MediaTek OpenWrt User Manual

Version: 1.0

Release date: 2023-06-27

Version History

Version	Date	Author	Reviewer	Description
1.0	2023-06-27	RogerCC Lin	Chung-Ping Lai	Official release

Table of Contents

1 About MediaTek OpenWrt SDK

1.1 Introduction to OpenWrt/LEDE

[From https://openwrt.org/about]
OpenWrt is a highly extensible GNU/Linux distribution for embedded devices (typically wireless routers). Unlike many other distributions for these routers, OpenWrt is built from the ground up to be a full-featured, easily modifiable operating system for your router. In practice, this means that you can have all the features you need with none of the bloat, powered by a Linux kernel that's more recent than most other distributions.

Instead of trying to create a single, static firmware, OpenWrt provides a fully writable filesystem with optional package management. This frees you from the restrictions of the application selection and configuration provided by the vendor and allows you to use packages to customize an embedded device to suit any application. For developers, OpenWrt provides a framework to build an application without having to create a complete firmware image and distribution around it. For users, this means the freedom of full customization, allowing the use of an embedded device in ways the vendor never envisioned.

Refer to https://openwrt.org/about for more details.

1.2 MediaTek's OpenWrt SDK Overview

MediaTek starts to support OpenWrt project since 2013. We provide customized SDK based on stable releases of OpenWrt or LEDE.

To provide better performance and stability, MediaTek replaces some of OpenWrt's components, including some applications and kernel drivers (Ethernet, USB, Wi-Fi, etc.).

Currently, MediaTek's OpenWrt SDK uses **OpenWrt 21.02 stable branch**, and it supports the following MediaTek chips:

- MT7986
- MT7981
- MT7988

For the bootloader of MT798x, MediaTek provides **ATF** and **Uboot-upstream**, which are the source code of MT798x bootloader (bl2.img and fip.bin).

The diagram below shows MediaTek OpenWrt Software components: **ATF**, **U-Boot (Uboot-upstream),** and **OpenWrt**.

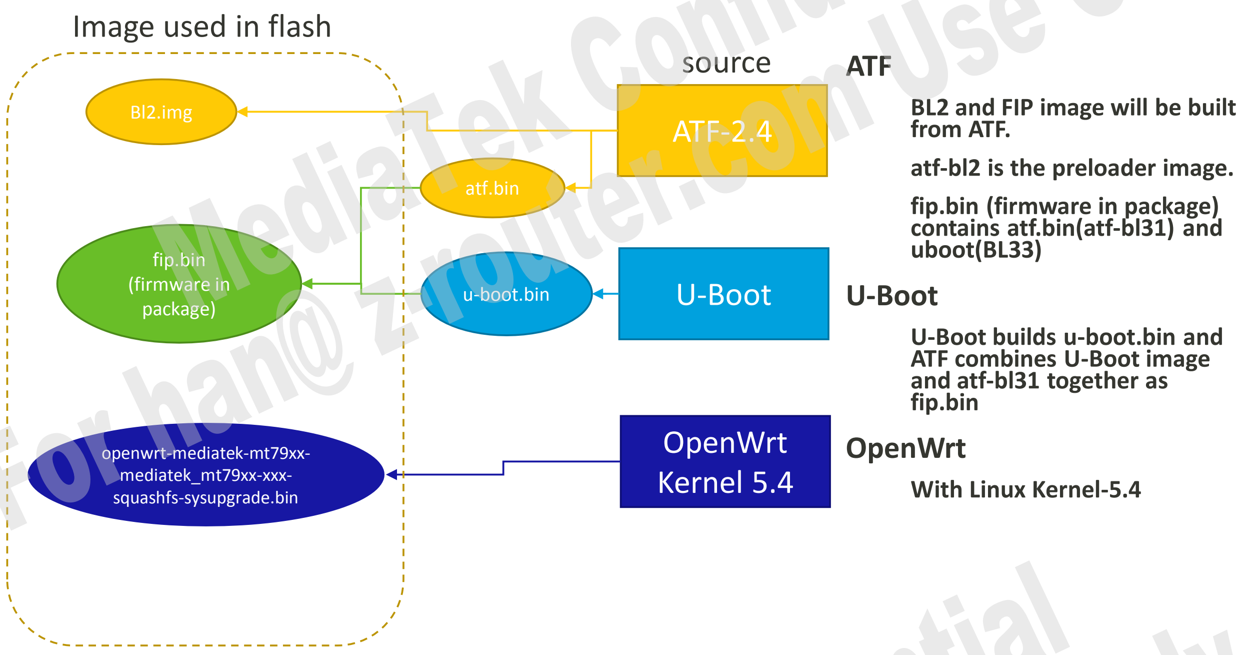
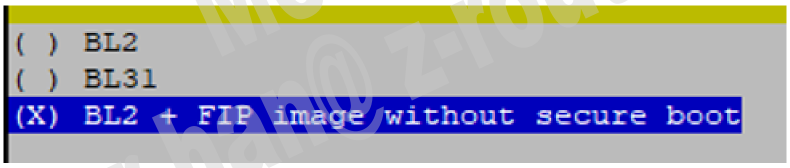


Image used in flash
Bl2.img
atf.bin
fip.bin (firmware in package)
u-boot.bin
openwrt-mediatek-mt79xx-mediatek_mt79xx-xxx-squashfs-sysupgrade.bin
source
ATF-2.4
U-Boot
OpenWrt Kernel 5.4
ATF
BL2 and FIP image will be built from ATF.
atf-bl2 is the preloader image.
fip.bin (firmware in package) contains atf.bin(atf-bl31) and uboot(BL33)
U-Boot
U-Boot builds u-boot.bin and ATF combines U-Boot image and atf-bl31 together as fip.bin
OpenWrt
With Linux Kernel-5.4

There are three images used in the MT798x device:

1. **bl2.img** is the first image loaded (by On-Chip BROM) after the device powers on.
2. The second image, **fip.bin** is composed of atf.bin and u-boot.bin (**FIP** stands for "**Firmware in Package**"). The last one is the **OpenWrt** image (aka Kernel image).

To build a FIP image(**fip.bin**), please compile Uboot-upsteam for u-boot.bin, then compile atf by selecting "BL2+FIP image (without secure boot)" in atf menu option and specify the path of u-boot.bin.



```
( ) BL2
( ) BL31
(X) BL2 + FIP image without secure boot
```

Refer to the U-Boot and ATF user manuals for more details about how to build the bootloaders.

1.3 Get MediaTek's OpenWrt Source

Get OpenWrt 21.02 source from Git server.

```
git clone --branch openwrt-21.02 https://git.openwrt.org/openwrt/openwrt.git
```

Untar Wi-Fi package "mtk-wifi-mt79xx-xxxxxxxx.tar.xz". Note that underlined words need to be replaced.

```
tar -Jxvf mtk-wifi-mt79xx-xxxxxxxx.tar.xz
```

Import Wi-Fi packages into OpenWrt SDK source. Note that underlined words need to be replaced.

```
cp -rf mtk-wifi-mt79xx/* openwrt/
```

Change to the openwrt folder.

```
cd openwrt/
```

Add MTK feeds.

```
echo "src-git mtk_openwrt_feed
https://git01.mediatek.com/openwrt/feeds/mtk-openwrt-feeds" >> feeds.conf.default
```

2 Abbreviations

Table 2-1. Abbreviations

Abbreviation	Explanation
AP	Access Point
ATF	Arm Trusted Firmware
DTS	Device Tree Source
eMMC	Embedded MultiMediaCard
FIP	Firmware in Package
MII	Media-Independent Interface
NAT	Network Address Translation
NMBM	NAND Mapping Block Management
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PWM	Pulse Width Modulation
SoC	System-on-Chip
UART	Universal Asynchronous Receiver and Transmitter
UI	User Interface

3 Getting Started with MTK LEDE OpenWrt

3.1 Compiler Machine Requirement

Ubuntu 18.04 or higher is required to compile the MediaTek OpenWrt SDK.
Install the following packages:

OpenWrt

```
apt-get install -y uuid-dev
```

Toolchain

```
apt-get install -y gcc-aarch64-linux-gnu
apt-get install -y clang-6.0
```

Or you can get Ubuntu 18.04 docker image from MTK support and install it.

3.2 OpenWrt General Source Structure

After the source code is downloaded, the working directory is under "openwrt/".

Here are the folders you can find in "openwrt/".

autobuild/ : Mediatek's build script for OpenWrt (for the first time compiling). See 3.6.
config/ : configuration files for menuconfig
include/ : makefile configuration files
package/ : packages makefile and configuration. See 3.7.
scripts/ : miscellaneous scripts used throughout the build process
target/ : makefile and configuration for building imagebuilder, kernel, sdk and the toolchain built by buildroot. See 3.8 for more details.
toolchain/ : makefile and configuration for building the toolchain
tools/ : miscellaneous tools used throughout the build process

3.3 Run autobuild Script (Only First-time Build)

If compiling the OpenWrt SDK for the first time, it is highly recommended that you use the autobuild script:

```
autobuild/mt79xx-xxxxxx/lede-branch-build-sanity.sh
```

If there is a build failure and you want to re-run the autobuild script, use the following command to clean:

```
autobuild/clean-staging.sh
```

3.4 Make Commands

Make the whole SDK

```
make V=s
```

Select SDK package menuconfig

```
make menuconfig
```

Select Linux kernel menuconfig

```
make kernel_menuconfig
```

Make kernel only

```
make V=s target/linux/install
```

Make package only

```
make V=s package/your_package_path/your_package/compile
make V=s package/your_package_path/your_package/install
```

(underlined characters should be replaced with appropriate words, and the same below)

#example:

```
make V=s package/mtk/applications/wificonf/compile
make V=s package/mtk/applications/wificonf/install
```

3.5 Image Location and How to Load Image

The image is located in

```
bin/targets/mediatek/mt798x/
```

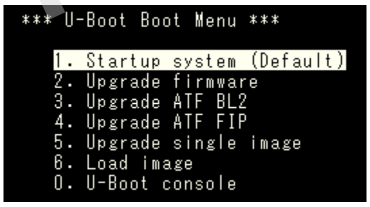
Squash image

```
bin/targets/mediatek/mt798x/xxxxxxxx-squashfs-sysupgrade.bin
```

Note:

- The underlined characters should be replaced with appropriate part names.

