MTK OpenWrt SDK User Manual

Version: 4.0

Release date: 2021-01-05

Document Revision History

Revision	Date	Author	Description
V4.0	2017-12-31	Hua Shao	Initial Draft.

Table of Contents

Lists of Tables and Figures

1 About MTK OpenWrt SDK

1.1 What is OpenWrt

Figure 1-1. OpenWrt Logo

- From https://openwrt.org/ :

  OpenWrt is described as a Linux distribution for embedded devices.

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

- From https://en.wikipedia.org/wiki/OpenWrt :

  OpenWrt is a linux distribution primarily used on embedded devices to route network traffic. The main components are the Linux kernel, uClibc, busybox, and OpenWrt framework utilities. All components have been optimized for size, to be small enough for fitting into the limited storage and memory available in the routers.

1.1.1 What is lede?

On 4 May, 2016, some of the OpenWrt developers announced that they are going to start a new project named “lede”, which stands for “*Linux Embedded Development Environment”*. Actually lede is a fork of OpenWrt, that’s why they call that announcement as “reborn of OpenWrt”.

- Description from https://lede-project.org/

  The LEDE Project (“Linux Embedded Development Environment”) is a Linux operating system based on OpenWrt. It is a complete replacement for the vendor-supplied firmware of a wide range of wireless routers and non-network devices.

About 1 year later, OpenWrt and Lede finally merged together. They both refer to the same thing now.

1.2 MediaTeK's OpenWrt SDK

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 better stability, some of OpenWrt's components are replaced by MTK 's proprietary implementations, including some applications, kernel and drivers (ethernet, USB, WiFi, SD Card, etc.).

1.3 SDK Revision History

1.3.1 V4.x

SDK V4.x series are independently developed from previous SDK.

SDK V4.0 is the latest MTK OpenWrt SDK. It is developed based on **LEDE 17.01**. It uses customized kernel linux-4.9.92.

Supported SoC Chips: MT7622, MT7623

Supported WiFi Chips: MT7622, MT7615e

Older chips such as MT7620/MT7628/MT7621, you should turn to v3.x series, which is still under maintaining.

2 Using MTK OpenWrt SDK

2.1 Unzip the code

Unzip the SDK, you will get the following folders and files:

```
drwxr-xr-x.  7 user user  4096 Dec 21 14:26 autobuild
-rw-r--r--.  1 user user   179 Dec 21 14:22 BSDmakefile
drwxr-xr-x.  2 user user    96 Dec 21 14:22 config
-rw-r--r--.  1 user user   573 Dec 21 14:22 Config.in
drwxr-xr-x.  2 user user  4096 Dec 21 14:26 dl
-rw-r--r--.  1 user user   244 Dec 21 14:22 do.not.release
-rw-r--r--.  1 user user   295 Dec 21 14:22 feeds.conf.default
drwxr-xr-x.  3 user user  4096 Dec 21 14:22 include
-rw-r--r--.  1 user user 17992 Dec 21 14:22 LICENSE
-rw-r--r--.  1 user user  3026 Dec 21 14:22 Makefile
drwxr-xr-x. 14 user user  4096 Dec 21 14:22 package
-rw-r--r--.  1 user user  1014 Dec 21 14:22 README
-rwxr-xr-x.  1 user user  4381 Dec 21 14:22 release_sdk.sh
-rw-r--r--.  1 user user 12767 Dec 21 14:22 rules.mk
drwxr-xr-x.  4 user user  4096 Dec 21 14:22 scripts
drwxr-xr-x.  6 user user    94 Dec 21 14:22 target
drwxr-xr-x. 13 user user  4096 Dec 21 14:22 toolchain
drwxr-xr-x. 58 user user  4096 Dec 21 14:22 tools
```

- **autbuild**: some configuration files which may help you to build a default firmware.
- **BSDmakefile**: makefile for BSD system.
- **config**: a collection of global configrations for menuconfig.
- **Config.in:** main entrance of all configuration menu. It refers to a lot of other “config.in” from under **config** folder.
- **dl:** This folder is where openwrt puts its downloaded components. We aready put some components here, including customized kernel, some MediaTek’s proprietary applications.
- **docs**: some help manual in TeX/LaTeX format.
- **feeds.conf.default**: OpenWrt have thousands of 3rd party components (plugins) you can install with “scripts/feeds” command, they are hosted in serval repositories, and this file records url of those repositories.

- **include**: a bunch of "*.mk" files. It consists OpenWrt's powful (also complicated) build system. They will be included into almost every Makefile in OpenWrt.
- **LICENCE**: GPLv2 licence.
- **Makefile**: Core makefile, it defines entrance to the giant make system.
- **package**: A package is a configurable comipling unit of OpenWrt. OpenWrt's consists of a lot of packages, such as busybox, dnsmasq, procd, ubus, uci…They are all defined under this folder.
- **README**: Read it.
- **rules.mk**: a general makefile that defines some global variables and routines, it is included in every Makefile, directly or indirectly.
- **scripts**: many helper scripts that is going to be used during configuration and compiling.
- **target**: It includes platform specific stuff. Every platform that is supported by OpenWrt has a folder which includes its source code, patches, scripts, configurations, dts, etc.
- **toolchain**: Defines cross compiler's configuration, compiling and installation.
- **tools**: source code of some tools, which will be used during compiling.

2.2 Setup precondition

OpenWrt requires some tools from your host, if one of them is missing, you will get errors at the beginning of configuring or compiling.

Normally you can skip this step and jump to configuration or compilation directly. Once you get errors, you can go back here to install tools that was missing.

Note that these commands requires root previlidge.

- **If you are working on a RedHat or CentOS server:**

```
yum install binutils
yum install bzip2
yum install gawk
yum install gcc
yum install gcc-c++
yum install gettext
yum install make
yum install ncurses-devel
yum install patch
yum install unzip
yum install wget
yum install zlib-devel
yum install subversion
```

- **If you are working on a Ubuntu server:**

```
apt-get install g++
apt-get install libncurses5-dev
apt-get install zlib1g-dev
apt-get install bison
apt-get install flex
apt-get install unzip
apt-get install autoconf
apt-get install gawk
apt-get install make
apt-get install gettext
apt-get install gcc
apt-get install binutils
apt-get install patch
apt-get install bzip2
apt-get install libz-dev
apt-get install asciidoc
apt-get install subversion
```

2.3 Configure the SDK

OpenWrt is highly configurable.

OpenWrt supports a lot of target boards, a lot of applications, and every component of OpenWrt may have a lot of configurable features.

Before you start to build it, you must configure it.

2.3.1 Configure OpenWrt

Under SDK root folder, execute:

```
make menuconfig
```

You will get a menu that lists out all configurable features of the SDK.

