientek_uboot/build/stm32mp13_defconfig"
make[1]: 离开目录"/home/alientek/ATK-DLMP135/alientek_uboot/u-boot-stm32mp-v2021.10-stm32mp1-r1"
alientek@alientek:~/ATK-DLMP135/alientek_uboot/u-boot-stm32mp-v2021.10-stm32mp1-r1$
alientek@alientek:~/ATK-DLMP135/alientek_uboot/u-boot-stm32mp-v2021.10-stm32mp1-r1$
```

Figure 2.3-3 The U-Boot source code has been compiled successfully.

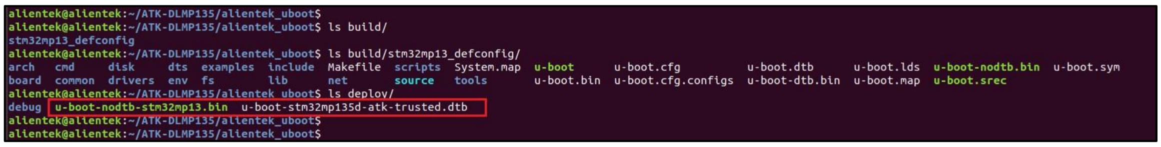
Similar to the previous source code, after the U-Boot source code is compiled, two directories "build" and "deploy" will be generated in the upper-level directory, namely the "alientek_uboot" directory, as shown in the following figure.



```
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls
build  deploy  Makefile.sdk  u-boot-stm32mp-v2021.10-stm32mp1-r1  u-boot-stm32mp-v2021.10-stm32mp1-r1-g1ae29320-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_uboot$
```

Figure 2.3-4 The U-Boot source code compilation generates the following directories:

The "build" directory contains all the files generated during the compilation process. Most of these files are not accessible to users. However, the files in the "deploy" directory are very important. They are processed by the "Makefile.sdk" script on the binary files in the "build" directory and then copied to the "deploy" directory. It contains two files: "u-boot-nodtb-stm32mp13.bin" and "u-boot-stm32mp135d-atk-trusted.dtb". These two files are the final products of compiling the U-Boot source code for the factory version, and they are the U-Boot firmware for the ATK-DLMP135 development board.



```
alientek@alientek:~/ATK-DLMP135/alientek_uboot$
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls build/
stm32mp13_defconfig
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls build/stm32mp13_defconfig/
arch   cmd     disk     dts  examples  include  Makefile  scripts  System.map  u-boot      u-boot.cfg          u-boot.dtb      u-boot.lds  u-boot-nodtb.bin  u-boot.sym
board  common  drivers  env  fs        lib      net       source   tools       u-boot.bin  u-boot.cfg.configs  u-boot-dtb.bin  u-boot.map  u-boot.srec
alientek@alientek:~/ATK-DLMP135/alientek_uboot$ ls deploy/
debug  u-boot-nodtb-stm32mp13.bin  u-boot-stm32mp135d-atk-trusted.dtb
alientek@alientek:~/ATK-DLMP135/alientek_uboot$
alientek@alientek:~/ATK-DLMP135/alientek_uboot$
```

Figure 2.3-5 The compiled output of U-Boot source code

The factory-installed U-Boot source code has been compiled. Now, we will introduce the process of packaging OP-TEE and U-Boot firmware using the fiptool tool.

2.4 Using fiptool to Package OP-TEE and U-Boot Folders

This section explains how to use the fiptool tool to combine the OP-TEE and U-Boot firmware files into a single image file named fip-stm32mp135d-atk-optee.bin, which will be subsequently burned onto the development board.

First, let's understand what FIP is and its main function.

FIP, in full form, is Firmware Image Package. Literally translated, it means "firmware image package". FIP is a packaging format used by TF-A to package the bootloader image file and other payload files into a single binary file. Currently, the ecosystem development package released by ST recommends using FIP to start the STM32MP1 platform. Only the TF-A (with the 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. The TF-A runtime can load and authenticate the FIP image.

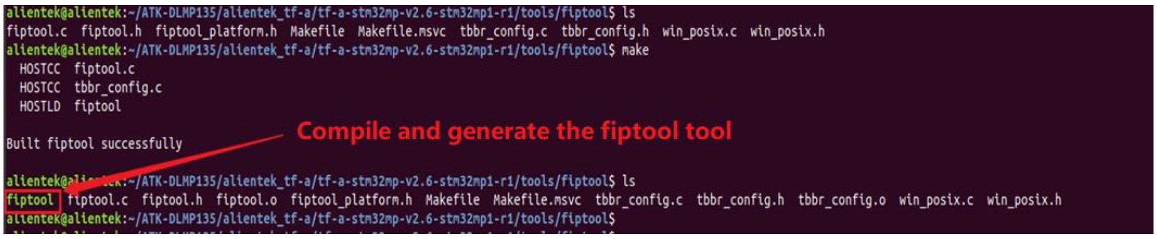
Therefore, the factory system startup and operation process of the ATK-DLMP135 development board can be generally described as:

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 that implements the FIP function is provided. Its source code path is tools/fiptool/. Now, let's enter the TF-A fiptool source code directory for compilation to generate the fiptool tool. The compilation instructions are as follows:

```
cd  tf-a-stm32mp-v2.6-stm32mp1-r1/      // Enter the directory of your own TF-A source code
cd  tools/fiptool                       // Enter the source code of fiptool
make                                    // Compile the source code of fiptool
```

The running result is shown in the following figure.