You can update this image by using the U-Boot menu's "**Upgrade firmware**" option or by using upgrade page in the web UI.



```
*** U-Boot Boot Menu ***

    1. Startup system (Default)
    2. Upgrade firmware
    3. Upgrade ATF BL2
    4. Upgrade ATF FIP
    5. Upgrade single image
    6. Load image
    0. U-Boot console
```

Option 1: **Boot the OpenWrt image from flash**
Option 2: **Upgrade OpenWrt image**
Option 3: **Upgrade bl2.img**
Option 4: **Upgrade fip.bin** (atf-bl31 + U-Boot)
Option 5: **Upgrade a single image**. Please check out the "**Image for generating single image**" part below.
Option 6: **Load OpenWrt ramdisk image to DRAM and then execute**. Please check out "**Ramdisk image**" section
Option 0: **Enter U-Boot command line prompt**

Ramdisk image

You can update this image by using the U-Boot menu's "**Load image**" option. Note that using this option does not write the image to flash.

Image for generating a single image

```
bin/targets/mediatek/mt798x/xxxxxxxx-squashfs-factory.bin
```

To compose a single image, you need "bl2.img", "fip.bin" and this image together.

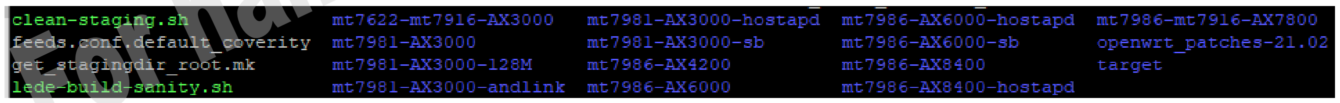
You can upgrade a single image via the U-Boot menu's "**Upgrade single image**" option. (Do not use this option if you are not sure whether your image is a single image)

For generating single image, you can find "**single_img_wrapper**" tool in atf. See the following for the usage of the tool:

```
Usage:
  ./mk_image -p <platform>
             -d <flash device type>
             -c <partition config>
             -b <BL2 image>, default=bl2.img
             -r <RF image>
             -f <FIP image>, default=fip.bin
             -k <kernel image>
             -g <GPT table>
             -h <usage menu>
             -o <single image name>
example:
  ./mk_image.sh -p mt7986a -d emmc \
                -g GPT_EMMC-iap-20220125 \
                -f fip-iap-emmc-20220125.bin \
                -k OF_openwrt-mediatek-mt7986-mt7986a-ax6000-emmc-rfb-squashfs-sysupgrade.bin
  ./mk_image.sh -p mt7986a -d spim-nand \
                -b bl2-iap-snand-20220114.img \
                -f fip-snand-20220114.bin \
                -k OF_openwrt-mediatek-mt7986-mt7986a-ax6000-spim-nand-rfb-squashfs-factory.bin \
```

3.6 MediaTek's autobuild Script

The following files and folders are located in the Autobuild folder:



```
clean-staging.sh            mt7622-mt7916-AX3000    mt7981-AX3000-hostapd  mt7986-AX6000-hostapd  mt7986-mt7916-AX7800
feeds.conf.default_coverity mt7981-AX3000           mt7981-AX3000-sb       mt7986-AX6000-sb       openwrt_patches-21.02
get_stagingdir_root.mk      mt7981-AX3000-128M      mt7986-AX4200          mt7986-AX8400          target
lede-build-sanity.sh        mt7981-AX3000-andlink   mt7986-AX6000          mt7986-AX8400-hostapd
```

clearn-staging.sh: the script to restore your working directory to a clean state, such as revert patches. If you have already run the autobuild script once, run this script before rerunning the autobuild script.

openwrt_patches-21.02/: contain MTK patches for OpenWrt.

mt798x-xxxxxx/: this folder contains scripts/files/config required to build OpenWrt images. In each folder, there are

.config : the configuration file

lede-branch-build-sanity.sh : the script to auto-build

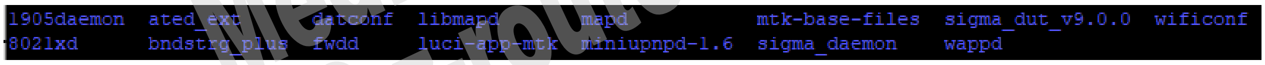
target/linux/mediatek/patches-5.4/ : this folder contains patches for the target, and the patches will be copied to "../target/linux/mediatek/patches-5.4/" before compiling

mt798x-xxxxxx-sb/: please note that the "**-sb**" suffix means "**secure boot**", which should be used when the "secure boot" is enabled in your target device.

3.7 Package

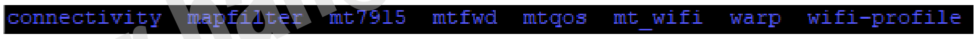
In the package folder, there are makefiles for packages and utils.

The package's source tarball is in **dl/**, and the makefiles of these MediaTek packages are located in "**package/mtk/",** below are some applications in mtk/applications.



```
1905daemon   ated_ext     datconf  libmapd      mapd           mtk-base-files  sigma_dut_v9.0.0  wificonf
8021xd       bndstrg_plus fwdd     luci-app-mtk miniupnpd-1.6  sigma_daemon    wappd
```

The following lists some drivers in mtk/drivers



```
connectivity  mapfilter  mt7915  mtfwd  mtqos  mt_wifi  warp  wifi-profile
```

3.8 Target

The target folder contains the descriptions and files to build the target:

Configuration Menu

Config.in: specify the menu option of the target.

Makefile

Makefile: top-level makefile of the target.

Kernel config

linux/mediatek/mt798x/config-5.4: the kernel configuration for this target. This configuration file will be applied to "build_dir/target-aarch64_cortex-a53_musl/linux-mediatek_mt798x/linux-5.4.xxx/.config".

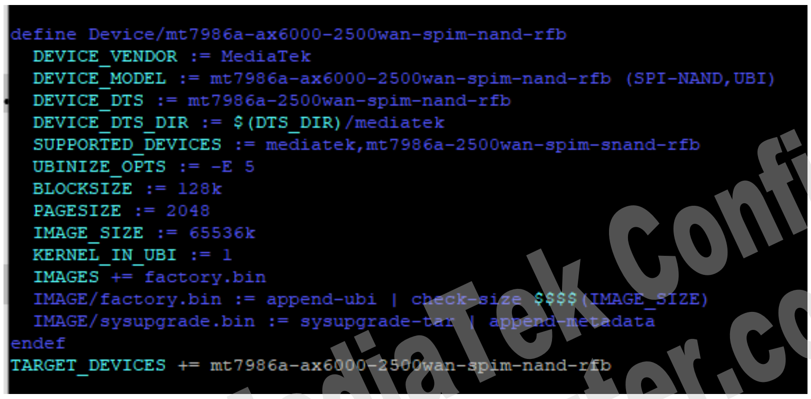
Kernel patches

linux/mediatek/patches-5.4/: contain MediaTek patch files for Linux kernel.

Target image description

linux/mediatek/image/mt798x.mk: specify how to build the target images, such as which dts file to be used and what kinds of images need to be generated.

Here is an example of MT7986 target description:



```
define Device/mt7986a-ax6000-2500wan-spim-nand-rfb
  DEVICE_VENDOR := MediaTek
  DEVICE_MODEL := mt7986a-ax6000-2500wan-spim-nand-rfb (SPI-NAND,UBI)
  DEVICE_DTS := mt7986a-2500wan-spim-nand-rfb
  DEVICE_DTS_DIR := $(DTS_DIR)/mediatek
  SUPPORTED_DEVICES := mediatek,mt7986a-2500wan-spim-snand-rfb
  UBINIZE_OPTS := -E 5
  BLOCKSIZE := 128k
  PAGESIZE := 2048
  IMAGE_SIZE := 65536k
  KERNEL_IN_UBI := 1
  IMAGES += factory.bin
  IMAGE/factory.bin := append-ubi | check-size $$$$(IMAGE_SIZE)
  IMAGE/sysupgrade.bin := sysupgrade-tar | append-metadata
endef
TARGET_DEVICES += mt7986a-ax6000-2500wan-spim-nand-rfb
```

This description specifies OpenWrt images in "/bin/targets/mediatek/mt7986/",

```
openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-initramfs-kernel.bin
```

```
openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-squashfs-factory.bin
openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-squashfs-sysupgrade.bin
```

The mt798x.mk target description also specifies the dts file. In this example, it is mt7986a-2500wan-spim-nand-rfb.dts

```
DEVICE_DTS := mt7986a-2500wan-spim-nand-rfb
```

3.9 Flash Type Supported by MT798x

MT798x supports three kinds of flash:

- SPI-NAND flash
- SPI-NOR flash
- eMMC (SD) flash

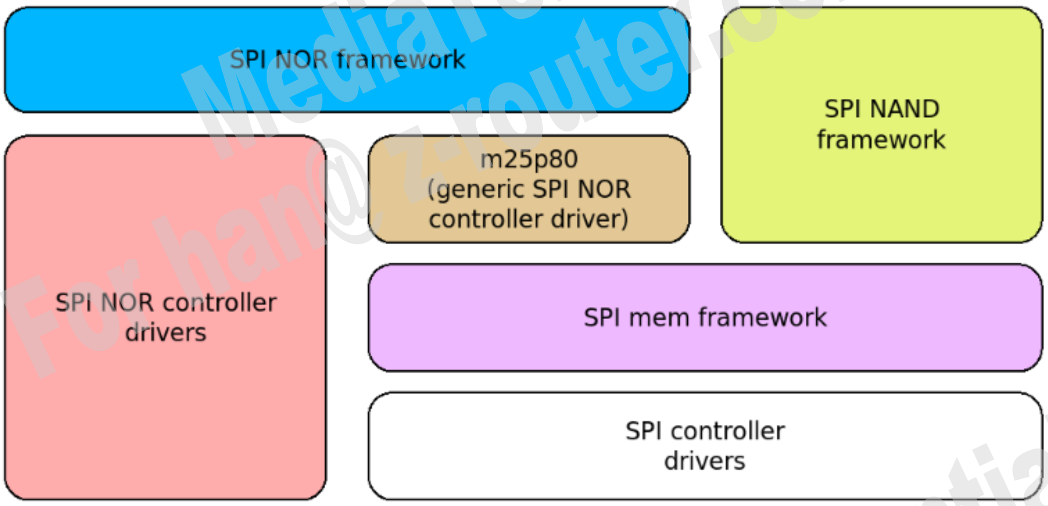
The following sections provide detail descriptions of the flash supported.

3.9.1 SPI-NAND

MT798x AP SoC can boot up from SPI-NAND flash, and it has two controllers for SPI-NAND flash:

SPI-NAND (mandatory):

"**SPI-NAND**" controller uses SPI interface to connect to SPI-NAND flash, it adopts SPI memory framework as in below diagram, SPI-NAND controller use **on-die ECC** for data correctness.



Reference: https://bootlin.com/blog/spi-mem-bringing-some-consistency-to-the-spi-memory-ecosystem/

SNFI-NAND controller (obsolete):

"**SNFI-NAND**" is a dedicated controller for SPI-NAND flash. It has an ECC engine and uses host ECC for data correctness. SNFI-NAND controller is obsolete.

History of MediaTek SPI-NAND:

Early MediaTek SoC MT7622 and MT7629 have only SNFI-NAND controller for SPI-NAND flash. They also have an SPI interface, but the SPI interface cannot be used to control flashes; they are only for non-flash SPI devices.

Since MT798x SDK uses Linux kernel 5.4.x which supports SPI memory framework, it can support controlling (SPI-NAND / SPI-NOR) flashes via SPI interface.

To distinguish from SNFI-NAND, the SPI-NAND controller is sometimes called SPIM-NAND (**SPI-M**emory-**NAND**). SPI-NAND controller has the advantage that it does not require to deal with ECC conversion. So SNFI-NAND is no longer necessarily used for MT798x SoC.

Part	SPI-NAND	SNFI-NAND
MT7622	N/A	Mandatory
MT7629	N/A	Mandatory
MT7986	Mandatory	Obsolete
MT7981	Mandatory	Obsolete
MT7988	Mandatory	Obsolete

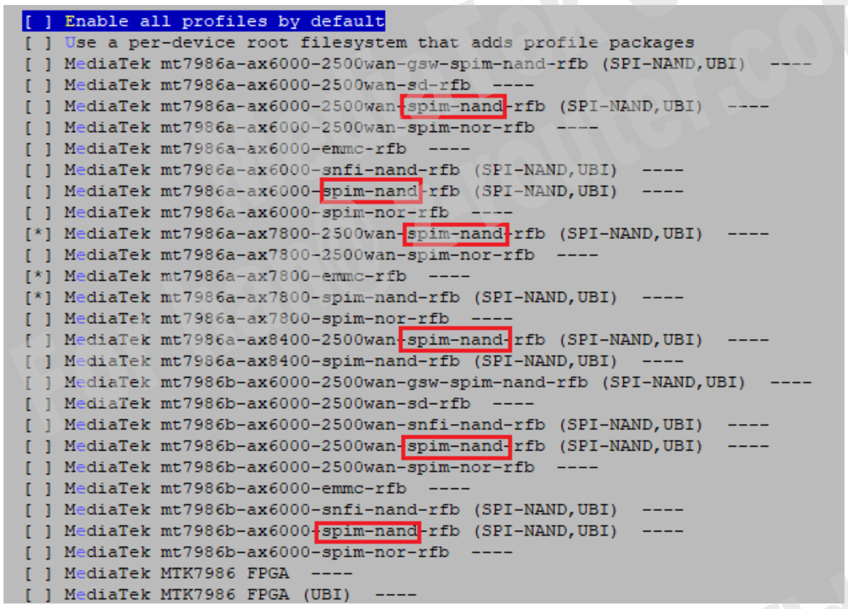
The controller described in this section is **SPI-NAND (SPIM-NAND)**.

3.9.1.1 Configurations

You can select the spim-nand target image in the following menuconfig option:

```
Target Devices  --->
```

Below is an example of MT7986 menu. The target with a "-spim-nand-" prefix is an image for SPI-NAND flash.



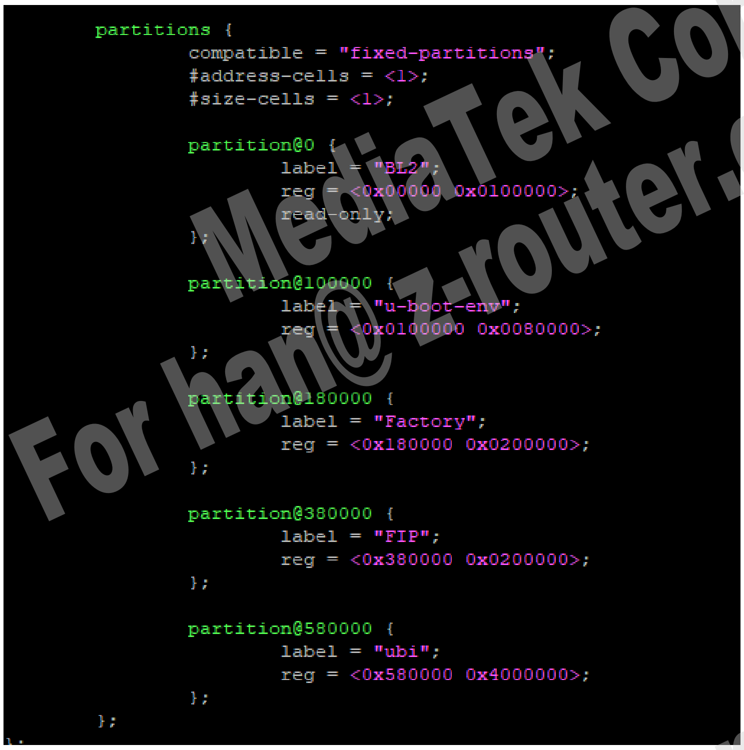
```
[ ] Enable all profiles by default
[ ] Use a per-device root filesystem that adds profile packages
[ ] MediaTek mt7986a-ax6000-2500wan-gsw-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax6000-2500wan-sd-rfb  ----
[ ] MediaTek mt7986a-ax6000-2500wan-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax6000-2500wan-spim-nor-rfb  ----
[ ] MediaTek mt7986a-ax6000-emmc-rfb  ----
[ ] MediaTek mt7986a-ax6000-snfi-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax6000-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax6000-spim-nor-rfb  ----
[*] MediaTek mt7986a-ax7800-2500wan-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax7800-2500wan-spim-nor-rfb  ----
[*] MediaTek mt7986a-ax7800-emmc-rfb  ----
[*] MediaTek mt7986a-ax7800-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax7800-spim-nor-rfb  ----
[ ] MediaTek mt7986a-ax8400-2500wan-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986a-ax8400-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-2500wan-gsw-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-2500wan-sd-rfb  ----
[ ] MediaTek mt7986b-ax6000-2500wan-snfi-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-2500wan-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-2500wan-spim-nor-rfb  ----
[ ] MediaTek mt7986b-ax6000-emmc-rfb  ----
[ ] MediaTek mt7986b-ax6000-snfi-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-spim-nand-rfb (SPI-NAND,UBI)  ----
[ ] MediaTek mt7986b-ax6000-spim-nor-rfb  ----
[ ] MediaTek MTK7986 FPGA  ----
[ ] MediaTek MTK7986 FPGA (UBI)  ----
```

3.9.1.2 Device Tree and Partition

You can find partition description in Linux device tree files below:

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

The following picture shows an example of a partition table that uses SPI-NAND.



```
        partitions {
                compatible = "fixed-partitions";
                #address-cells = <1>;
                #size-cells = <1>;

                partition@0 {
                        label = "BL2";
                        reg = <0x00000 0x0100000>;
                        read-only;
                };

                partition@100000 {
                        label = "u-boot-env";
                        reg = <0x0100000 0x0080000>;
                };

                partition@180000 {
                        label = "Factory";
                        reg = <0x180000 0x0200000>;
                };

                partition@380000 {
                        label = "FIP";
                        reg = <0x380000 0x0200000>;
                };

                partition@580000 {
                        label = "ubi";
                        reg = <0x580000 0x4000000>;
                };
        };
```