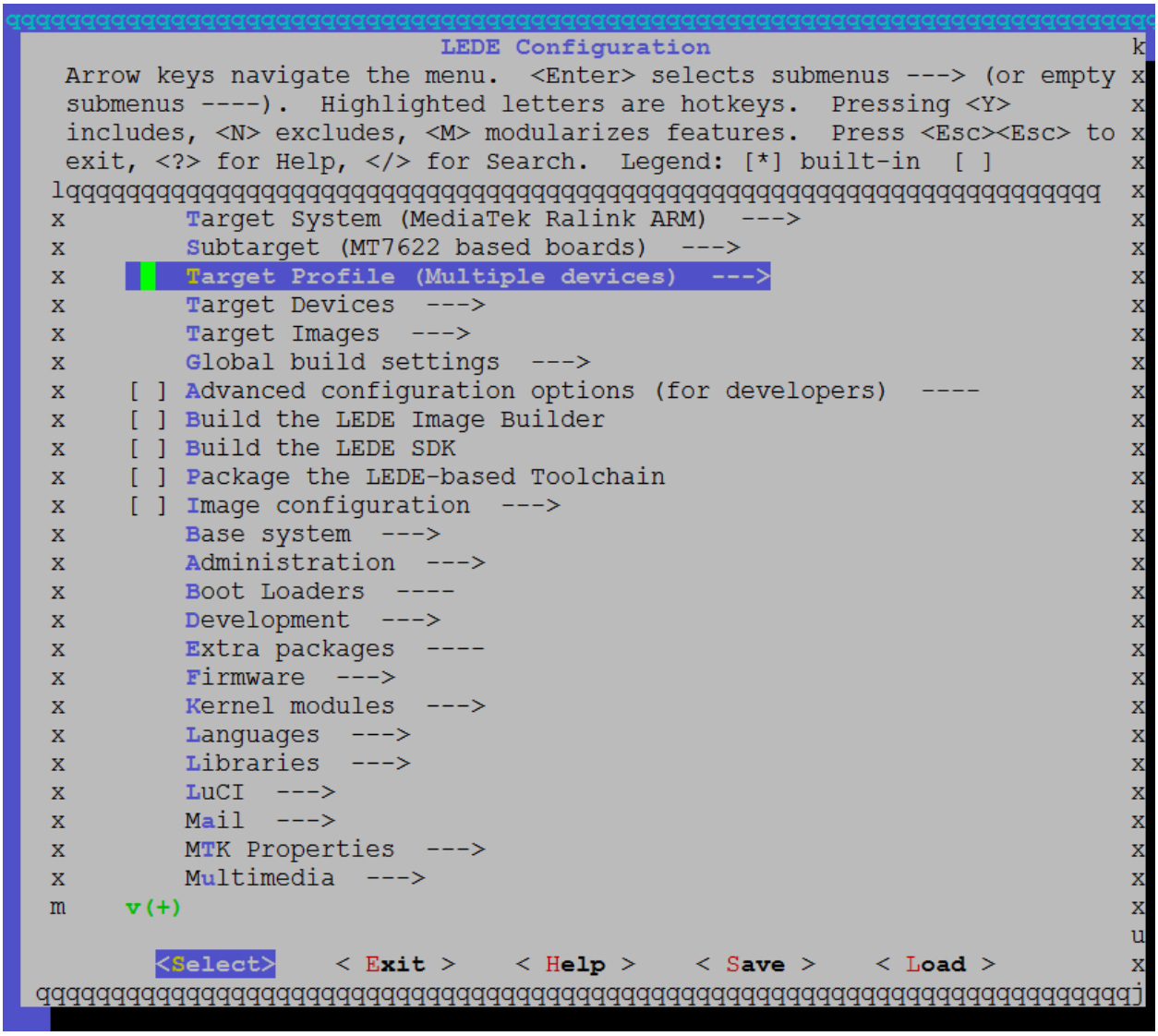

Figure 2-1 SDK menuconfig

For the simplest start, you need to configure at least 3 items:

```
Target System (MediaTek Ralink ARM)  ---> # platform
Subtarget (MT7622 based boards)  ---> # chipset
Target Profile (MTK7622 ac2600 rfb1 AP )  ---> # product model
```

If you select “Multiple devices” as target profile, a new “Target Devices” will turn up. You should select the devices your need.

```
Target System (MediaTek Ralink ARM)  ---> # platform
Subtarget (MT7622 based boards)  ---> # chipset
Target Profile (Multiple devices)  ---> # product model
Target Devices  ---> # target devices
```

After menuconfig done, you configuration will be saved into "**.config**".

Note that OpenWrt uses a different configuration syntax from kernel for ***tristage features***.

- (*) means this feature will be built and installed into the firmware directly.
- (M) means this feature will be built as a stand alone package (or plugin), that can be installed into the target system at runtime.

2.3.2 Configure Linux Kernel

OpenWrt provides default kernel configuration, you can find it at target/linux/<target>/<subtarget>/config-<kernel-version>. If the default configuration does not meet your needs, you can configure the kernel by yourself.

Under SDK root folder, execute:

```
make kernel_menuconfig
```

Then you will see the classic kernel configuration menu. It uses the same syntax as linux kernel configuration.

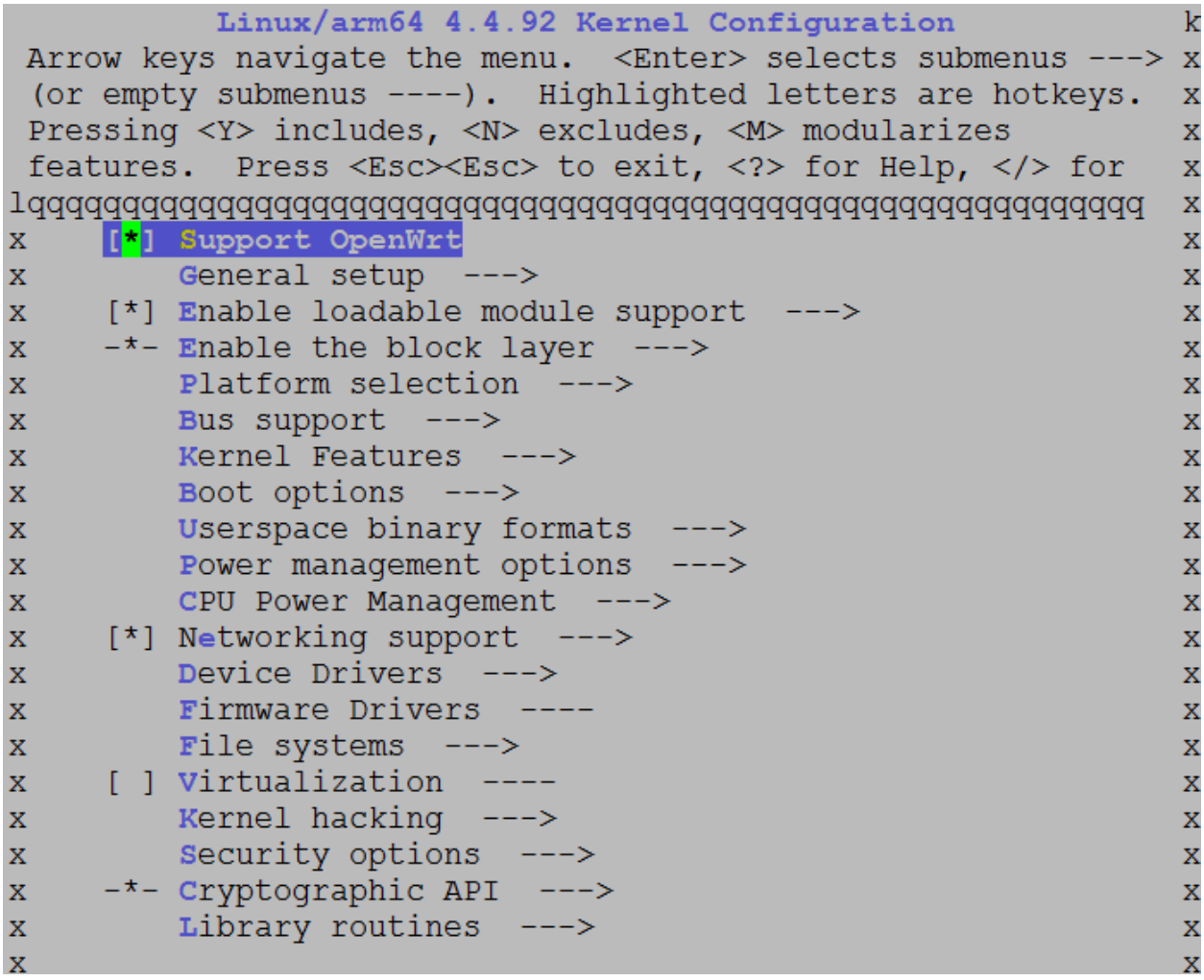

Figure 2-2 OpenWrt Kernel_menuconfig

After configuration done, you configuration will be saved into target/linux/<target>/<subtarget>/config-<kernel-version>.