```
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$ ls
fiptool.c  fiptool.h  fiptool_platform.h  Makefile  Makefile.msvc  tbbr_config.c  tbbr_config.h  win_posix.c  win_posix.h
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$ make
  HOSTCC  fiptool.c
  HOSTCC  tbbr_config.c
  HOSTLD  fiptool

Built fiptool successfully

alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$ ls
fiptool  fiptool.c  fiptool.h  fiptool.o  fiptool_platform.h  Makefile  Makefile.msvc  tbbr_config.c  tbbr_config.h  tbbr_config.o  win_posix.c  win_posix.h
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$
```

Figure 2.4-1 Generate the fiptool tool

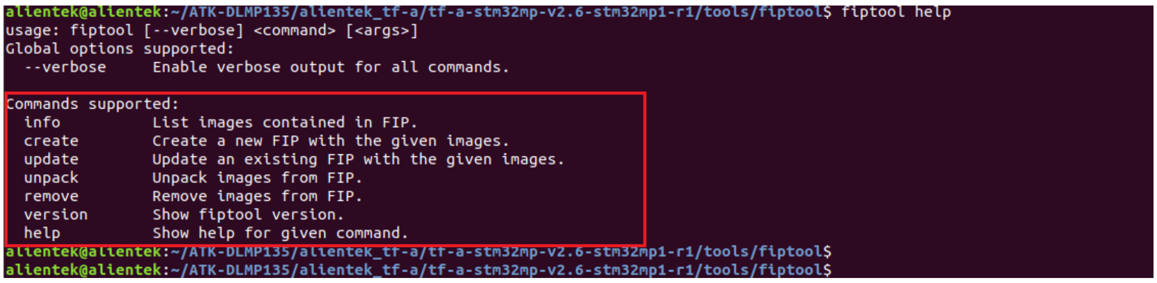
Since fiptool is a tool used on PC computers under Ubuntu, there is no need to use a cross-compilation toolchain for compilation.

We will copy the compiled fiptool tool generated above to the /usr/bin directory of Ubuntu, so that the tool can be invoked in any terminal of Ubuntu.

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

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

```
fiptool  help
```



```
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$ 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.
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$
alientek@alientek:~/ATK-DLMP135/alientek_tf-a/tf-a-stm32mp-v2.6-stm32mp1-r1/tools/fiptool$
```

Figure 2.4-2 fiptool help information

The fiptool tool mainly provides the following function parameters:

- **info**: Lists the image information in FIP;
- **create**: Creates a new FIP image based on the provided binary image file;
- **update**: Updates the existing FIP image;
- **unpack**: Unpacks the FIP image;
- **remove**: Deletes some binary files from the FIP image;

Next, we use the fiptool tool to package the binary image files such as OP-TEE and U-Boot into a single image file named "fip-stm32mp135d-atk-optee.bin".

First, we create a directory named "fip" in the ATK-DLMP135 directory of Ubuntu. We then copy the six image files generated after compiling TF-A, OP-TEE, and U-Boot to the "fip" directory.

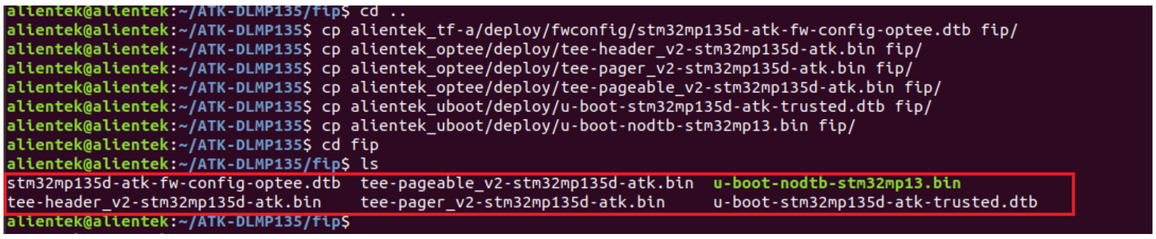
Table 1 Files Required for Packaging FIP Image Package

Code	Current Directory	File name
TF-A	**alientek_tf-a**/deploy/fwconfig/	stm32mp135d-atk-fw-config-optee.dtb
OP-TEE	**alientek_optee**/deploy/	tee-header_v2-stm32mp135d-atk.bin
		tee-pager_v2-stm32mp135d-atk.bin
		tee-pageable_v2-stm32mp135d-atk.bin
U-Boot	**alientek_uboot**/deploy/	u-boot-stm32mp135d-atk-trusted.dtb
		u-boot-nodtb-stm32mp13.bin

During the process of copying files, you can execute the following command:

```
mkdir fip      //First create the "fip" directory, then copy
cp alientek_tf-a/deploy/fwconfig/stm32mp135d-atk-fw-config-optee.dtb fip/
cp alientek_optee/deploy/tee-header_v2-stm32mp135d-atk.bin fip/
cp alientek_optee/deploy/tee-pager_v2-stm32mp135d-atk.bin fip/
cp alientek_optee/deploy/tee-pageable_v2-stm32mp135d-atk.bin fip/
cp alientek_uboot/deploy/u-boot-stm32mp135d-atk-trusted.dtb fip/
cp alientek_uboot/deploy/u-boot-nodtb-stm32mp13.bin fip/
```

The running result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135/fip$ cd ..
alientek@alientek:~/ATK-DLMP135$ cp alientek_tf-a/deploy/fwconfig/stm32mp135d-atk-fw-config-optee.dtb fip/
alientek@alientek:~/ATK-DLMP135$ cp alientek_optee/deploy/tee-header_v2-stm32mp135d-atk.bin fip/
alientek@alientek:~/ATK-DLMP135$ cp alientek_optee/deploy/tee-pager_v2-stm32mp135d-atk.bin fip/
alientek@alientek:~/ATK-DLMP135$ cp alientek_optee/deploy/tee-pageable_v2-stm32mp135d-atk.bin fip/
alientek@alientek:~/ATK-DLMP135$ cp alientek_uboot/deploy/u-boot-stm32mp135d-atk-trusted.dtb fip/
alientek@alientek:~/ATK-DLMP135$ cp alientek_uboot/deploy/u-boot-nodtb-stm32mp13.bin fip/
alientek@alientek:~/ATK-DLMP135$ cd fip
alientek@alientek:~/ATK-DLMP135/fip$ ls
stm32mp135d-atk-fw-config-optee.dtb  tee-pageable_v2-stm32mp135d-atk.bin  u-boot-nodtb-stm32mp13.bin
tee-header_v2-stm32mp135d-atk.bin    tee-pager_v2-stm32mp135d-atk.bin     u-boot-stm32mp135d-atk-trusted.dtb
alientek@alientek:~/ATK-DLMP135/fip$
```

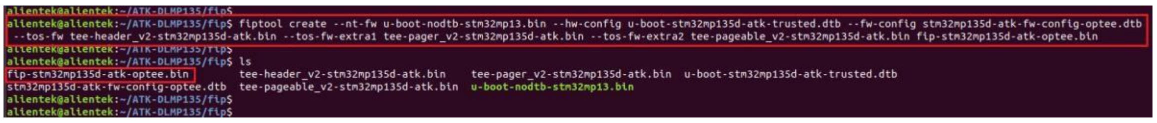
Figure 2.4-3 The required files under the fip directory

One tip: To facilitate your own copying, users can write the copy instructions into a single script to achieve a one-click copying effect.