In case that partition table is modified in the device tree(dts/dtsi), the bootloader's partition description should also be adjusted. For example, the following U-Boot config describes the partitions of SPI-NAND (with NMBM):

```
CONFIG_MTDPARTS_DEFAULT="nmbm0:1024k(bl2),512k(u-boot-
env),2048k(factory),2048k(fip),65536k(ubi)"
```

For ATF, if a FIP's partition offset is changed, please also modify plat/mediatek/mt798x/bl2_boot_spim_nand.c

```
#define FIP_BASE                        0x380000
#define FIP_SIZE                         0x200000
```

3.9.1.3 e2p

The e2p (WiFi calibration data) binary is in the "Factory" partition, and you can use the below command to find out the mtd device for e2p.

```
cat /proc/mtd
```

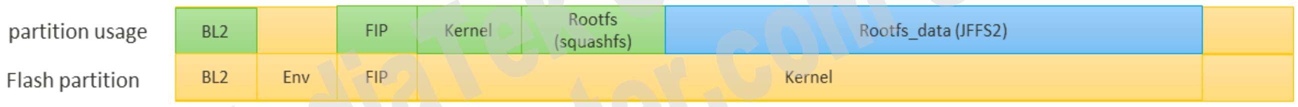
The e2p file (in the filesystem) is in.

```
/lib/firmware/e2p
```

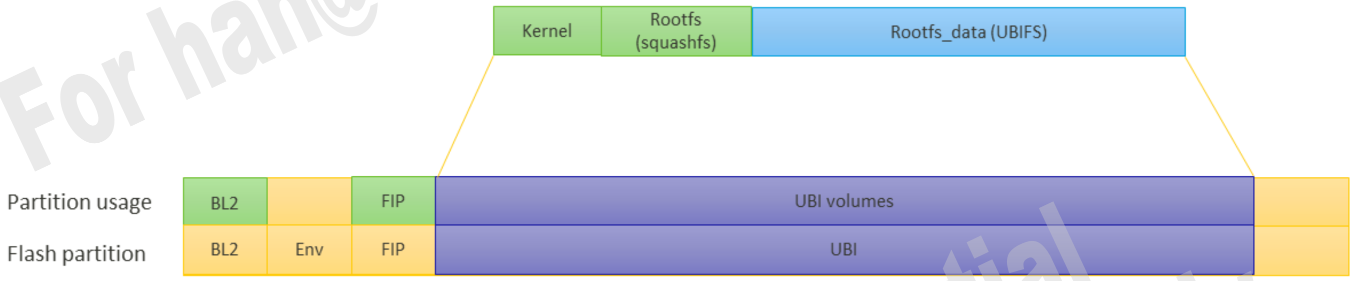
3.9.1.4 Filesystem for SPI-NAND

MediaTek changed the default filesystem of OpenWrt from JFFS2 to UBIFS.

The following diagram shows OpenWrt's original partition usage. The "kernel" partition was a "raw" partition, and the (original) OpenWrt image was composed of the kernel binary and squashFS rootfs image together. After OpenWrt boots up, it will mount the rest of the "kernel" partition as "rootfs_data", which is a JFFS2 filesystem partition.



MediaTek had changed the Kernel partition format from "raw" to "UBI", as shown below. The partition name was also changed from "kernel" to "**ubi**". Kernel image / rootfs / rootfs_data were encapsulated as separate UBI volumes in the "ubi" partition. Please note that each volume in the "ubi" partition did not have a fixed address.

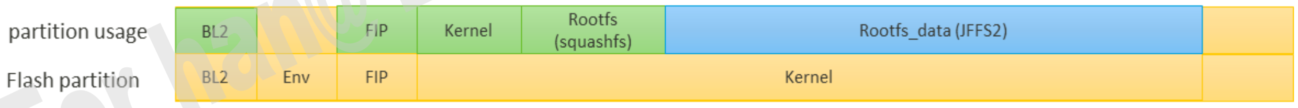


3.9.1.5 Bad Block Management

SPI-NAND flash may have bad block issues. MediaTek OpenWrt SDK uses **NMBM** (**N**AND **M**apping **B**lock **M**anagement) to handle bad blocks.

3.9.2 SPI-NOR

MT798x can boot from SPI-NOR flash. MT798x SPI-NOR controller also uses SPI interface and SPI-mem framework. For SPI-NOR flash, MediaTek SDK keeps the original OpenWrt partition usage:



3.9.2.1 Configurations

You can select "spim-nor" target image in menuconfig option:

```
 Target Devices  --->
```

The target with a "-spim-nor-" prefix is an image for SPI-NOR flash.

3.9.2.2 Device Tree and Partition

You can find partition description in Linux device tree files below:

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

In case that partition table is modified in the device tree(dts/dtsi), the bootloader's partition description should also be adjusted. For example, the following U-Boot config describes the partitions of SPI-NOR:

```
CONFIG_MTDPARTS_DEFAULT="nor0:256k(bl2),64k(u-boot-env),704k(factory),512k(fip),-(firmware)"
```

For ATF, if the FIP partition offset is changed, please also modify plat/mediatek/mt798x/bl2_boot_spim_nor.c

```
#define FIP_BASE_NOR    0x100000
#define FIP_SIZE_NOR    0x80000
```

3.9.2.3 e2p

e2p for SPI-NOR is the same as for SPI-NAND. See 3.9.1.3.

3.9.3 eMMC (SD)

MT798x supports booting from eMMC flash (or SD card), for eMMC. MT798x can also support eMMC v4.5 or eMMC v5.1. See the table below.

Part	eMMC v4.5	eMMC v5.1	SD
MT7986A		V	
MT7986B / MT7986C	V		
MT7981	V		V
MT7988	V	V	V

3.9.3.1 Configurations

You can select the eMMC or SD target image in the menuconfig option.

```
Target Devices  --->
```

The target with an "-emmc-" prefix is an image for eMMC flash.
The target with an "-sd-" prefix is an image for SD card.

3.9.3.2 Partition and GPT Table

Unlike SPI-NAND or SPI-NOR, eMMC did not have a fixed partition description in the device tree. It uses GPT table to describe the partition.
The tool for GPT table is in "tools/dev/**gpt_editor**" in **ATF**; please check out Readme for the tool usage.

In case that atf partition offset is changed, please modify plat/mediatek/mt798x/bl2_boot_mmc.c

```
#define FIP_BASE                0x680000
#define FIP_SIZE                 0x200000
```

If an eMMC is used, the default filesystem is ext4.
If an SD card is used, the default filesystem is FAT.

3.9.3.3 e2p

The e2p (Wi-Fi calibration data) binary for eMMC flash or SD card is located in

```
/dev/mmcblk0p3
```

The e2p file (in the filesystem) is located in

```
/lib/firmware/e2p
```

3.9.3.4 MT7986 eMMC

There is only one eMMC controller for MT7986. For MT7986**A**, the eMMC controller is v5.1. The pins listed below are used.

GPIO50	EMMC_DATA_0
GPIO51	EMMC_DATA_1
GPIO52	EMMC_DATA_2
GPIO53	EMMC_DATA_3
GPIO54	EMMC_DATA_4
GPIO55	EMMC_DATA_5
GPIO56	EMMC_DATA_6
GPIO57	EMMC_DATA_7
GPIO58	EMMC_CMD
GPIO59	EMMC_CK
GPIO60	EMMC_DSL
GPIO61	EMMC_RSTB

For MT7986**B** or MT7986**C**, the eMMC controller is v4.5. The pins listed below are used.

GPIO22	PWM1	EMMC_RSTB
GPIO23	SNFI_CLK	EMMC_DATA_0
GPIO24	SNFI_MOSI	EMMC_DATA_1
GPIO25	SNFI_MISO	EMMC_DATA_2
GPIO26	SNFI_CS	EMMC_DATA_3
GPIO27	SNFI_HOLD	EMMC_DATA_4
GPIO28	SNFI_WP	EMMC_DATA_5
GPIO29	SNFI_CLK	EMMC_DATA_6
GPIO30	SNFI_MOSI	EMMC_DATA_7
GPIO31	SNFI_MISO	EMMC_CMD
GPIO32	SNFI_CS	EMMC_CK

4 SDK Component Introduction

4.1 Toolchain

In general cases, you do not need to configure the OpenWrt's toolchain.

Once you choose a platform, the corresponding toolchain configuration is generated for you. Everything you need to cross-compile a firmware for your platform has been integrated into the OpenWrt's built framework. Toolchains are built the first time you build the project, and then saved under "staging_dir" for further use.

Note that "make clean" does not clean up the "staging_dir", but "make distclean" does.

4.2 C/C++ Libraries

OpenWrt can be built with multiple C/C++ implementations.

- **Musl**: musl (https://www.musl-libc.org/) is a C standard library for operating systems based on the Linux kernel, released under the MIT License. It was developed by Rich Felker with the goal of writing a clean, efficient and standards-conformant libc implementation.

- **Glibc**: The GNU C Library (https://www.gnu.org/s/libc/) project provides the core libraries for the GNU system and GNU/Linux systems, as well as many other systems that use Linux as the kernel.

You can choose the implementations by "**make menuconfig**":

make menuconfig

```
#make menuconfig
  --> Advanced configuration options (for developers)
      --> Toolchain options
          --> C library implementations
              --> ( ) Use glibc
              --> (x) Use musl
```

By default, MT7986, MT7981 and MT7988 use "musl".