2.3.3 Install and configure wireless drivers

Since SDK V4.x, MediaTek's Wireless drivers are not released along with OpenWrt SDK. They are taken as a normal package of OpenWrt, and are built outside the kernel source tree.

Check the "WiFi Drivers" section for more information.

2.3.4 Using pre-defined configurations

Under “autobuild” folder, there are some pre-defined configurations, which can be a good start for you.

You can copy one of them to the root folder and rename it to “.config”.

```
cp autobuild/xxxx.config .config
make defconfig
```

2.4 Build

Under SDK root folder, call:

```
make
```

or

```
make V=s  # this will produce verbose log
```

or

```
make V=s -j4  # this makes 4 parallel build threads, can be faster in multi-core environment.
```

OpenWrt’s Compilation happens in “build_dir”, normally it includes 4 stages:

1. Prepare
   a) locate the source code. Typically it resides under “dl” folder as a tarball.
   b) Install the source code under “build_dir”
   c) Apply patches to the source code if necessary.
2. Compile
3. Install
   a) Install the component into rootfs.
   b) Install the IPK under “bin” folder.
   c) Install headers, libraries into “staging_dir” if necessary.
4. Clean
   a) Clean up the code under “build_dir”

Some other info you may want to know:

- If you have selected a feature that is not in the default configurations, the SDK may needs to download corresponding source code from Internet. In that case, you need to make sure your build host can access the open Internet.
- The first compilation will take a long time because it needs to build the toolchain and host tools first. After the first build, normally your build will be ready in minutes.
- If anything goes wrong during building, use “make V=s” (strip the “-j” parameter) to see what happened.

If everything is OK, you will get a “bin” folder, with both firmwares and packages(*.ipk) inside.

2.4.1 Partial build

Other than the full firmware, OpenWrt supports to build a component individually. This is useful when you want to develop or debug a component.

To build the kernel:

```
make target/linux/compile
```

To build a package:

```
make package/<package path>/compile
```

Some times you need to clean the component first, you can combine several stages together:

```
make target/linux/{clean,prepare,compile}

make package/<package path>/{clean,prepare,compile}
```

Note that a partial build will not update the final firmware, it just updates the component.

2.4.2 Firmwares and packages

After a successful build, you will get a “bin” folder with the following contents:

- **Firmware**: includes firmwares you can flash into your boards.

```
bin/targets/
└──── mediatek/
    └──── mt7622-glibc/
        ├──── config.seed
        ├──── lede-mediatek-mt7622-device-mtk-ac2600-rfb1.manifest
        ├──── lede-mediatek-mt7622-MTK-AC2600-RFB1-squashfs-
sysupgrade.bin
        ├──── mt7622-ac2600rfb1.dtb
        ├──── packages/
        └──── sha256sums
```

- **config.seed**: SDK configuration.
- **lede-xxxx.manifest**: a text file that keep record of revision of current firmware's components.
- **lede-xxxx-sysupgrade.bin**: the firmware you can flash into your board.
- **xxxx.dtb**: binary format of dts file of corresponding firmware.
- **packages**/ : platform specific ipks.
- **sha256sum**: a text file that keep record of sha256sum values of some important files. You can use "sha256sum <filename>" to check if the file was unexpectly changed.

- **Packages**: IPKs that you can install into a running system via "opkg". They are usually platform independent.

2.5 Install

There are several ways of flashing new firmware into your device.

2.5.1 Upgrade via uboot

MediaTek's uboot provides various ways of flashing new firmware.

Normally option 2 is your best choice. You need to setup a TFTP server on your PC, and connect your PC's net cable with the board's LAN port.

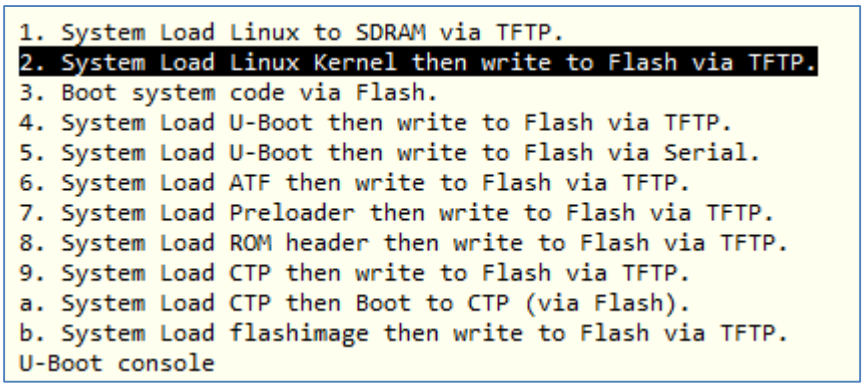

Figure 2-3 Upgrade via u-boot

2.5.2 Upgrade via "sysupgrade"

OpenWrt has a built-in command "sysupgrade" which can upgrade the firmware in one line.

```
/sbin/sysupgrade [<upgrade-option>...] <image file or URL>
```

Check the help message of "sysupgrade" for advanced usage.

2.5.3 Upgrade via WEB UI

Precondition: You have LuCI installed properly.

Step 1: Visit http://192.168.1.1 , "System" -> "Backup/Flash Firmware"

Step 2: Find the "Flash new firmware image" section, choose your firmware, then click "Flash image" button.

See the images below.