Enter the fip directory and use the fiptool to create the FIP image file. Execute the following command:

```
cd fip/
fiptool create --nt-fw u-boot-nodtb-stm32mp13.bin --hw-config u-boot-stm32mp135d-atk-trusted.dtb --fw-config stm32mp135d-atk-fw-config-optee.dtb --tos-fw tee-header_v2-stm32mp135d-atk.bin --tos-fw-extra1 tee-pager_v2-stm32mp135d-atk.bin --tos-fw-extra2 tee-pageable_v2-stm32mp135d-atk.bin fip-stm32mp135d-atk-optee.bin
```

The running result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135/fip$
alientek@alientek:~/ATK-DLMP135/fip$ fiptool create --nt-fw u-boot-nodtb-stm32mp13.bin --hw-config u-boot-stm32mp135d-atk-trusted.dtb --fw-config stm32mp135d-atk-fw-config-optee.dtb
 --tos-fw tee-header_v2-stm32mp135d-atk.bin --tos-fw-extra1 tee-pager_v2-stm32mp135d-atk.bin --tos-fw-extra2 tee-pageable_v2-stm32mp135d-atk.bin fip-stm32mp135d-atk-optee.bin
alientek@alientek:~/ATK-DLMP135/fip$
alientek@alientek:~/ATK-DLMP135/fip$ ls
fip-stm32mp135d-atk-optee.bin        tee-header_v2-stm32mp135d-atk.bin    tee-pager_v2-stm32mp135d-atk.bin  u-boot-stm32mp135d-atk-trusted.dtb
stm32mp135d-atk-fw-config-optee.dtb  tee-pageable_v2-stm32mp135d-atk.bin  u-boot-nodtb-stm32mp13.bin
alientek@alientek:~/ATK-DLMP135/fip$
alientek@alientek:~/ATK-DLMP135/fip$
```

Figure 2.4-4 Create FIP image file

As shown in the above figure, the FIP image file fip-stm32mp135d-atk-optee.bin has been generated. This binary file contains the OP-TEE firmware and the U-Boot firmware. When the system of the development board starts up at the factory, TF-A will load and authenticate this firmware, and then run the next stage of OP-TEE and U-Boot.

One tip is that when users modify the OP-TEE source code or the U-Boot source code, they need to copy the image file generated by compiling the corresponding source code to this fip directory, and then re-execute the above command to create a new FIP image file fip-stm32mp135d-atk-optee.bin, and then burn it onto the development board. For example, if the user has modified the U-Boot source code, they need to copy the compiled u-boot-stm32mp135d-atk-trusted.dtb and u-boot-nodtb-stm32mp13.bin files to the fip directory, and then re-execute the fiptool command to generate a new FIP image. You can write a script and use the script for compilation. The code is as follows:

```
#!/bin/sh

cp ../alientek_tf-a/deploy/fwconfig/stm32mp135d-atk-fw-config-optee.dtb./
cp ../alientek_optee/deploy/tee-* ./
cp    ../alientek_uboot/deploy/u-boot-* ./

fiptool create --nt-fw u-boot-nodtb-stm32mp13.bin --hw-config u-boot-stm32mp135d-atk-trusted.dtb --fw-config stm32mp135d-atk-fw-config-optee.dtb --tos-fw tee-header_v2-stm32mp135d-atk.bin --tos-fw-extra1 tee-pager_v2-stm32mp135d-atk.bin --tos-fw-extra2 tee-pageable_v2-stm32mp135d-atk.bin fip-stm32mp135d-atk-optee.bin
```

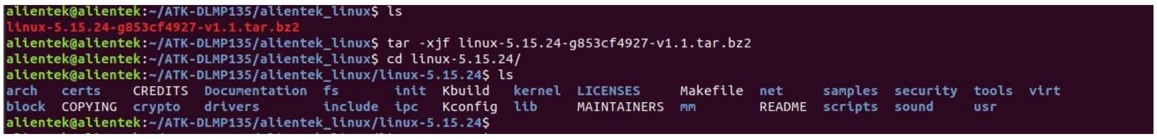
Create a file named "build.sh" in the "fip" directory. Copy the above code into the script and grant the file execution permission.

2.5 Compile the Linux kernel source code

Create a directory named "alientek_linux" in the ATK-DLMP135 directory of Ubuntu. Then, copy the ALIENTEK Technology factory-provided Linux source code package linux-5.15.24-g853cf4927-v1.1.tar.bz2 to this directory. Execute the decompression command and enter the kernel source code directory; (Please confirm the version of the source code you downloaded, do not directly copy the following decompression command)

```
tar -xjf linux-5.15.24-g853cf4927-v1.1.tar.bz2
cd linux-5.15.24/
```

The running result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135/alientek_linux$ ls
linux-5.15.24-g853cf4927-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_linux$ tar -xjf linux-5.15.24-g853cf4927-v1.1.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_linux$ cd linux-5.15.24/
alientek@alientek:~/ATK-DLMP135/alientek_linux/linux-5.15.24$ ls
arch   certs   CREDITS  Documentation  fs       init  Kbuild   kernel  LICENSES     Makefile  net     samples  security  tools  virt
block  COPYING crypto   drivers        include  ipc   Kconfig  lib     MAINTAINERS  mm        README  scripts  sound     usr
alientek@alientek:~/ATK-DLMP135/alientek_linux/linux-5.15.24$
```