4.3 Peripherals

4.3.1 PCI

4.3.1.1 Kernel Configuration

```
CONFIG_PCIE_MEDIATEK_GEN3=y
```

4.3.1.2 Source Code

```
<linux>/drivers/pci/controller/pcie-mediatek-gen3.c
```

4.3.1.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of PCIe node:

```
&pcie0 {
        pinctrl-names = "default";
        pinctrl-0 = <&pcie0_pins>;
        status = "okay";
};
```

4.3.1.4 Usage and Testing

```
lspci -v
```

4.3.2 I2C

4.3.2.1 Kernel Configuration

```
CONFIG_I2C=y
CONFIG_I2C_MT65XX=y
```

4.3.2.2 Source Code

```
<linux>/drivers/i2c/busses/i2c-mt65xx.c
```

4.3.2.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of I2C node:

```
&i2c0 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c_pins>;
        status = "disabled";

        wm8960: wm8960@1a {
                compatible = "wlf,wm8960";
                reg = <0x1a>;
        };
};
```

4.3.2.4 Usage and Testing

Menuconfig:

```
#make menuconfig
  --> Kernel modules
      --> I2C support
         --> <*> kmod-i2c-core
  --> Utilities
      --> <*> i2c-tools
      --> <*> eepromer
  --> Kernel modules
      --> Other modules
          --> <*> kmod-rtc-ds1307
          --> <*> kmod-eeprom-at24
```

For example, to use 24c128 eeprom on i2c-0

```
  # Scenario 1:
  # Use i2c-tools read/write 24c128 eeprom directly (base on ioctl via i2c adaptor)
  # get i2c adaptor information
  i2cdetect -l
  # list devices on i2c-<adaptor-id>
  i2cdetect -y -r -a <adaptor-id>
  # ds-1307 at 0x68 (i2c-0)
  root@LEDE:/# i2cdetect -y -r -a 0
       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
  00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  # 24c128 eeprom read/write byte at 0x0010
  echo -n -e "\xFF" | eeprog -fx -16  -w 0x0010 /dev/i2c-0  0x50
  eeprog -fx -16 -r 0x0010 /dev/i2c-0  0x50

  # Scenario 2:
  # Use at24 i2c client driver to read/write 24c128 eeprom as a file

  # 24c128 eeprom read/write as a file
  echo 24c128 0x50 > /sys/bus/i2c/devices/i2c-0/new_device
  i2cdetect -y -r -a 0
  # at24 i2c client driver lock the 24c128 eeprom address
  root@LEDE:/# i2cdetect -y -r -a 0
       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
  00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

hexdump -v -C /sys/bus/i2c/drivers/at24/0-0050/eeprom
echo -n -e "\xAA\xBB\xCC" >  /sys/bus/i2c/drivers/at24/0-0050/eeprom
```

If the I2C device is single-byte addressing (most sensors, ds1307 rtc, 24c02 eeprom, I2C lcd …):
For example, to read/write register 0x10 directly on an I2C sensor

```
# get i2c adaptor information
i2cdetect -l
# list devices on i2c-<adaptor-id>
i2cdetect -y -r -a <adaptor-id>
# read register 0x10
i2cget -y 0 <adaptor-id> <device-id> 0x10
# write 100 to register 0x10
i2cset -f -y <adaptor-id> <device-id> 0x10 100
# dump all registers
i2cdump -y <adaptor-id> <device-id>
```

For example, to use ds1307 rtc on i2c-0

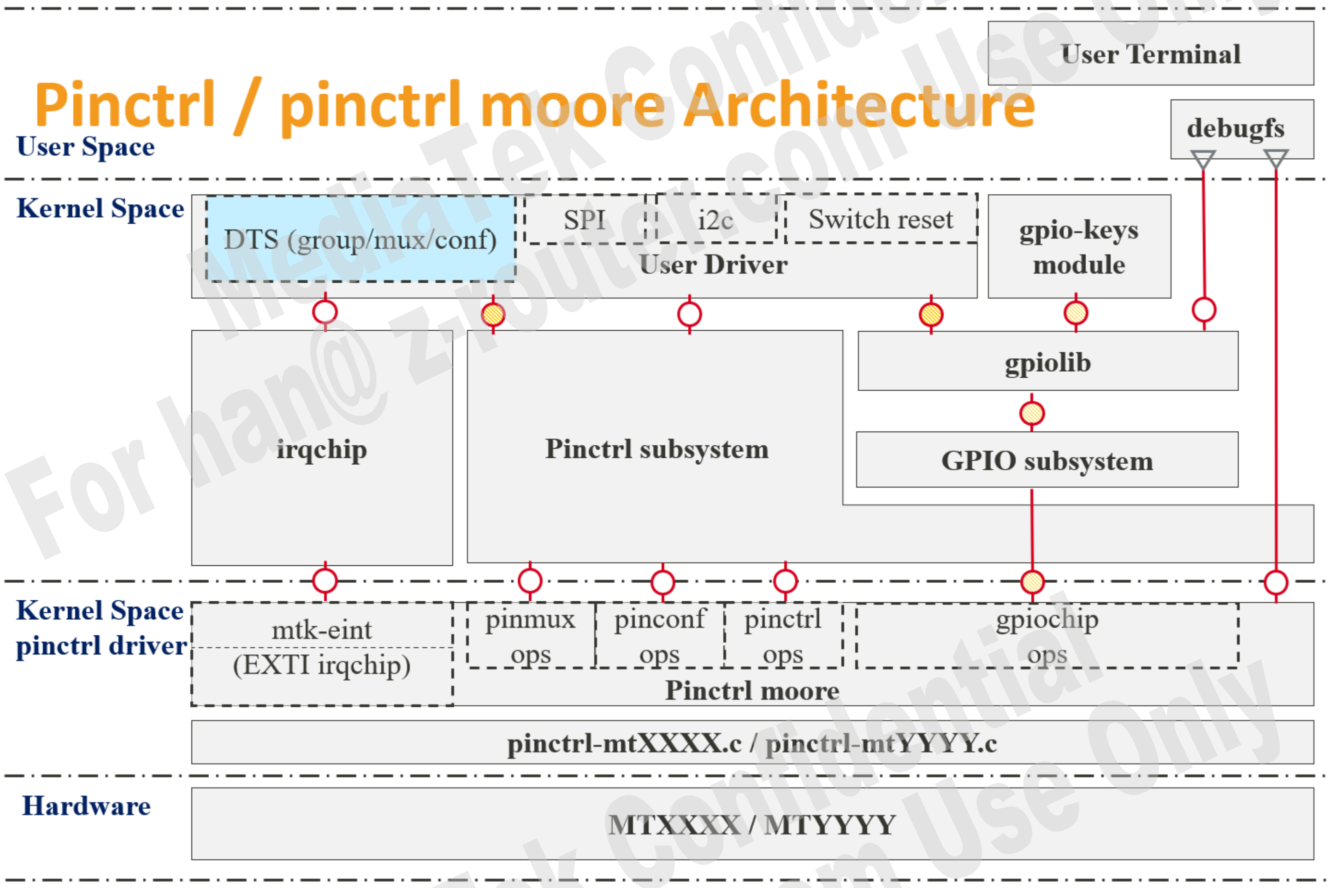
```
# 0x68: ds1307
i2cdetect -y -r -a 0
modprobe rtc-ds1307
echo ds1307 0x68 > /sys/bus/i2c/devices/i2c-0/new_device
cat /sys/bus/i2c/devices/i2c-0/0-0068/rtc/rtc0/time
# set system date to 2018.03.20-13:51:00
date -s 2018.03.20-13:51:00
# write system date to rtc
hwclock -w
# read date from rtc
hwclock -r
# use rtc to set system data
hwclock -s
date
```

4.3.3 GPIO

The following diagram shows the architecture of MT798x pinctrl subsystem, also known as pinctrl Moore or MTK pinctrl.

Note:

- drivers/pinctrl/mediatel/pinctrl-**moore**.c, drivers/pinctrl/mediatek/pinctrl-mt7xxx.*



pinmux: Pin Multiplexer

Allows for reusing the same pin for different purposes. For example, one pin can be used as the UART TX pin, I2C data line or GPIO due to different multiplexing.

pinconf: Pin Configuration

Configures the electronic properties of pins, such as pull-up, pull-down, driver strength settings.

Configures an individual pin as a specific name.

4.3.3.1 Kernel Configuration

```
CONFIG_PINCTRL_MTK=y
CONFIG_PINCTRL_MTK_MOORE=y
CONFIG_PINCTRL_MT7986=y
```

Note:

- MT7986, MT7981 and MT7988 use the same configuration “CONFIG_PINCTRL_MT7986”.

4.3.3.2 Source Code

See the file below for the name of the pin or group of pinctrl.

```
<linux>/drivers/pinctrl/mediatek/pinctrl-mt798x.c
```

4.3.3.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of GPIO node:

```
    pio: pinctrl@1001f000 {
             compatible = "mediatek,mt7986-pinctrl";
             reg = <0 0x1001f000 0 0x1000>,
                   <0 0x11c30000 0 0x1000>,
                   <0 0x11c40000 0 0x1000>,
                   <0 0x11e20000 0 0x1000>,
                   <0 0x11e30000 0 0x1000>,
                   <0 0x11f00000 0 0x1000>,
                   <0 0x11f10000 0 0x1000>,
                   <0 0x1000b000 0 0x1000>;
             reg-names = "gpio_base", "iocfg_rt_base", "iocfg_rb_base",
                         "iocfg_lt_base", "iocfg_lb_base", "iocfg_tr_base",
                         "iocfg_tl_base", "eint";
             gpio-controller;
             #gpio-cells = <2>;
             gpio-ranges = <&pio 0 0 100>;
             interrupt-controller;
             interrupts = <GIC_SPI 225 IRQ_TYPE_LEVEL_HIGH>;
             interrupt-parent = <&gic>;
             #interrupt-cells = <2>;
    };
```

```
&pio {
         spi_flash_pins: spi-flash-pins-33-to-38 {
                 mux {
                         function = "flash";
                         groups = "spi0", "spi0_wp_hold";
                 };
                 conf-pu {
                         pins = "SPI2_CS", "SPI2_HOLD", "SPI2_WP";
                         drive-strength = <MTK_DRIVE_8mA>;
                         bias-pull-up = <MTK_PUPD_SET_R1R0_11>;
                 };
                 conf-pd {
                         pins = "SPI2_CLK", "SPI2_MOSI", "SPI2_MISO";
                         drive-strength = <MTK_DRIVE_8mA>;
                         bias-pull-down = <MTK_PUPD_SET_R1R0_11>;
                 };

         };
         ......
```

How to change GPIO driving strength

To modify driving strength, add a “conf-xxxx” in the node.