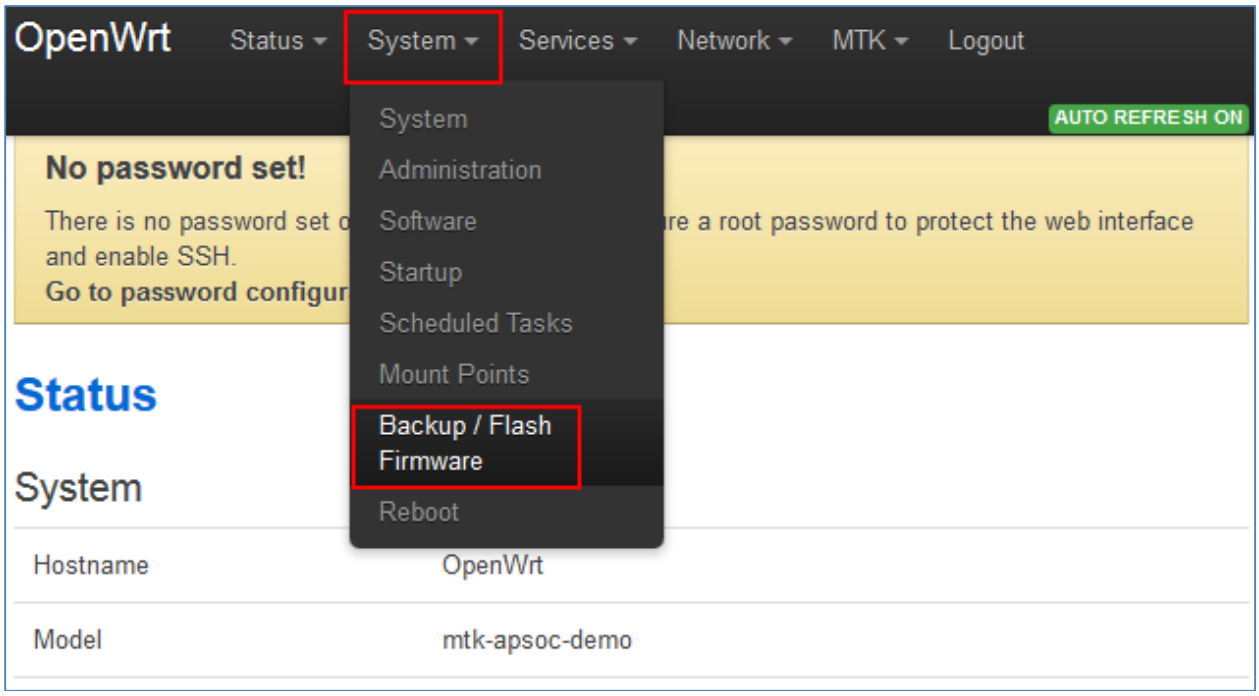

Figure 2-4 Upgrade via LuCI Step 1

Figure 2-5 Upgrade via LuCI Step 2

2.5.4 Install packages (ipk)

An ipk file can be installed into a running system via “opkg” command.

```
opkg install <ipk path or url>
```

If you are developing an application, combining “partial build” and “opkg” together are quite handy.

3 SDK components introduction

3.1 Components Overview

3.2 Toolchain

Usually you don't need to configure OpenWrt's toolchain.

Once you choose a platform, corresponding toolchain configuration will be generated for you. Everything you need to cross-compile a firmware for you platform has been integrated into OpenWrt's built framework.

Toolchains will be built the first time you build the project, then be saved under "staging_dir" for further use.

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

3.2.1 C/C++ Libraries

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

- **uClibc** : uClibc (https://www.uclibc.org/) is a small C standard library intended for Linux kernel-based operating systems for embedded systems and mobile devices. *uClibc* was created to support µClinux, a version of Linux not requiring a memory management unit and thus suited for microcontrollers.
- **Musl** : musl (https://www.musl-libc.org/) is a C standard library intended for operating systems based on the Linux kernel, released under the MIT License. It was developed by Rich Felker with the goal to write 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
--> Advanced configuration options (for developers)
    --> Toolchain options
        --> C library implementations
            --> (x) Use glibc
            --> ( ) Use musl
```

By default, MT7622 uses "glibc", while MT7623 uses "musl".

3.3 Peripherals

3.3.1 PCI

3.3.1.1 Kernel configuration:

```
CONFIG_PCIE_MEDIATEK=y
```

3.3.1.2 Source code:

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

3.3.1.3 DTS section:

```
pcie: pcie@1a140000 {
    compatible = "mediatek,mt7622-pcie";
    device_type = "pci";
    reg = <0 0x1a140000 0 0x1000>,
          <0 0x1a143000 0 0x1000>,
          <0 0x1a145000 0 0x1000>;
    reg-names = "subsys", "port0", "port1";
    #address-cells = <3>;
    #size-cells = <2>;
    interrupts = <GIC_SPI 228 IRQ_TYPE_LEVEL_LOW>,
                 <GIC_SPI 229 IRQ_TYPE_LEVEL_LOW>;
    clocks = <&pciesys CLK_PCIE_P0_MAC_EN>,
             <&pciesys CLK_PCIE_P1_MAC_EN>,
             <&pciesys CLK_PCIE_P0_AHB_EN>, /* designer has connect rc1 with p0_ahb
clock */
             <&pciesys CLK_PCIE_P0_AHB_EN>,
             <&pciesys CLK_PCIE_P0_AUX_EN>,
             <&pciesys CLK_PCIE_P1_AUX_EN>,
             <&pciesys CLK_PCIE_P0_AXI_EN>,
             <&pciesys CLK_PCIE_P1_AXI_EN>,
             <&pciesys CLK_PCIE_P0_OBFF_EN>,
             <&pciesys CLK_PCIE_P1_OBFF_EN>,
             <&pciesys CLK_PCIE_P0_PIPE_EN>,
             <&pciesys CLK_PCIE_P1_PIPE_EN>;
    clock-names = "sys_ck0", "sys_ck1", "ahb_ck0", "ahb_ck1",
                  "aux_ck0", "aux_ck1", "axi_ck0", "axi_ck1",
                  "obff_ck0", "obff_ck1", "pipe_ck0", "pipe_ck1";

    power-domains = <&scpsys MT7622_POWER_DOMAIN_HIF0>;
    bus-range = <0x00 0xff>;
    ranges = <0x82000000 0 0x20000000  0x0 0x20000000  0 0x10000000>;

    pcie0: pcie@0,0 {
```

```
        device_type = "pci";
        reg = <0x0000 0 0 0 0>;
        #address-cells = <3>;
        #size-cells = <2>;
        #interrupt-cells = <1>;
        ranges;
        num-lanes = <1>;
        interrupt-map-mask = <0 0 0 7>;
        interrupt-map = <0 0 0 1 &pcie_intc0 0>,
                        <0 0 0 2 &pcie_intc0 1>,
                        <0 0 0 3 &pcie_intc0 2>,
                        <0 0 0 4 &pcie_intc0 3>;
        pcie_intc0: interrupt-controller {
            interrupt-controller;
            #address-cells = <0>;
            #interrupt-cells = <1>;
        };
    };

    pcie1: pcie@1,0 {    device_type = "pci";
        reg = <0x0800 0 0 0 0>;
        #address-cells = <3>;
        #size-cells = <2>;
        #interrupt-cells = <1>;
        ranges;
        num-lanes = <1>;
        interrupt-map-mask = <0 0 0 7>;
        interrupt-map = <0 0 0 1 &pcie_intc1 0>,
                        <0 0 0 2 &pcie_intc1 1>,
                        <0 0 0 3 &pcie_intc1 2>,
                        <0 0 0 4 &pcie_intc1 3>;
        pcie_intc1: interrupt-controller {
            interrupt-controller;
            #address-cells = <0>;
            #interrupt-cells = <1>;
        };
    };
};
```

3.3.1.4 Usage and testing:

```
lspci -v
```

3.3.2 I2C

3.3.2.1 Kernel configuration:

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

3.3.2.2 Source code:

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

3.3.2.3 DTS section:

In dtsi:

```
i2c0: i2c@11007000 {
    compatible = "mediatek,mt7622-i2c";
    reg = <0 0x11007000 0 0x90>,
    <0 0x11000100 0 0x80>;
    interrupts = <GIC_SPI 84 IRQ_TYPE_LEVEL_LOW>;
    clock-div = <16>;
    clocks = <&pericfg CLK_PERI_I2C0_PD>,
    <&pericfg CLK_PERI_AP_DMA_PD>;
    clock-names = "main", "dma";
    #address-cells = <1>;
    #size-cells = <0>;
    status = "disabled";
};
```

In dts

```
&i2c0 {
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins>;
    status = "okay";
};
```

3.3.2.4 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
```

3.3.3 GPIO

3.3.3.1 Kernel configuration:

```
CONFIG_PINCTRL_MTK_COMMON=y
CONFIG_PINCTRL_MT7622=y
```

3.3.3.2 Source code:

```
<linux>/ drivers/pinctrl/mediatek/pinctrl-mtk-common.c
<linux>/ drivers/pinctrl/mediatek/pinctrl-mt7622.c
```