Figure 2.5-1 Uncompressed Linux source code

When compiling the factory source code, users do not need to modify any content. The device tree path for the Linux kernel source code adapted to the hardware resources of the ATK-DLMP135 development board is arch/arm/boot/dts/. The device tree files are stm32mp135d-atk.dts, stm32mp135d-atk-hdmi.dts, stm32mp135d-atk-wifi-bluetooth.dts, and stm32mp13-pinctrl-atk.dtsi. To facilitate everyone's compilation, we have provided a one-key compilation script. When compiling for the first time, you can directly run the script for compilation. The script is as shown in the following figure:

```
#!/bin/sh
# script-name: build.sh
# version: 1.0
# date: 2023-04-08
# author: alientek
# This script is used as the Linux script for compiling ATK-DLM135.
#

make distclean    # Clear compilation
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- stm32mp1_atk_defconfig
# Select the configuration file as "stm32mp1_atk_defconfig"
# Compile the kernel
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- uImage vmlinux dtbs
LOADADDR=0xC2000040 -j4
# Compile the kernel module
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- modules -j16
# Create a new "tmp" directory in the current directory to store the compiled target files.
if [ ! -e "./tmp" ]; then
mkdir tmp
fi
rm -rf tmp/* 23
# Install the compiled module to the tmp directory and remove the module debug information by
setting INSTALL_MOD_STRIP=1
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- modules_install
INSTALL_MOD_PATH=tmp INSTALL_MOD_STRIP=1
# Delete the "source" directory in the module directory
```

```
rm -rf tmp/lib/modules/5.15.24/source
# Delete the "build" directory in the directory of the module
rm -rf tmp/lib/modules/5.15.24/build 30
# Jump to the module directory
cd tmp/lib/modules
# Compress the kernel module
tar -jcvf ../../modules.tar.bz2 .
cd -
rm -rf tmp/lib
# Copy the uImage file to the tmp directory
cp arch/arm/boot/uImage tmp
# Copy all the compiled device tree files to the current tmp directory.
cp arch/arm/boot/dts/stm32mp135d-atk*.dtb tmp

echo " After compilation is complete, please check the tmp directory."
```

Line 9: Clear all compilations.

Line 10: Select the configuration file of the ALIENTEK, and running this command will generate a ".config" file in the source code directory of Linux.

Line 13: Run the compilation of the kernel and the device tree.

Line 15: Run the compilation of the kernel module.

Lines 19 to 22: Create a tmp directory to be used for saving the required files generated during compilation. Line 25: Package the compiled module into the tmp directory.

Line 27: Delete the source directory.

Line 29: Delete the build directory.

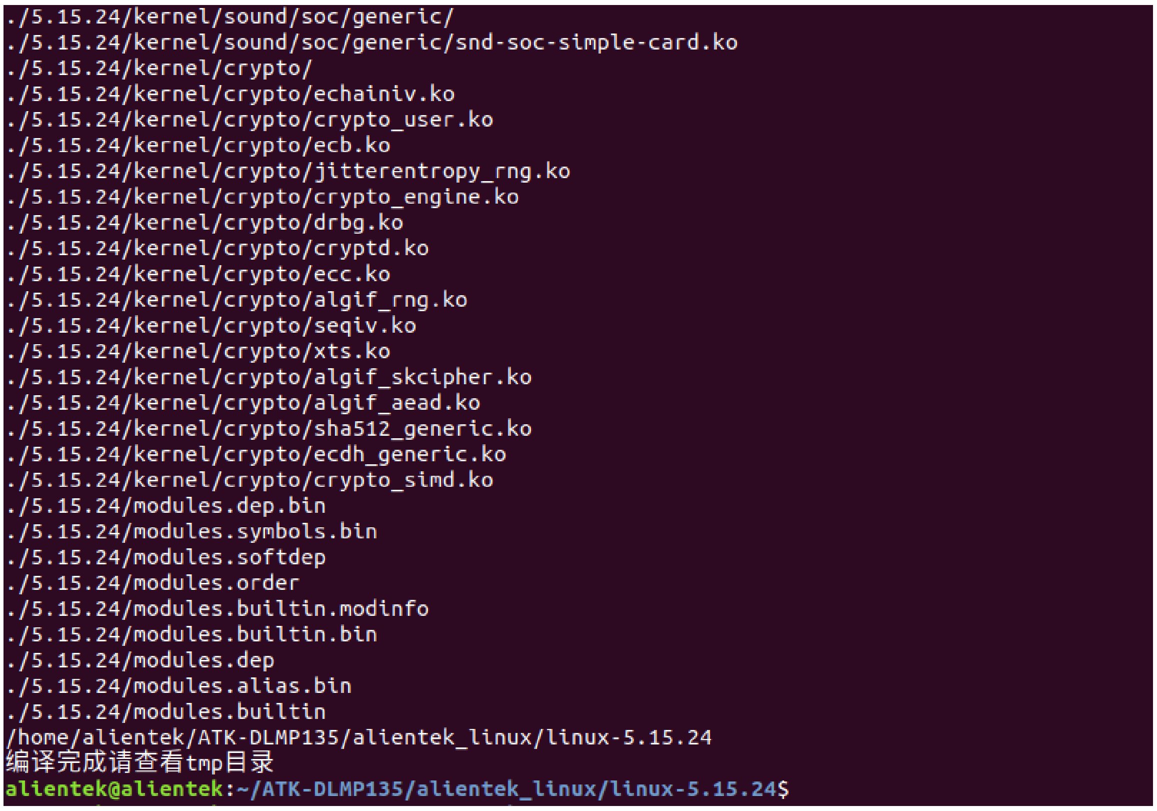
Line 39: Copy the uImage to the tmp directory.

Line 42: Copy the device tree required by the ATK-DLM135 development board to the tmp directory.