For example, to modify the driving strength of SPI flash pins, use "spi_flash_pins" to declare the SPI flash node, add "**conf-drive**" in the node, specify the pin name in the "pins" property (See the pinctrl driver pin name assignment), and specify the driving strength in "drive-strength" property. The available driving strengths include "MTK_DRIVE_2mA", "MTK_DRIVE_4mA", ... and "MTK_DRIVE_16mA".

```
&spi0 {
        pinctrl-names = "default";
        pinctrl-0 = <&spi_flash_pins>;
        …
};
```

```
&pio {
        spi_flash_pins: spi-flash-pins-33-to-38 {
                mux {
                        ...
                };
                conf-drive {
                        pins = "SPI2_CS", "SPI2_HOLD", "SPI2_WP";
                        drive-strength = <MTK_DRIVE_8mA>;
                };
        };
};
```

4.3.3.4 Usage and Testing

To configure GPIO, enable OpenWrt Package PACKAGE_gpiod-tools.

```
#make menuconfig
  --> Utilities
      --> gpiod-tools
```

Usage:

```
root@(none):/# gpioset --help
Usage: gpioset [OPTIONS] <chip name/number> <offset1>=<value1> <offset2>=<value2> ...
Set GPIO line values of a GPIO chip and maintain the state until the process exits

root@(none):/# gpioget --help
Usage: gpioget [OPTIONS] <chip name/number> <offset 1> <offset 2> ...
Read line value(s) from a GPIO chip
```

4.3.4 SPI

4.3.4.1 Kernel Configuration

```
CONFIG_SPI=y
CONFIG_SPI_MASTER=y
CONFIG_SPI_MEM=y
CONFIG_SPI_MT65XX=y
```

CONFIG_SPI_MEM is an "SPI memory extension", which is used when SPI-NOR or SPI-NAND is utilized.

4.3.4.2 Source Code

```
<linux>/drivers/spi/spi-mt65xx.c
```

4.3.4.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of SPI node:

```
        spi0: spi@1100a000 {
                compatible = "mediatek,ipm-spi-quad";
                reg = <0 0x1100a000 0 0x100>;
                interrupts = <GIC_SPI 140 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&topckgen CK_TOP_CB_M_D2>,
                         <&topckgen CK_TOP_SPI_SEL>,
                         <&infracfg_ao CK_INFRA_SPI0_CK>,
                         <&infracfg_ao CK_INFRA_SPI0_HCK_CK>;
                clock-names = "parent-clk", "sel-clk", "spi-clk", "spi-hclk";
                status = "disabled";
        };

        spi1: spi@1100b000 {
                compatible = "mediatek,ipm-spi-single";
                reg = <0 0x1100b000 0 0x100>;
                interrupts = <GIC_SPI 141 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&topckgen CK_TOP_CB_M_D2>,
                         <&topckgen CK_TOP_SPIM_MST_SEL>,
                         <&infracfg_ao CK_INFRA_SPI1_CK>,
                         <&infracfg_ao CK_INFRA_SPI1_HCK_CK>;
                clock-names = "parent-clk", "sel-clk", "spi-clk", "spi-hclk";
                status = "disabled";
        };
```

```
&spi0 {
        pinctrl-names = "default";
        pinctrl-0 = <&spi_flash_pins>;
        cs-gpios = <0>, <0>;
        status = "okay";
        spi_nor@0 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-frequency = <52000000>;
                spi-tx-buswidth = <4>;
                spi-rx-buswidth = <4>;
        };
};
```

```
&spi1 {
          pinctrl-names = "default";
          pinctrl-0 = <&spic_pins_g2>;
          status = "okay";
};
```

4.3.5 UART

4.3.5.1 Kernel Configuration

```
CONFIG_SERIAL_8250=y
```

4.3.5.2 Source Code

```
<linux>/drivers/tty/serial/8250/8250_mtk.c
```

4.3.5.3 DTS Section

The device tree source of MT798x is located in,

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

4.3.6 PWM

4.3.6.1 Kernel Configuration

```
CONFIG_PWM=y
CONFIG_PWM_MEDIATEK=y
```

4.3.6.2 Source Code

```
<linux>/drivers/pwm/pwm-mediatek.c
```

4.3.6.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of PWM node:

```
&pwm {
          pinctrl-names = "default";
          pinctrl-0 = <&pwm0_pin &pwm1_pin_g1>;
```

```
        status = "okay";
};
```

4.3.7 USB

The table below shows MT798x USB default modes.

Part	Default mode	Force USB3	Force USB2
MT7986	USB3	-	mediatek,u3p-dis-msk=<0x1>;
MT7981	USB2	mediatek,u3p-dis-msk=<0x0>; or delete this node	-
MT7988	USB3	-	mediatek,u3p-dis-msk=<0x1>;

4.3.7.1 Kernel Configuration

USB3:

```
CONFIG_USB_XHCI_MTK=y
```

USB PHY:

```
CONFIG_PHY_MTK_TPHY=y
```

4.3.7.2 Source Code

```
<linux>/drivers/usb/host/xhci-mtk.c
<linux>/drivers/usb/host/xhci-mtk-xxx.c
```

4.3.7.3 DTS Section

The device tree source of MT798x is located in,

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of USB node:

```
&xhci {
        mediatek,u3p-dis-msk = <0x0>;
        phys = <&u2port0 PHY_TYPE_USB2>,
               <&u3port0 PHY_TYPE_USB3>;
        status = "okay";
};
```

How to force USB3 or USB2

Force USB2, add the following property in the xhci node

```
mediatek,u3p-dis-msk=<0x1>;
```

Force USB3, add the following property in the xhci node

```
mediatek,u3p-dis-msk=<0x0>;
```

4.3.8 Ethernet

4.3.8.1 Kernel Configuration

```
CONFIG_NET_MEDIATEK_SOC=y
```

For **MT7986/MT7981:**

```
CONFIG_MEDIATEK_NETSYS_V2=y
```

For **MT7988:**

```
CONFIG_MEDIATEK_NETSYS_V3=y
```

4.3.8.2 Source Code

```
<linux>/drivers/net/ethernet/mediatek/
```

4.3.8.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of Ethernet node:

```
eth: ethernet@15100000 {
            compatible = "mediatek,mt7986-eth";
            reg = <0 0x15100000 0 0x80000>;
            interrupts = <GIC_SPI 196 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 197 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 198 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 199 IRQ_TYPE_LEVEL_HIGH>;
            clocks = <&ethsys CK_ETH_FE_EN>,
                           <&ethsys CK_ETH_GP2_EN>,
   ......
```

```
&eth {
          status = "okay";

          gmac0: mac@0 {
                  compatible = "mediatek,eth-mac";
                  ......
```

4.3.9 Hardware NAT

4.3.9.1 Kernel Configuration

```
CONFIG_NET_MEDIATEK_HNAT=m
```

4.3.9.2 Source Code

```
<linux>/drivers/net/ethernet/mediatek/mtk_hnat/
```

4.3.9.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of hardware NAT node:

```
hnat: hnat@15000000 {
             compatible = "mediatek,mtk-hnat_v4";
             reg = <0 0x15100000 0 0x80000>;
             resets = <&ethsys 0>;
             reset-names = "mtketh";
             status = "disabled";
};
```

```
&hnat {
        mtketh-wan = "eth1";
        mtketh-lan = "lan";
        mtketh-max-gmac = <2>;
        mtketh-ppe-num = <2>;
        status = "okay";
};
```

4.3.10 Watchdog

4.3.10.1 Kernel Configuration

```
CONFIG_WATCHDOG=y
CONFIG_MEDIATEK_WATCHDOG=y
```

4.3.10.2 Source Code

```
<linux>/drivers/watchdog/mtk_wdt.c
```

4.3.10.3 DTS Section

The device tree source of MT798x is located in,

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of watchdog node:

```
&watchdog {
          status = "okay";
};
```

4.3.11 eMMC

4.3.11.1 Kernel Configuration

```
CONFIG_MMC=y
CONFIG_MMC_MTK=y
```

4.3.11.2 Source Code

```
<linux>/drivers/mmc/host/mtk-sd.c
```

4.3.11.3 DTS Section

The device tree source of MT798x is located in,

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of eMMC node:

```
&mmc0 {
         pinctrl-names = "default", "state_uhs";
         pinctrl-0 = <&mmc0_pins_default>;
         pinctrl-1 = <&mmc0_pins_uhs>;
         bus-width = <8>;
         max-frequency = <200000000>;
         cap-mmc-highspeed;
         mmc-hs200-1_8v;
         mmc-hs400-1_8v;
         hs400-ds-delay = <0x14014>;
         vmmc-supply = <&reg_3p3v>;
         vqmmc-supply = <&reg_1p8v>;
         non-removable;
         no-sd;
         no-sdio;
         status = "okay";
};
```

4.3.12 Crypto Engine

For MT798x crypto engine, refer to the table below.