3.3.3.3 DTS section:

```
&pio {
```

```
    pinctrl-names = "default";
    pinctrl-0 = <&state_default>;
    state_default:pinconf_default {
    };
/* more pinctrl sections */
}
```

3.3.3.4 Usage and testing:

For example, to configure gpio75:

```
cd /sys/devices/platform/10005000.pinctrl/
echo mode 75 1 > mt_gpio          # set GPIO mode
echo dir 75 1 > mt_gpio           # set Output mode
echo out 75 1 > mt_gpio           # set Output High
echo out 75 0 > mt_gpio           # set Output Low
```

3.3.4 SPI

3.3.4.1 Kernel configuration:

```
CONFIG_SPI_SPIDEV=y
CONFIG_SPI_MT65XX=y
```

3.3.4.2 Source code:

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

3.3.4.3 DTS section:

```
&spi0 {
    status = "okay";
    spidev: spidev@0 {
        compatible = "spidev";
        spi-max-frequency = <1000000>;
        reg = <0>;
    };
};
```

3.3.4.4 Usage and testing:

1. Build spidev_test.c to a binary file (an ELF file), and copy to board.
2. Run the command in linux shell(10000000 means 10Mhz, 30000000 means 30M):

```
./spidev_test -s 10000000 –D   /dev/spidevxxx
```

3. 3. After type step2 command, it loopback pass if SPI bus get the same values that spidev_test.c sent

3.3.5 PMIC

3.3.5.1 Kernel configuration:

```
CONFIG_REGULATOR=y
CONFIG_REGULATOR_FIXED_VOLTAGE=y
```

3.3.5.2 Source code:

```
<linux>/drivers/regulator/mt6380-regulator.c
<linux>/drivers/soc/mediatek/mtk-pmic-wrap.c
<linux>/drivers/misc/mediatek/base/power/mt7622/pmic_mt6380.c
```

3.3.5.3 DTS section:

<linux>/arch/arm64/boot/dts/mediatek/mt6380.dtsi

```
&pwrap {
    status = "okay";
    pmic: mt6380 {
        compatible = "mediatek,mt6380";
    };

    mt6380regulator: mt6380regulator {
        compatible = "mediatek,mt6380-regulator";

        mt6380_vcpu_reg: buck_vcore1 {
            regulator-name = "vcpu";
            regulator-min-microvolt = < 600000>;
            regulator-max-microvolt = <1393750>;
            regulator-ramp-delay = <6250>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_vcore_reg: buck_vcore {
            regulator-name = "vcore";
            regulator-min-microvolt = <600000>;
            regulator-max-microvolt = <1393750>;
            regulator-ramp-delay = <6250>;
            regulator-always-on;
            regulator-boot-on;
        };
```

```
        mt6380_vrf_reg: buck_vrf {
            regulator-name = "vrf";
            regulator-min-microvolt = <1200000>;
            regulator-max-microvolt = <1575000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_vm_reg: ldo_vmldo {
            regulator-name = "vmldo";
            regulator-min-microvolt = <1050000>;
            regulator-max-microvolt = <1400000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_va_reg: ldo_valdo {
            regulator-name = "valdo";
            regulator-min-microvolt = <2200000>;
            regulator-max-microvolt = <3300000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_vphy_reg: ldo_vphyldo {
            regulator-name = "vphyldo";
            regulator-min-microvolt = <1800000>;
            regulator-max-microvolt = <1800000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_vddr_reg: ldo_vddrldo {
            regulator-name = "vddr";
            regulator-min-microvolt = <1240000>;
            regulator-max-microvolt = <1840000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };

        mt6380_vt_reg: ldo_vtldo {
            regulator-name = "vadc18";
            regulator-min-microvolt = <2200000>;
```

```
            regulator-max-microvolt = <3300000>;
            regulator-ramp-delay = <0>;
            regulator-always-on;
            regulator-boot-on;
        };
    };
};
```

3.3.5.4 Usage and testing:

<none>

3.3.6 UART

3.3.6.1 Kernel configuration:

```
CONFIG_SERIAL_8250_MT6577=y
CONFIG_DMA_MTK_UART=y
```

3.3.6.2 Source code:

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

3.3.6.3 DTS section:

```
&uart1 {
  status="okay"
}
```

3.3.6.4 Usage and testing:

1. confirm the TX,RX pin whether UART mode at device tree;
2. make TX, RX PIN loopback by use cable;
3. build the uart autotest program and then run it to check the result.

3.3.7 PWM

3.3.7.1 Kernel configuration:

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

3.3.7.2 Source code:

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

3.3.7.3 DTS section:

```
&pwm {
status="okay"
}
```

3.3.7.4 Usage and testing:

PWM driver export a few sysfs entries:

```
/sys/class/pwm/
    ├──── pwmchipN/ #for each PWM chip
    ├──── export (w/o) #ask the kernel to export a PWM channel
    ├──── npwm (r/o) #number of PWM channels in this PWM chip
    ├──── pwmX/ #for each exported PWM channel
    |      ├──── duty (r/w) #duty cycle (in nanoseconds)
    |      ├──── enable (r/w) #enable/disable PWM
    |      ├──── period (r/w) #period (in nanoseconds)
    |      └────── polarity (r/w) #polarity of PWM (normal/inversed)
    └────── unexport (w/o) #return a PWM channel to the kernel
```

The following commands will make PWM4 generate squre wave:

```
echo 3 >/sys/class/pwm/pwmchip0/export
echo 200000 >/sys/class/pwm/pwmchip0/pwm3/period
echo 100000 >/sys/class/pwm/pwmchip0/pwm3/duty_cycle
echo 1 >/sys/class/pwm/pwmchip0/pwm3/enable
```

3.3.8 USB

3.3.8.1 Kernel configuration:

```
CONFIG_USB_XHCI_MTK=y   # usb3
CONFIG_PHY_MTK_TPHY=y   # u3phy
```

3.3.8.2 Source code:

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

3.3.8.3 DTS section:

```
&usb1 {
    status = "okay";
```

```
};
&u3phy1 {
    status = "okay";
};
```

3.3.8.4 Usage and testing:

Mount a USB disk, then do reading and writing test with dd command.

3.3.9 SATA

3.3.9.1 Kernel configuration:

```
CONFIG_AHCI_MTK=y
```

3.3.9.2 Source code:

```
drivers/ata/ahci_mtk.c
```

3.3.9.3 DTS section:

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

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

3.3.9.4 Usage and testing:

Connect SSD/HDD to the board, you will find a new device “/dev/block/sda1” occurs, then try to mount it.