The above script can be run as a single command (it must have the.config file). There is no need to use this script every time. For example, when the author needs to clear the kernel compilation, they can copy the 9th line of the script and run it. Simply running the script can compile the code as follows:

```
./build.sh
```

The compilation results are shown in the following figure:



```
./5.15.24/kernel/sound/soc/generic/
./5.15.24/kernel/sound/soc/generic/snd-soc-simple-card.ko
./5.15.24/kernel/crypto/
./5.15.24/kernel/crypto/echainiv.ko
./5.15.24/kernel/crypto/crypto_user.ko
./5.15.24/kernel/crypto/ecb.ko
./5.15.24/kernel/crypto/jitterentropy_rng.ko
./5.15.24/kernel/crypto/crypto_engine.ko
./5.15.24/kernel/crypto/drbg.ko
./5.15.24/kernel/crypto/cryptd.ko
./5.15.24/kernel/crypto/ecc.ko
./5.15.24/kernel/crypto/algif_rng.ko
./5.15.24/kernel/crypto/seqiv.ko
./5.15.24/kernel/crypto/xts.ko
./5.15.24/kernel/crypto/algif_skcipher.ko
./5.15.24/kernel/crypto/algif_aead.ko
./5.15.24/kernel/crypto/sha512_generic.ko
./5.15.24/kernel/crypto/ecdh_generic.ko
./5.15.24/kernel/crypto/crypto_simd.ko
./5.15.24/modules.dep.bin
./5.15.24/modules.symbols.bin
./5.15.24/modules.softdep
./5.15.24/modules.order
./5.15.24/modules.builtin.modinfo
./5.15.24/modules.builtin.bin
./5.15.24/modules.dep
./5.15.24/modules.alias.bin
./5.15.24/modules.builtin
/home/alientek/ATK-DLMP135/alientek_linux/linux-5.15.24
编译完成请查看tmp目录
alientek@alientek:~/ATK-DLMP135/alientek_linux/linux-5.15.24$
```

Figure 2.5-2 Linux source code compilation result

If you need to reconfigure the default kernel, please enter the following command to open the kernel menuconfig menu for configuration selection:

```
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- menuconfig
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- savedefconfig
cp defconfig arch/arm/configs/stm32mp1_atk_defconfig
```

After the modification, we replaced the original configuration file arch/arm/configs/stm32mp1_atk_defconfig with the modified file defconfig.

We can view the files in the tmp directory, as well as the compiled kernel uImage image and device tree file:

Table 2 shows the post-compile image files of the Linux system.

Mirror name	Mirror effect	Mirror files and paths
Kernel	Linux image	uImage
Device tree	Common peripheral functions of the development board, such as RGB display, TF card, etc.	stm32mp135d-atk.dtb
	Support HDMI interface display	stm32mp135d-atk-hdmi.dtb
	Support WIFI and Bluetooth	stm32mp135d-atk-wifi-bluetooth.dtb
Kernel module	The modules that need to be loaded after the kernel starts are all in this compressed package	modules.tar.bz2

- Packaging of Linux image files

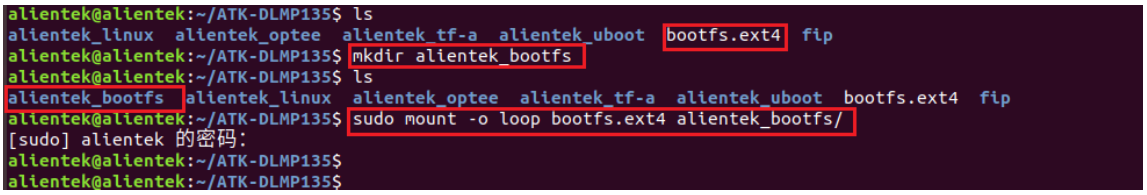
Previously, the kernel image uImage, the device tree file, and the kernel driver module directory have been compiled and generated. Now, we will introduce the process of packaging them together to form the Linux kernel firmware file bootfs.ext4, and finally burn it onto the development board.

In the factory system image provided with the development board materials, the bootfs.ext4 image file has been prepared. We can directly use it. The path is STM32MP135 development board -> Development board CD A - Basic materials -> 08_system_image -> 01_Factory_file_system -> 01, STM32CubeProg firmware burning package -> bootfs.ext4.

First, we copy the factory image file bootfs.ext4 to the ATK-DLMP135 directory in Ubuntu. Then, create a directory named "alientek_bootfs", and mount bootfs.ext4 to the alientek_bootfs directory. The command is as follows:

```
// Create the "alienek_bootfs" directory
mkdir  alientek_bootfs
// Mount the bootfs.ext4 file system to the alientek_bootfs directory
sudo  mount  -o  loop  bootfs.ext4  alientek_bootfs/
```

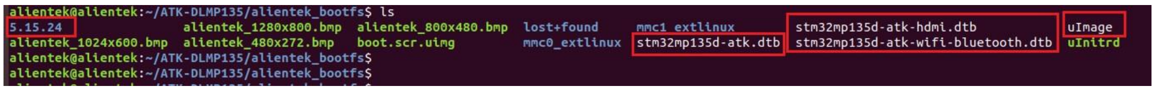
The running result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135$ ls
alientek_linux  alientek_optee  alientek_tf-a  alientek_uboot  bootfs.ext4  fip
alientek@alientek:~/ATK-DLMP135$ mkdir alientek_bootfs
alientek@alientek:~/ATK-DLMP135$ ls
alientek_bootfs  alientek_linux  alientek_optee  alientek_tf-a  alientek_uboot  bootfs.ext4  fip
alientek@alientek:~/ATK-DLMP135$ sudo mount -o loop bootfs.ext4 alientek_bootfs/
[sudo] alientek 的密码:
alientek@alientek:~/ATK-DLMP135$
alientek@alientek:~/ATK-DLMP135$
```

Figure 2.5-3 Mount the bootfs.ext4

After successful mounting, a Linux kernel image file that has been pre-made by the factory system will appear in the alientek_bootfs directory.