Part	Crypto engine
MT7981	EIP97
MT7986	EIP97
MT7988	EIP197

4.3.12.1 Kernel Configuration

```
CONFIG_CRYPTO_HW=y
```

4.3.12.2 Source Code

```
<linux>/drivers/crypto/inside-secure/
```

4.3.12.3 DTS Section

The device tree source of MT798x is located in

```
target/linux/mediatek/files-5.4/arch/arm64/boot/dts/mediatek/
```

Example of crypto node:

```
crypto: crypto@15600000 {
                  compatible = "inside-secure,safexcel-eip197b";
                  reg = <0 0x15600000 0 0x180000>;
                  interrupts = <GIC_SPI 214 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 215 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 216 IRQ_TYPE_LEVEL_HIGH>,
                               <GIC_SPI 217 IRQ_TYPE_LEVEL_HIGH>;
                  interrupt-names = "ring0", "ring1", "ring2", "ring3";
                  status = "okay";
};
```

4.4 Feeds

In OpenWrt, a "feed" is a collection of packages that share a common location. Feeds may reside on a remote server, in a version control system, on the local filesystem, or in any other location addressable by a single name (path/URL) over a protocol with a supported feed method.

4.4.1 Search and Install Package from Feeds

To search and install packages from feeds:

```
scripts/feeds update                                                  # update the index and
save it into <openwrt>/feeds
scripts/feeds search <keywords>
scripts/feeds install <package-name>
```

After “scripts/feeds install”, you will be able to find the package entry in “**make menuconfig**”.

For advanced usage, please read its help message and refer to: https://wiki.openwrt.org/doc/devel/feeds

4.4.2 Add New Feeds

Feeds URLs are recorded in a text file, "<openwrt>/feeds.conf.default".

```
src-git mtk_openwrt_feed https://gerrit.mediatek.inc/openwrt/feeds/mtk_openwrt_feeds
src-git packages https://gerrit.mediatek.inc/openwrt/feeds/packages;openwrt-21.02
src-git luci https://gerrit.mediatek.inc/openwrt/feeds/luci;openwrt-21.02
src-git routing https://gerrit.mediatek.inc/openwrt/feeds/routing;openwrt-21.02
```

You can add your own feeds by adding new lines into it.

```
<src-method> <feeds-name> <feeds-url>[;<branch-name>]
```

"scripts/feeds" command supports a lot of methods to get feeds:

- src-svn: subversion vcs
- src-cpy: file copy
- src-link: symbol link
- src-git: git vcs (shallow clone)
- src-git-full: git vcs (full clone)
- src-gitsvn: gitsvn fusion
- src-bzr: GNU bazaar vcs
- src-hg: mecury vcs
- src-darcs: darcs vcs

4.4.3 Make Your Own Feeds

The easiest way of building your feeds is to fork from existing feeds. https://github.com/openwrt/packages is a good example for your reference. Download this repository, take a look inside, and put your package into it.

4.5 MTK Proprietary Packages

4.5.1 mii_mgr/mii_mgr_cl45

Description: "mii_mgr" is an application to read and write MII registers. It uses ioctl to communicate with kernel's Ethernet drivers.

Usage:

Get:

```
mii_mgr -g -p [phy number] -r [register number]
```

Set:

```
mii_mgr -s -p [phy number] -r [register number] -v [0xvalue]
```

Examples 1:

```
mii_mgr -g -p 3 -r 4
```

Example 2:

```
mii_mgr -s -p 4 -r 1 -v 0xff11
```

mii_mgr_cl45

Usage:

Get:

```
mii_mgr_cl45 -g -p [port number] -d [dev number] -r [register number]
```

Set:

```
mii_mgr_cl45 -s -p [port number] -d [dev number] -r [register number] -v [value]
```

Examples 1:

```
mii_mgr_cl45 -g -p 3 -d 0x5 -r 0x4
```

Example 2:

```
mii_mgr_cl45 -s -p 4 -d 0x6 -r 0x1 -v 0xff11
```

4.5.2 regs

Description: “regs” is an application to read and write registers.

Usage:

```
regs [Type] [ Offset:Hex ] [ Data:Hex ] [StartBit:Dec] [DataLen:Dec]
      Type     : access operation type : [m]odify, [w]wite, [d]ump
      Offset   : offset into memory region to act upon
      Data     : data to be written
      Startbit: Startbit of Addr that want to be modified
      DataLen : Data length of Data
```

Example 1, dump 0x1b100000~0x1b1000f0

```
regs d 0x1b100000
```

Example 2, write 0x1b100000=0x1234

```
regs w 0x1b100000 0x1234
```

Example 3, modify 0x1b100000[29:31]=0

```
regs m 0x1b100000 0x0 29 3
```

4.5.3 Switch

Description: “switch” is an MT753x switch tool.

Usage: Enter the “switch” command to see the usage description.

5 Develop with MTK OpenWrt SDK

5.1 Create a New Package

Take the following “helloworld” application as an example.

```
helloworld
  ├── Makefile # openwrt’s “package makefile”
  └── src
        ├── Makefile # helloworld’s makefile
        └── helloworld.c # helloworld source code
```

- src/helloworld.c

```
#include<stdio.h>
int main(void)
{
    printf("HelloWorld!\n");
    printf("This is my first package program\n");
    return 0;
}
```

- src/Makefile

```
OBJ = helloWorld
OBJS = helloWorld.o
CC = gcc
CFLAGS = -c -Wall -O
RM = rm -rf

$(OBJ):$(OBJS)
       $(CC) -o $(OBJ) $(OBJS)
$(OBJS): helloWorld.c
       $(CC) $(CFLAGS) helloWorld.c
clean:
       $(RM) *.o $(OBJ)
```

- Package Makefile

```
# Import common build rules
include $(TOPDIR)/rules.mk

PKG_NAME:=helloWorld
PKG_RELEASE:=1
PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)

# Import package definitions
include $(INCLUDE_DIR)/package.mk

# Define a new package
define Package/helloWorld
	SECTION:=MTK Properties
	CATEGORY:=MTK Properties
	SUBMENU:=Applications
	TITLE:=helloWorld - learn from example.
endef

define Package/helloWorld/description
	It's my first package demo
endef

# Prepare source code. use tabs, not spaces.
define Build/Prepare
	echo "Here is Build/Prepare"
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

# Install scripts. use tabs, not spaces.
define Package/helloWorld/install
	echo "Here is Package/install"
	$(INSTALL_DIR) $(1)/bin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloWorld $(1)/bin/
endef

# In the end, call build command.
$(eval $(call BuildPackage, helloWorld))
```

Put this “helloworld” folder under “**<openwrt>/package/**” or its subfolder. Execute “make menuconfig”, and you will find a new configurable option under “MTK properties ---> Applications”. Select it as either “*” or “m”. Then, you can build “helloworld” with:

```
make package/helloworld/{clean,prepare,compile} V=s
```

5.2 Work with Patches

5.2.1 The “quilt” Utility

Quilt is a powerful tool that helps you to create, apply and modify many patches. It is based on the “diff & patch” utility and provides stack-style management for all your patches.
Refer to the following for its usage:

- https://wiki.debian.org/UsingQuilt
- https://linux.die.net/man/1/quilt

5.2.2 Make Patch for the Kernel

For example, to modify or add the file “drivers/mtd/xxx.c” in the kernel, follow the steps below.

```
1. make target/linux/{clean,prepare} QUILT=1
2. cd <LINUX_DIR>
3. quilt new platform/001-test.patch
4. quilt add drivers/mtd/xxx.c
5. vim drivers/mtd/xxx.c
6. quilt diff
7. quilt refresh
8. cd <OPENWRT_ROOTDIR>
9. make target/linux/update
```

1. [Optional] Reinstall the kernel to get a clean kernel source tree. “**QUILT=1**” ensures you can use the quilt utility with the source code.
2. Enter the kernel source folder. It is usually located at: “build_dir/target-<toolchain info>/linux-<platform>-<model>/linux-<version>”.
3. Use the “**quilt new**” command to create a new patch. After this command, a new empty patch file will be generated under the “.patches” folder.
   a. Usually, you should choose a number prefix greater than any other patches for the component, because the build script will apply patches one after another according to the numbers.
   b. The prefix “platform” means the patch is platform specific, then the patch will be saved into “target/linux/<platform>/patches” later, for generic patches, you can use the prefix “generic”, then the patch will go to “target/linux/generic/patches”.
4. Use “**quilt add**” command to add the file you want to change or create. This tells the quilt utility to keep track of every change you will make on this file.
5. Edit the file as you wish. You can use every editor you like. You can also remove the file.
6. Use “**quilt diff**” command to check the modification you just made. If you are not happy with the current code, just go back to step 5.
7. Use “**quilt refresh**” to flush your changes into a patch file.
8. Go back to OpenWrt’s root dir.
9. “**make target/linux/update**” will copy all the patches into corresponding folders. After updating, you will find a new patch occurs in “target/linux/<platform>/patches”.