3.3.10 Ethernet

3.3.10.1 Kernel configuration:

```
CONFIG_RAETH=y
```

3.3.10.2 Source code:

```
<liunux>/drivers/net/ethernet/raeth/
```

3.3.10.3 DTS section:

```
ethsys: syscon@1b000000 {
```

```
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "mediatek,mt7622-ethsys", "syscon";
    reg = <0 0x1b000000 0 0x1000>;
    #clock-cells = <1>;
};

eth: ethernet@1b100000 {
    compatible = "mediatek,mt7622-eth";
    reg = <0 0x1b100000 0 0x20000>;
    interrupts = <GIC_SPI 223 IRQ_TYPE_LEVEL_LOW>,
                 <GIC_SPI 224 IRQ_TYPE_LEVEL_LOW>,
                 <GIC_SPI 225 IRQ_TYPE_LEVEL_LOW>,
                 <GIC_SPI 240 IRQ_TYPE_LEVEL_LOW>;
    clocks = <&topckgen CLK_TOP_ETH_SEL>,
             <&apmixedsys CLK_APMIXED_ETH1PLL>,
             <&apmixedsys CLK_APMIXED_ETH2PLL>,
             <&apmixedsys CLK_APMIXED_SGMIPLL>,
             <&clk25m>,
             <&ethsys CLK_ETH_ESW_EN>,
             <&ethsys CLK_ETH_GP2_EN>,
             <&ethsys CLK_ETH_GP1_EN>,
             <&ethsys CLK_ETH_GP0_EN>,
             <&sgmiisys CLK_SGMII_TX250M_EN>,
             <&sgmiisys CLK_SGMII_RX250M_EN>,
             <&sgmiisys CLK_SGMII_CDR_REF>,
             <&sgmiisys CLK_SGMII_CDR_FB>;
    clock-names = "ethif", "eth1pll", "eth2pll",
                  "sgmipll", "trgpll", "esw", "gp2",
                  "gp1", "gp0", "sgmii_tx250m",
                  "sgmii_rx250m", "sgmii_cdr_ref",
                  "sgmii_cdr_fb";
    power-domains = <&scpsys MT7622_POWER_DOMAIN_ETHSYS>;
    mediatek,ethsys = <&ethsys>;
    mediatek,switch = <&gsw>;
    #reset-cells = <1>;
    #address-cells = <1>;
    #size-cells = <0>;
    status = "disabled";
};
```

3.3.10.4 Usage and testing:

<none>

3.3.11 Timer

3.3.11.1 Kernel configuration:

```
CONFIG_ARM_ARCH_TIMER=y    # arm timer
CONFIG_MTK_TIMER=y         # mtk timer
```

3.3.11.2 Source code:

```
drivers/clocksource/arm_arch_timer.c   # arm timer
drivers/clocksource/mtk_timer.c        # mtk timer
```

3.3.11.3 DTS section:

```
/* arm timer */
timer {
    compatible = "arm,armv8-timer";
    interrupt-parent = <&gic>;
    interrupts = <GIC_PPI 13 (GIC_CPU_MASK_SIMPLE(4) | IRQ_TYPE_LEVEL_HIGH)>,
                 <GIC_PPI 14 (GIC_CPU_MASK_SIMPLE(4) | IRQ_TYPE_LEVEL_HIGH)>,
                 <GIC_PPI 11 (GIC_CPU_MASK_SIMPLE(4) | IRQ_TYPE_LEVEL_HIGH)>,
                 <GIC_PPI 10 (GIC_CPU_MASK_SIMPLE(4) | IRQ_TYPE_LEVEL_HIGH)>;
};

/* mtk timer */
timer: timer@10004000 {
    compatible = "mediatek,mt7622-timer",
                 "mediatek,mt6577-timer";
    reg = <0 0x10004000 0 0x80>;
    interrupts = <GIC_SPI 152 IRQ_TYPE_LEVEL_LOW>;
    clocks = <&infracfg CLK_INFRA_APXGPT_PD>,
             <&topckgen CLK_TOP_RTC>;
    clock-names = "clk13m", "clk32k";
};
```

3.3.11.4 Usage and testing:

If arm timer is not ready, the system will not run.

As for MTK timer, you check the interrupt info via:

```
cat /proc/interrupts
```

3.3.12 Watchdog

3.3.12.1 Kernel configuration:

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

3.3.12.2 Source code:

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

3.3.12.3 DTS section:

```
watchdog: watchdog@10212000 {
    compatible = "mediatek,mt7622-wdt",
    "mediatek,mt6589-wdt";
    reg = <0 0x10212000 0 0x1000>;
    interrupts = <GIC_SPI 128 IRQ_TYPE_EDGE_FALLING>;
    #reset-cells = <1>;
};
```

3.3.12.4 Usage and testing:

MT7622's reseting is done by watchdog, so you just need:

```
reboot
```

To verify if watchdog is working.

3.3.13 eMMC

3.3.13.1 Kernel configuration:

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

3.3.13.2 Source code:

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

3.3.13.3 DTS section:

```
&mmc0 {
    pinctrl-names = "default", "state_uhs";
    pinctrl-0 = <&mmc0_pins_default>;
    pinctrl-1 = <&mmc0_pins_uhs>;
    status = "okay";
```

```
    bus_width = <8>;
    max-frequency = <50000000>;
    cap-mmc-highspeed;
    mmc-hs200-1_8v;
    vmmc-supply = <&mmc_fixed_3v3_power>;
    vqmmc-supply = <&mmc_fixed_1v8_io>;
    assigned-clocks = <&topckgen CLK_TOP_MSDC30_0_SEL>;
    assigned-clock-parents = <&topckgen CLK_TOP_UNIV48M>;
    non-removable;
};
```

3.3.13.4 Usage and testing:

Check if eMMC nodes exists:

```
cat /proc/partitions
```

Then use **dd** or **hexdump** to test **/dev/mmcblk0pX** (X is the partition index).

3.3.14 SDXC

3.3.14.1 Kernel configuration:

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

3.3.14.2 Source code:

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

3.3.14.3 DTS section:

```
&mmc1 {
    pinctrl-names = "default", "state_uhs";
    pinctrl-0 = <&mmc1_pins_default>;
    pinctrl-1 = <&mmc1_pins_uhs>;
    status = "okay";
    bus_width = <4>;
    max-frequency = <50000000>;
    cap-sd-highspeed;
    r_smpl = <1>;
    cd-gpios = <&pio 81 0>;
    vmmc-supply = <&mmc_fixed_3v3_power>;
    vqmmc-supply = <&mmc_fixed_3v3_power>;
    assigned-clocks = <&topckgen CLK_TOP_MSDC30_1_SEL>;
    assigned-clock-parents = <&topckgen CLK_TOP_UNIV48M>;
};
```

3.3.14.4 Usage and testing:

if eMMC available, **/dev/mmcblk0** is emmc block device, and **/dev/mmcblk1** is sdcard block device.

if eMMC not availble, **/dev/mmcblk0** is sdcard block device.

You can use **dd** or **eMMC** to test sdcard device.

3.3.15 Flash

3.3.15.1 Kernel configurations:

```
CONFIG_MTD_MT81xx_NOR=y         # spi-nor flash
CONFIG_MTD_NAND_MTK=y           # nand flash
```

3.3.15.2 Source code:

Flash Type	DTS Section	Driver
spi-nand	&snand	drivers/mtd/nand/mtk_snand.c
parallel-nand	&nandc	drivers/mtd/nand/mtk_nand.c
spi-nor	&nor_flash	drivers/mtd/spi-nor/mtk-quadspi.c

Table 3-1 Flash Type - DTS – Driver

3.3.15.3 DTS section:

```
/* spi-nor flash */
&nor_flash {
    status = "okay";
};
/* spi-nand flash */
&snand {
    pinctrl-0 = <&snand_pins_default>;
    status = "okay";
    flash@0 {
        partitions {
            compatible = "fixed-partitions";
            #address-cells = <1>;
            #size-cells = <1>;
            partition@0 {
                label = "preloader";
                reg=<0x00000 0x0080000>;
                read-only;
            }
            /* more patitions */
        }
    }
}
```

3.3.15.4 Usage and testing:

Check if flash device is properly detected:

```
cat /proc/mtd
```

Then use **dd** & **hexdump** to test **/dev/mtd***.

3.3.15.5 Flash layout

Since SDK V4.x, flash layout is defined in dts files.