```
alientek@alientek:~/ATK-DLMP135/alientek_bootfs$ ls
5.15.24                  alientek_1280x800.bmp  alientek_800x480.bmp  lost+found    mmc1_extlinux          stm32mp135d-atk-hdmi.dtb            uImage
alientek_1024x600.bmp    alientek_480x272.bmp   boot.scr.uimg         mmc0_extlinux  stm32mp135d-atk.dtb  stm32mp135d-atk-wifi-bluetooth.dtb  uInitrd
alientek@alientek:~/ATK-DLMP135/alientek_bootfs$
alientek@alientek:~/ATK-DLMP135/alientek_bootfs$
```

Figure 2.5-4 Check the mounted alientek_bootfs directory

All we need to do is to replace the pre-compiled factory Linux image file we generated earlier with the one in the alientek_bootfs directory. Run the following command to perform the replacement:

```
// Delete all kernel modules, image files and device trees
sudo  rm  stm32mp135d-atk*  uImage  5.15.24/ -rf
// Copy all files to the "bootfs" directory
sudo  cp  ../alientek_linux/linux-5.15.24/tmp/*  ./
// Extract the module to the current directory
sudo  tar  -axvf  modules.tar.bz2
//delete  modules.tar.bz2 sudo  rm  modules.tar.bz2
// Go to the parent directory  cd ..
// Finally, unmount the device.
sudo  umount  alientek_bootfs
```

At this point, the bootfs.ext4 image file already contains the latest compiled Linux kernel image. In the next chapter, we will introduce how to use the factory source code image file compiled in this chapter and burn it onto the development board.

2.6 Buildroot File System Compilation

Buildroot is a very lightweight tool that can quickly build embedded Linux systems. In contrast, Yocto has more powerful functions and a steeper learning curve, requiring more learning and configuration work. Busybox is also relatively simple and easy to use, but its build process requires manually writing Makefiles.

Overall, compared to Busybox and Yocto, Buildroot is more lightweight and easier to use, making it easier to build and customize the system. If your project has strict requirements for build time and you need to quickly build a lightweight embedded Linux system, then Buildroot is a great choice.

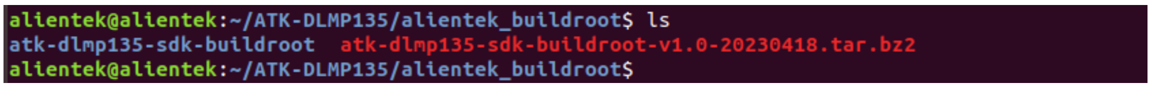
First, install the Ubuntu environment package and enter the following command:

```
sudo apt install expect libssl-dev
```

Create a directory named "alientek_buildroot" in the ATK-DLMP135 directory of Ubuntu. Then, copy the ALIENTEK factory Buildroot source code package atk-dlmp135-sdk-buildroot-v1.0-20230418.tar.bz2 (modified based on the official buildroot-2022.03 version) to this alientek_buildroot directory. The decompression command is as follows: (Please confirm the version of the source code you have downloaded. Do not directly copy the following decompression command)

```
tar -xjf atk-dlmp135-sdk-buildroot-v1.0-20230418.tar.bz2
```

The running result is shown in the following figure:



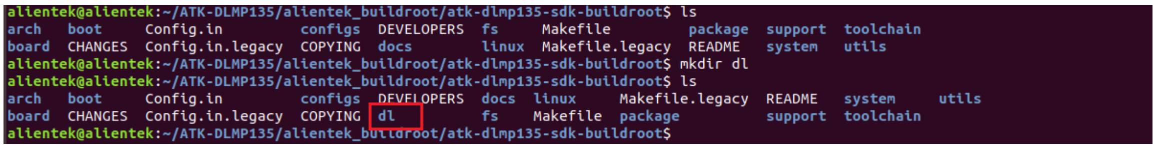
```
alientek@alientek:~/ATK-DLMP135/alientek_buildroot$ ls
atk-dlmp135-sdk-buildroot  atk-dlmp135-sdk-buildroot-v1.0-20230418.tar.bz2
alientek@alientek:~/ATK-DLMP135/alientek_buildroot$
```

Figure 2.6-1 The decompressed buildroot source code

After successful decompression, a directory named "atk-dlmp135-sdk-buildroot" will be generated in the current directory. Go to this directory to enter the source code directory of buildroot and create a "dl" directory. Run the following command:

```
mkdir dl
```

The creation result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ ls
arch   boot     Config.in          configs  DEVELOPERS  fs     Makefile         package  support  toolchain
board  CHANGES  Config.in.legacy   COPYING  docs        linux  Makefile.legacy  README   system   utils
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ mkdir dl
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ ls
arch   boot     Config.in          configs  DEVELOPERS  docs  linux     Makefile.legacy  README   system     utils
board  CHANGES  Config.in.legacy   COPYING  dl          fs    Makefile  package          support  toolchain
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$
```

Figure 2.6-2 Create the "dl" directory

After creating the "dl" directory, copy the "dl.tar.gz" file to the Ubuntu system. The path of this file is located at: STM32M P135 Development Board   Development Board CD A - Basic Materials →01_codes→06, buildroot Download Source Code Package. As shown in the following figure:

dl.tar.gz   2023/4/18 9:23   GZ 文件   711,735 KB

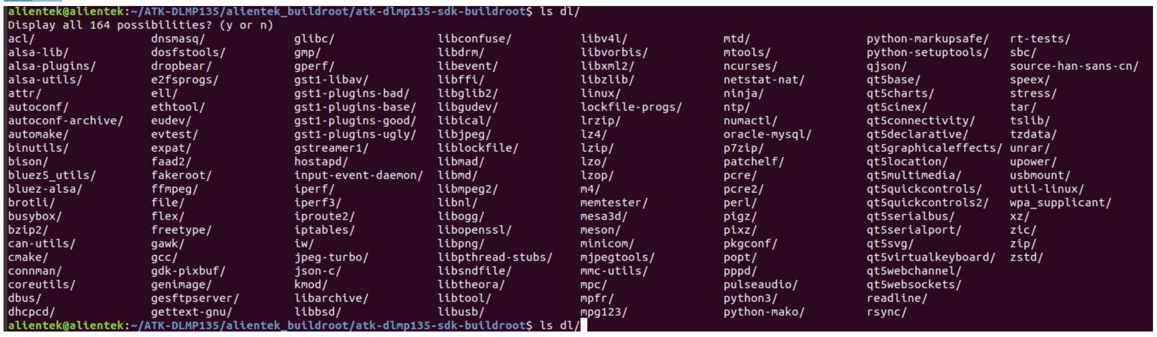
Figure 2.6-3 The directory of the dl compressed package

The author copied the file dl.tar.gz to the home directory, so the following command was used to extract it (note that the dl directory needs to be extracted):

```
tar -axf ~/dl.tar.gz -C dl
```

-C: Indicates decompression to the "dl" directory.