5.2.3 Make Patch for a Package

It’s pretty much the same as to make a patch for the kernel.

```
make package/<package name>/{clean,prepare} QUILT=1
cd <PACKAGE_BUILD_DIR>
quilt new 001-test.patch
quilt add xxx.c
vim xxx.c
quilt diff
quilt refresh
cd <OPENWRT_ROOTDIR>
make package/<package name>/update
```

5.3 Customize Booting Sequence

5.3.1 “/etc/inittab”

If you want your job to be invoked by **init process** directly, you should turn to “**/etc/inittab**”.
The inittab file describes which processes are started at bootup and during normal operation (e.g. /etc/init.d/boot, /etc/init.d/rc, gettys...). It is read by the “init process”. Init process distinguishes multiple runlevels, each of which can have its own set of processes that are started.
By default, you will find the following lines in “/etc/inittab”.

```
::sysinit:/etc/init.d/rcS S boot # invoke rcS boot scripts
::shutdown:/etc/init.d/rcS K shutdown # invoke rcS shutdown scripts
ttyS0::respawnlate:/usr/libexec/login.sh # start login via console
```

During system booting, Init process will read “/etc/inittab” for the "sysinit" entries (default is "::sysinit:/etc/init.d/rcS S boot"). So, in the example above, init process will call “/etc/init.d/rcS **S boot**”. There are more methods supported by OpenWrt’s “init process”.

1. **respawn** - this works just like you expect it. It starts a process and will respawn it once it has been completed.
2. **respawnlate** - this works like the respawn but will start the process only when the procd init is completed.
3. **askfirst** - this works just like respawn but will print the line “Please press Enter to activate this console.” before starting the process.
4. **askconsole** - this works like askfirst but, instead of running on the tty passed as a parameter, it will look for the tty defined in the kernel command line using “console=”.
5. **askconsolelate** - this works like the askconsole but will start the process only when the procd init is completed.
6. **sysinit** - this will trigger procd to run the command, given as a parameter, only once. This is usually used to trigger the execution of /etc/rc.d/.

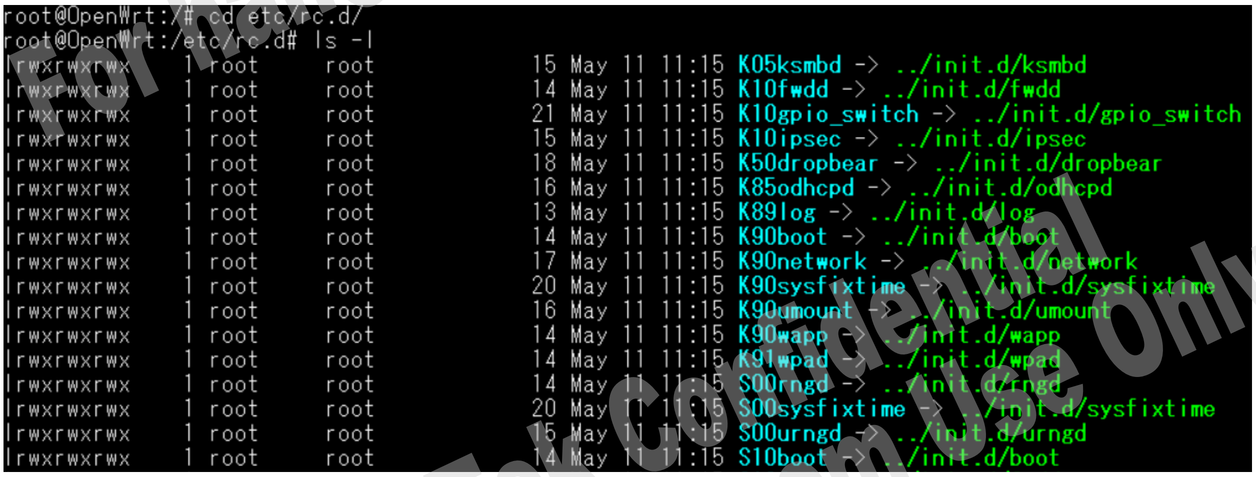
5.3.2 /etc/init.d and /etc/rc.d

“/etc/inittab” will invoke the rcS scripts during system booting. rcS executes the symlinks to the actual startup scripts located in /etc/init.d/S##xxxxxx with the option "start".

After rcS finishes, the system should be up and running.

The following example from “**<openwrt>/package/mtk/applications/hwnat/Makefile**” shows how to install your jobs into rcS.

```
define Package/hwnat/install
    $(INSTALL_DIR) $(1)/bin
    $(INSTALL_DIR) $(1)/etc/init.d/
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/hwnat $(1)/bin
    $(INSTALL_BIN) ./files/* $(1)/etc/init.d
endef
```



```
root@OpenWrt:/# cd etc/rc.d/
root@OpenWrt:/etc/rc.d# ls -l
lrwxrwxrwx    1 root     root            15 May 11 11:15 K05ksmbd -> ../init.d/ksmbd
lrwxrwxrwx    1 root     root            14 May 11 11:15 K10fwdd -> ../init.d/fwdd
lrwxrwxrwx    1 root     root            21 May 11 11:15 K10gpio_switch -> ../init.d/gpio_switch
lrwxrwxrwx    1 root     root            15 May 11 11:15 K10ipsec -> ../init.d/ipsec
lrwxrwxrwx    1 root     root            18 May 11 11:15 K50dropbear -> ../init.d/dropbear
lrwxrwxrwx    1 root     root            16 May 11 11:15 K85odhcpd -> ../init.d/odhcpd
lrwxrwxrwx    1 root     root            13 May 11 11:15 K89log -> ../init.d/log
lrwxrwxrwx    1 root     root            14 May 11 11:15 K90boot -> ../init.d/boot
lrwxrwxrwx    1 root     root            17 May 11 11:15 K90network -> ../init.d/network
lrwxrwxrwx    1 root     root            20 May 11 11:15 K90sysfixtime -> ../init.d/sysfixtime
lrwxrwxrwx    1 root     root            16 May 11 11:15 K90umount -> ../init.d/umount
lrwxrwxrwx    1 root     root            14 May 11 11:15 K90wapp -> ../init.d/wapp
lrwxrwxrwx    1 root     root            14 May 11 11:15 K91wpad -> ../init.d/wpad
lrwxrwxrwx    1 root     root            14 May 11 11:15 S00rngd -> ../init.d/rngd
lrwxrwxrwx    1 root     root            20 May 11 11:15 S00sysfixtime -> ../init.d/sysfixtime
lrwxrwxrwx    1 root     root            15 May 11 11:15 S00urngd -> ../init.d/urngd
lrwxrwxrwx    1 root     root            14 May 11 11:15 S10boot -> ../init.d/boot
```

5.3.3 /etc/rc.local

This file is a shell script that will be invoked at the end of **rcS**. If you want a job done after the system’s fully started, you can put your job script here.

For example, the following “rc.local” script prints how many times the set has been rebooted on the console.

```
# Put your custom commands here that should be executed once
# the system init finished. By default this file does nothing.

echo 1 >> /etc/reboot-times
cat /etc/reboot-times | wc -l > /dev/console

exit 0
```

6 Frequently Asked Questions

6.1 What Is the Difference between squashfs-factory.bin, squashfs-sysupgrade.bin and initramfs-kernel.bin?

When building OpenWrt kernel images, three types of images are generated.

openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-**initramfs-kernel**.bin
openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-**squashfs-factory**.bin
openwrt-mediatek-mt7986-mt7986a-ax6000-2500wan-spim-nand-rfb-**squashfs-sysupgrade**.bin

- The image file with the suffix "***squashfs-sysupgrade***" is used to upgrade the kernel image to flash, either by Web UI "upgrade firmware" page or U-Boot menu option 2 (upgrade firmware).
- The image file with the suffix "***squashfs-factory***" is used when generating a single image. A single image must be written to flash by a flash programmer or U-Boot menu option 5 (upgrade single image).
- The image file with the suffix "***initramfs-kernel***" is used to run on DRAM. You can only load the initramfs-kernel.bin image to DRAM by U-Boot menu 6 (Load image). Note that the image will be lost after a power-reset or power-off.

6.2 What Is the Difference between initramfs.bin and squashfs.bin? Which One Should I Use?

The main differences between the squashfs firmware and initramfs firmware are:

- Squashfs firmware puts the rootfs on flash, and Linux kernel will mount it during booting, while initramfs puts the Linux rootfs as initramfs, which is linked with the kernel.
- Squashfs firmware takes flash partition "rootfs_data" as its backend storage, while initramfs firmware uses a tmpfs (which is part of your ram) as the backend storage. So, **initramfs firmware does not preserve your data, you will lose the changes you've made after reboot.**
- Squashfs firmware is supposed to be written into the flash storage, while initramfs firmware is supposed to stay in RAM. You should choose the right option in the U-Boot menu.

In general cases, you should use squashfs, but in some cases, you may need to use initramfs. See the following for the special cases.

- Your firmware does not work properly because of flash-related errors, such as the kernel failing to find rootfs.
- You want to analyze the flash data on the device.
- You want to try new firmware, but don't want to erase the current firmware.

6.3 How to Place Files into rootfs?

6.3.1 Option 1: Place Your Files under “base-files“

Files under “base-files” folder are copied into rootfs during the build. There are several folders named “base-files”, and you can put your files into any of them.

- **package/base-files**, generic files, can be used for all platforms.
- **target/linux/mediatek/base-files**, MediaTek specific files.

6.3.2 Option 2: “Package/<package name>/install” Section in a Package Makefile

```
define Package/<package name>/install
		$(INSTALL_DIR) $(1)/etc/init.d/
		$(INSTALL_BIN) ./myfile $(1)/etc/init.d
endef
```

- This section is executed as “make install” after the package is successfully built.
- **$(INSTALL_DIR)** creates the folders, and **$(INSTALL_BIN)** copies your files into rootfs. $(1) is the target rootfs path during the build.
- This script uses the package’s path as its working directory, so you can use the relative path to access the package’s files.

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.