For example: **<linux>arch/arm64/boot/dts/mediatek/<profile>.dts**

```
&snand {
    pinctrl-0 = <&snand_pins_default>;
    status = "okay";
    flash@0 {
        partitions {
            compatible = "fixed-partitions";
            #address-cells = <1>;
            #size-cells = <1>;

            partition@0 {
                label = "Preloader";
                reg = <0x00000 0x0080000>;
                read-only;
            };
            /* more partitions */
        };
    };
};
```

Predefined partitions are:

- **Preloader:** preloader code
- **ATF**: Configuration for “Air Time Fairness”
- **Bootloader**: uboot code
- **Config**: nvram configurations, including uboot’s parameters.
- **Factory**: Factory data, like calibration data, RF parameters.
- **Kernel**: Firmware partition, your firmware will be flashed into this partition. During OpenWrt/LEDE booting, this partition will be split into 3 partitions automatically:
  a) **firmware**, Linux kernel image starts here.
  b) **rootfs**: rootfs resides here.

c) **rootfs_data**: This partition takes the rest of space that is not taken by your firmware. OpenWrt uses it to store user data. It's usually formatted as JFFS2.

- **User_data**: reserved, make use of it as you wish.

3.3.15.6 Bad block management

- **MT7622**

MT7622's nand driver will take cares of bad blocks. It will setup an internal table that maps bad blocks to a reserved area in the end of flash storage.

So, above the driver, you don't have to worry about bad blocks at all. All you got are all good blocks. The only thing you need to do is to ensure you have reserved enough space (at least 2% of the full size) for the mapping.

You can change the size of reserved area by modifying the flash partition table in dts files.

3.3.15.7 Jffs2 compatibility

3.3.16 Thermal

3.3.16.1 Kernel configuration:

```
CONFIG_MTK_TURNKEY_THERMAL=y
CONFIG_THERMAL=y
```

3.3.16.2 Source code:

```
<linux>/drivers/misc/mediatek/thermal/
```

3.3.16.3 DTS section:

```
thermal: thermal@1100b000 {
    compatible = "mediatek,mt7622-thermal";
    reg = <0 0x1100b000 0 0x1000>;
    interrupts = <GIC_SPI 110 IRQ_TYPE_LEVEL_LOW>;
    clocks = <&pericfg CLK_PERI_THERM_PD>,
             <&pericfg CLK_PERI_AUXADC_PD>;
    clock-names = "therm", "auxadc";
    auxadc = <&auxadc>;
    apmixedsys = <&apmixedsys>;
    pericfg = <&pericfg>;
};
```

3.3.16.4 Usage and testing:

1. get thermal temperature:

```
cat /proc/mtktz/mtktscpu
```

2. change the thermal throttle policy, for example, to set CPU01 at 105 degree:

```
echo 1 117000 0 mtktscpu-sysrst 105000 0 cpu01 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 0 0 no-cooler 250 1 > /proc/driver/thermal/tzcpu
```

3.4 Feeds

In OpenWrt, a "feed" is a collection of packages which 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.

3.4.1 Search and install package from feeds

Pretty simple:

```
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

3.4.2 Add new feeds

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

```
src-git packages https://git.lede-project.org/feed/packages.git;lede-17.01
src-git luci https://git.lede-project.org/project/luci.git;lede-17.01
src-git routing https://git.lede-project.org/feed/routing.git;lede-17.01
src-git telephony https://git.lede-project.org/feed/telephony.git;lede-17.01
```

You can add your own feeds by add 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

3.4.3 Make your own feeds

The easiest way of building your own feeds is to fork from existing feeds.

https://github.com/openwrt/packages would be a good example to follow.

Download this repo, take a look inside, and put your package into it just like other did.

3.5 Web Interface (LuCi)

3.5.1 Install LuCi

Since SDK V3.x, LuCi is already integrated within the SDK. If you cannot find it, you can install it via “feeds”.

```
/scripts/feeds update
/scripts/feeds install luci
```

To enable LuCi in your firmware, at least you should select these 2 modules in LuC’s configuration:

```
LuCI
--> Collections
     -->  luci (*)
-->  Applications
     -->  luci-app-mtk (*)
```

3.5.2 Luci-app-mtk

This is a LuCI plugin written by MediaTek to manipulate MediaTek’s wifi configurations. It uses a lua library “/usr/lib/lua/mtkwifi.lua” to read and write wifi’s profile directly.

3.6 MTK proprietary packages

3.6.1 mii_mgr

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
```

3.6.2 qdma

3.6.3 regs

Description: 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
```

3.6.4 switch

Description:

Usage:

```
  switch mib                                        - dump mib counter
  switch dump                                       - dump switch table
  switch clear                                      - clear switch table
  switch ingress-rate on [port] [Kbps]              - set ingress rate limit on port 0~4
  switch egress-rate on [port] [Kbps]               - set egress rate limit on port 0~4
  switch ingress-rate off [port]                    - del ingress rate limit on port 0~4
  switch egress-rate off [port]                     - del egress rate limit on port 0~4
  switch igmpsnoop on                               - enable hw igmp snoop
  switch igmpsnoop off                              - disable hw igmp snoop
  switch mirror [monitor_port] [target_rx_portmask] [target_tx_portmask]    - set
port mirror
  switch phy [phy_addr]                             - get phy link status
  switch regs r [offset]                            - register read from offset
  switch regs w [offset] [value]                    - register write value to offset
  switch vlan dump                                  - dump switch vlan setting
                                                    - portmap is the order of port 0~4,
port16, port17.
  switch vlan clear                                 - clear switch vlan setting
  switch vlan set [vlan idx] [vid] [portmap]        - set vlan id and associated member.
                                                    - portmap is the order of port 0~4,
port16, port17.
  switch tag on [port] [vid]                        - keep vlan tag for egress packet on
prot 0~4, 16, 17
  switch tag off [port] [vid]                       - remove vlan tag for egress packet
on port 0~4, 16, 17
  switch test_mode [port] [mode]                    - set phy test mode. port: 0~4; mode:
1 or 4
  switch qos on                                     - enable switch qos
  switch qos off                                    - disable switch qos
  switch qos set_table2type [table] [type]          - set table qos type
  switch qos get_table2type [table]                 - get table qos type
  switch qos set_port2table [port] [table]          - set port to table mapping
  switch qos get_port2table [port]                  - get port to table mapping
```

3.6.5 ufsd_tools

3.6.6 802.1x

3.6.7 ated_ext

Description:

Usage:

```
ated [-huvd][-b<br_ifname>] [-i<driver_ifname>]
options:
  -b = bridge interface name
  -h = show this help text
  -u = respond to QA by unicast frame
  -f = daemonize ATED
  -i = driver interface name
  -v = show version
  -d = increase debugging verbosity (-dd even more)
  -q = decrease debugging verbosity (-qq even less)
  -l = path of l1profile.dat
  -x = disable mtd flash read/write by ATED feature
  -c = CLI mode to process predefined command(s)
```

Examples:

1. example 2:

```
ated -b br1 -i ra1 -v
```

2. example 3:

```
ated -u
```

3. example 4:

```
ated -d
```

4. example 5 for Dual Adapter and QA support Dual Adapter:

```
ated -i ra0 -i ra1
```

5. example 6 change dev mac address:

```
ated -b br0 -i ra0 -m mac_addr
```

6. example 7 change path of l1profile:

```
ated -l/etc/wireless/l1profile.dat
```

7. example 8 disable mtd flash read/write by ATED feature

```
ated -x
```

8. example 9 cli mode with interface, ated -irai0 -c"sync eeprom 2f00[20:2f]" (w/o l1profile.dat)

```
ated -irai0 -c"sync eeprom all"
```

9. example 10 with interface and eeprom infomation: (w/o l1profile.dat) , eeprom locate at flash offset 0x0 with length, 0x4000

```
ated -ira0 -e[0:4000]
```

3.6.8 uci2dat

Description: MediaTek's wifi drivers do not read OpenWrt's uci configuration, they use a unique configure syntax in "*.dat" files. "uci2dat" is the tool to translate OpenWrt's uci configuration into MediaTek's wifi profile.

Since SDK V4.0, "uci2dat" is deprecated because a new configuration mechanism is available for MediaTek's wifi drivers (read more from "luci-app-mtk" section). We just keep it for back compatibility.

Usage:

```
root@OpenWrt:/# uci2dat -h
uci2dat  -- a tool to translate openwrt wifi config (/etc/config/wireless)
            into ralink driver dat. typical usage:


Usage:
    uci2dat -d <dev-name> -f <dat-path>

Arguments:
    -h              help
    -d <dev-name>   device name, mt7620, eg.
    -f <dat-path>   dat file path.