The decompression process is as shown in the following image (check the "dl" directory):



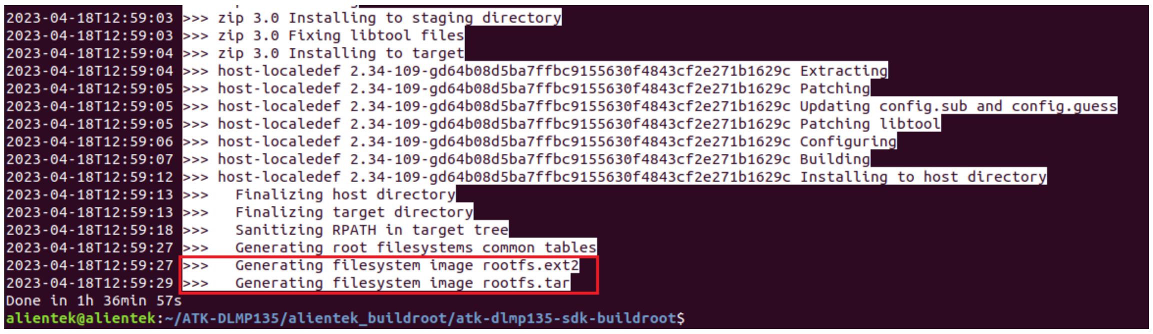
```
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ ls dl/
Display all 164 possibilities? (y or n)
acl/                 dnsmasq/             glibc/               libconfuse/          libv4l/              mtd/                 python-markupsafe/   rt-tests/
alsa-lib/            dosfstools/          gmp/                 libdrm/              libvorbis/           mtools/              python-setuptools/   sbc/
alsa-plugins/        dropbear/            gperf/               libevent/            libxml2/             ncurses/             qjson/               source-han-sans-cn/
alsa-utils/          e2fsprogs/           gst1-libav/          libffi/              libzlib/             netstat-nat/         qt5base/             speex/
attr/                ell/                 gst1-plugins-bad/    libglib2/            linux/               ninja/               qt5charts/           stress/
autoconf/            ethtool/             gst1-plugins-base/   libgudev/            lockfile-progs/      ntp/                 qt5cinex/            tar/
autoconf-archive/    eudev/               gst1-plugins-good/   libical/             lrzip/               numactl/             qt5connectivity/     tslib/
automake/            evtest/              gst1-plugins-ugly/   libjpeg/             lz4/                 oracle-mysql/        qt5declarative/      tzdata/
binutils/            expat/               gstreamer1/          liblockfile/         lzip/                p7zip/               qt5graphicaleffects/ unrar/
bison/               faad2/               hostapd/             libmad/              lzo/                 patchelf/            qt5location/         upower/
bluez5_utils/        fakeroot/            input-event-daemon/  libmd/               lzop/                pcre/                qt5multimedia/       usbmount/
bluez-alsa/          ffmpeg/              iperf/               libmpeg2/            m4/                  pcre2/               qt5quickcontrols/    util-linux/
brotli/              file/                iperf3/              libnl/               memtester/           perl/                qt5quickcontrols2/   wpa_supplicant/
busybox/             flex/                iproute2/            libogg/              mesa3d/              pigz/                qt5serialbus/        xz/
bzip2/               freetype/            iptables/            libopenssl/          meson/               pixz/                qt5serialport/       zic/
can-utils/           gawk/                iw/                  libpng/              minicom/             pkgconf/             qt5svg/              zip/
cmake/               gcc/                 jpeg-turbo/          libpthread-stubs/    mjpegtools/          popt/                qt5virtualkeyboard/  zstd/
connman/             gdk-pixbuf/          json-c/              libsndfile/          mmc-utils/           pppd/                qt5webchannel/
coreutils/           genimage/            kmod/                libtheora/           mpc/                 pulseaudio/          qt5websockets/
dbus/                gesftpserver/        libarchive/          libtool/             mpfr/                python3/             readline/
dhcpcd/              gettext-gnu/         libbsd/              libusb/              mpg123/              python-mako/         rsync/
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ ls dl/
```

Figure 2.6-4 Check the "dl" directory

Then you can run the following command to complete the compilation of the Buildroot file system. The command is as follows:

```
make  alientek_dlmp135_defconfig
./utils/brmake  or  make
```

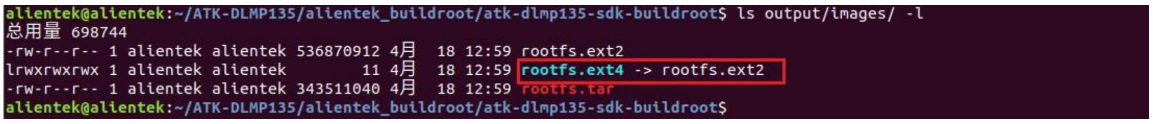
The brmake compilation command does not provide detailed print information, while the make compilation command does. The author has already compiled it before and did not want to see the detailed print information, so they used the bramke command (if there are compilation errors, you can use make for compilation). The running result is shown in the following figure:



```
2023-04-18T12:59:03 >>> zip 3.0 Installing to staging directory
2023-04-18T12:59:03 >>> zip 3.0 Fixing libtool files
2023-04-18T12:59:04 >>> zip 3.0 Installing to target
2023-04-18T12:59:04 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Extracting
2023-04-18T12:59:05 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Patching
2023-04-18T12:59:05 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Updating config.sub and config.guess
2023-04-18T12:59:05 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Patching libtool
2023-04-18T12:59:06 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Configuring
2023-04-18T12:59:07 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Building
2023-04-18T12:59:12 >>> host-localedef 2.34-109-gd64b08d5ba7ffbc9155630f4843cf2e271b1629c Installing to host directory
2023-04-18T12:59:13 >>>   Finalizing host directory
2023-04-18T12:59:13 >>>   Finalizing target directory
2023-04-18T12:59:18 >>>   Sanitizing RPATH in target tree
2023-04-18T12:59:27 >>>   Generating root filesystems common tables
2023-04-18T12:59:27 >>>   Generating filesystem image rootfs.ext2
2023-04-18T12:59:29 >>>   Generating filesystem image rootfs.tar
Done in 1h 36min 57s
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$
```

Figure 2.6-5 Run results

Then, a "rootfs.ext4" and "rootfs.tar" file will be generated in the "output/images/" directory under the source code directory of buildroot. The result is shown in the following figure:



```
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$ ls output/images/ -l
总用量 698744
-rw-r--r-- 1 alientek alientek 536870912 4月  18 12:59 rootfs.ext2
lrwxrwxrwx 1 alientek alientek        11 4月  18 12:59 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 alientek alientek 343511040 4月  18 12:59 rootfs.tar
alientek@alientek:~/ATK-DLMP135/alientek_buildroot/atk-dlmp135-sdk-buildroot$
```

Figure 2.6-6 View the results

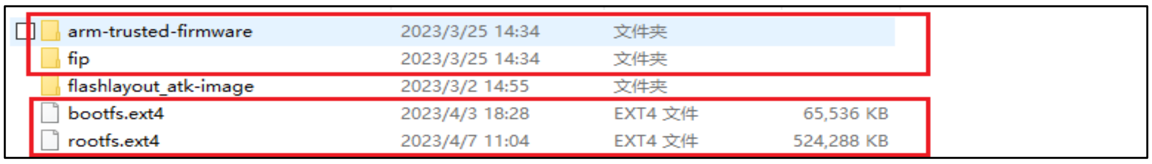
Chapter 3. Factory system image burning

The factory system image package for the ATK-DLMP135 development board is located in the STM32MP135 development board -> Development Board CD A - Basic Data -> 08_system_image -> 01_Factory_System_Image -> 01, STM32CubeProg firmware burning package.

We can follow the system burning steps in Chapter 2 of "[ALIENTEK] STM32MP135 Quick Test V1.X" to perform the burning.

Here, we will replace the image file compiled in the previous chapter to the corresponding directory of 01_STM32CubeProg_firmware_burning_package, and then re-burn. This process is relatively simple. Please make a backup of the original files to restore the default factory system.

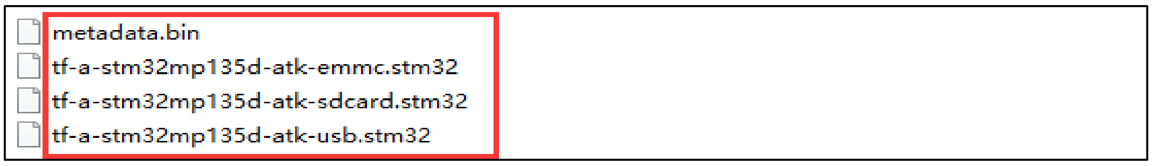
The following involve the arm-trusted-firmware directory, fip directory, bootfs.ext4 file and rootfs.ext4.



```
arm-trusted-firmware    2023/3/25 14:34    文件夹
fip                     2023/3/25 14:34    文件夹
flashlayout_atk-image   2023/3/2 14:55     文件夹
bootfs.ext4             2023/4/3 18:28     EXT4 文件     65,536 KB
rootfs.ext4             2023/4/7 11:04     EXT4 文件    524,288 KB
```

Figure 2.6-1 STM32CubeProg firmware burning package

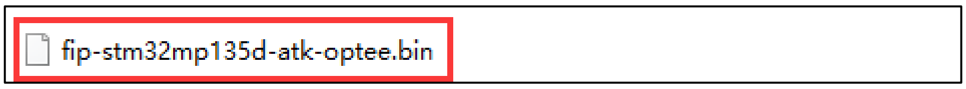
(1) First, replace the four image files (metadata.bin, tf-a-stm32mp 135d-atk-emmc.stm32, tf-a-stm32mp135d-atk-sdcard.stm32, and tf-a-stm32mp135d-atk-usb.stm32) generated from the TF-A source code compiled in Section 2.1 to the same-named files in the arm-trusted-firmware directory;



```
metadata.bin
tf-a-stm32mp135d-atk-emmc.stm32
tf-a-stm32mp135d-atk-sdcard.stm32
tf-a-stm32mp135d-atk-usb.stm32
```

Figure 2.6-2 The TF-A image in the "arm-trusted-firmware" directory

(2) Replace the FIP image file "fip-stm32 mp135d-atk-optee.bin" obtained after packaging the OP-TEE and U-Boot firmware using the fiptool tool in Section 2.4 with the same-named file in the "fip" directory;

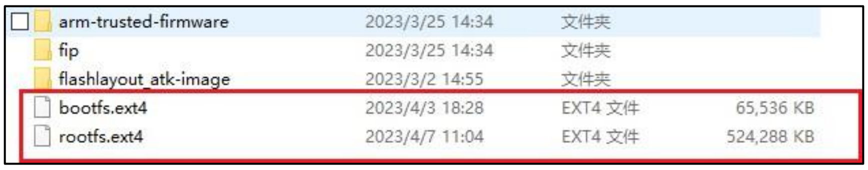


```
fip-stm32mp135d-atk-optee.bin
```

Figure 2.6-3 FIP image file

(3) Replace the bootfs.ext4 firmware file that was packaged in Section 2.5 with the same-named file in the firmware burning package directory of 01 and STM32CubeProg.

(4) Replace the rootfs.ext4 firmware file that was compiled in Section 2.6 with the same-named file in the firmware burning package directory of 01 and STM32CubeProg.



```
arm-trusted-firmware    2023/3/25 14:34    文件夹
fip                     2023/3/25 14:34    文件夹
flashlayout_atk-image   2023/3/2 14:55     文件夹
bootfs.ext4             2023/4/3 18:28     EXT4 文件     65,536 KB
rootfs.ext4             2023/4/7 11:04     EXT4 文件    524,288 KB
```

Figure 2.6-4 bootfs.ext4 firmware

Just reflash the factory system firmware that you have compiled yourself.