Supported uci keywords:
    [uci-eyword]        [dat-keyword]           [default]
 0. region              CountryRegion
 1. aregion             CountryRegionABand
 2. country             CountryCode
 3. wifimode            WirelessMode            9
 4. txrate              TxRate                  0
 5. channel             Channel                 0
 6. basicrate           BasicRate               15
 7. beacon              BeaconPeriod            100
 8. dtim                DtimPeriod              1
 9. txpoer              TxPower                 100
10. bgprotect           BGProtection            0
11. txpreamble          TxPreamble              0
12. rtsthres            RTSThreshold            2347
13. fragthres           FragThreshold           2346
14. txburst             TxBurst                 1
```

Examples:

1. dadfa
2. setasfa

3.7 WiFi Drivers

3.7.1 Install WiFi drivers into SDK

Since SDK V4.0, MediaTek's wifi drivers are not released along with OpenWrt SDK. You need to install drivers according to the driver's user manual.

3.7.2 Init flow

There are 4 scripts involved:

- **/etc/init.d/network**, a script to operate network subsystem. Network subsystem's initialization is registered to rcS via "/etc/init.d/network". During system initialization, this script takes charge of bringing up the network subsystem, including wifi devices and drivers. This script can be called individually.
- **/sbin/wifi**, a script to operate wifi devices and drivers, it is invoked by "/etc/init.d/network". It can be called individually too.
- **/lib/wifi/<chip-specific>.lua**, chip specific operating script, it is supporting library of "/sbin/wifi" that defines how to "up/down/reload" the specific chip. This file is supposed to be be imported into "/sbin/wifi".
- **/usr/lib/lua/mtkwifi.lua**, a lua library to manipulate Mediatek wifi drivers' profile. It is used in "luci-app-mtk" and "<chip specific>.lua".

So, the overall init flow is:

Init process (procd) → rcS → /etc/rcs.d/S20network → /etc/init.d/network → /sbin/wifi → /lib/wifi/<chip-specific>.lua

3.7.3 /sbin/wifi and /sbin/wifi.legacy

OpenWrt's original wifi init script is located at "**<openwrt>/package/base-files/files/sbin/wifi**", it is written in shell. Since MTK's SDK v4.0, a different MediaTek implemtation (in lua) has replaced the original script. The original scripts is backed up at "package/base-files/files/sbin/wifi.legacy".

3.7.4 l1profile

MTK's SDK v4.0 introduces a global configuration files for MediaTek wifi drivers: **/etc/wireless/l1profile.dat**.

```
Default
INDEX0=MT7622
INDEX0_profile_path=/etc/wireless/mt7615e/mt7615e.1.dat
INDEX0_init_script=/lib/wifi/mt7615e.lua
INDEX0_init_compatible=mt7615e
INDEX0_EEPROM_offset=0x0
INDEX0_EEPROM_size=0x4000
INDEX0_EEPROM_name=e2p
INDEX0_main_ifname=ra0
INDEX0_ext_ifname=ra
INDEX0_wds_ifname=wds
```

```
INDEX0_apcli_ifname=apcli
INDEX0_mesh_ifname=mesh
INDEX0_nvram_zone=dev1
INDEX0_single_sku_path=/etc/wireless/mt7615e/mt7615e-sku.dat
INDEX0_bf_sku_path=/etc/wireless/mt7615e/mt7615e-sku-bf.dat
……
```

This file is read both by wifi drivers and "/usr/lib/lua/mtkwifi.lua", fields are explaines as below:

Field Name	Type	Description	Example
profile_path	Mandatory	L2 profile file path	INDEX0_profile_path=/etc/Wireless/MT7622AP.dat INDEX1_profile_path=/etc/Wireless/MT7615_2G.dat; /etc/Wireless/MT7615_5G.dat (DBDC mode)
EEPROM_offset	7622 Driver Mandatory	Apply on flash mode support	INDEX0_EEPROM_offset=0x0000 INDEX1_EEPROM_offset=0x4000
EEPROM_size	7622 Driver Mandatory	Apply on flash mode support	INDEX2_EEPROM_size=0x4000
EEPROM_name	7622 Driver Mandatory	Apply on flash mode support	INDEX2_EEPROM_size=0x4000
main_ifname	Mandatory	Main interface name	INDEX0_if_name=wlan0 INDEX1_if_name=wlan1;wlan2 (DBDC mode)
ext_ifname	Mandatory	MBSS interface name	INDEX0_if_name=wlan0_ INDEX1_if_name=wlan1_;wlan2_ (DBDC mode) result: wlan0_1, wlan0_2, …, wlan0_15
wds_ifname	Optional	WDS interface name	INDEX0_wds_name=wds0_ INDEX1_wds_name=wds1_;wds2_ (DBDC mode) result: wds0_0, wds0_1, …, wds0_3
apcli_ifname	Optional	AP client interface name	INDEX0_apcli_name=apcli0_ INDEX1_apcli_name=apcli1_; apcli2_ (DBDC mode) result: apcli0_0, apcli0_1, ...
mesh_ifname	Optional	Mesh interface name	INDEX0_mesh_name=mesh0_ INDEX1_mesh_name=mesh1_; mesh2_ (DBDC mode) result: mesh0_0, mesh0_1, …, mesh0_3
init_script	LEDE APP Mandatory	OpenWRT network scripts path for specific MTK Wi-Fi chip.	INDEX0_init_script=/lib/wifi/mt7615e.lua
init_compatible	LEDE APP Mandatory	Compatible chipset name for OpenWRT network setting	INDEX0_init_compatible=mt7615emt7615e
nvram_zone	LSDK APP Mandatory	Specify the sequence order and the nvram zone storage of this	INDEX0_nvram_zone=dev1 value scope: [dev1, dev2, dev3]

		Wi-Fi device.	

Table 3-2 L1profile fields

4 Development with MTK OpenWrt SDK

4.1 Creating 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 rogram\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. Then execute “make menuconfig”, you will find a new configurable option turns up under “MTK properties ---> Applications”. Select it as either “*” or “m”, now you can build “helloworld” with:

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

4.2 Using device tree (*.dts and *.dtb) (hua)

4.3 Work with patches

Here’s a much better version of working with patches in OpenWrt SDK.

https://lede-project.org/docs/guide-developer/use-patches-with-buildsystem

4.3.1 The “quilt” utility

Quilt is a powerful tool that help you to create, apply, modify a big number of patches. It is based on “diff & patch” utility, and provides stack-style management for all your patches.

You can get the basic usage here:

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

4.3.2 Make patch for the kernel

For example, if we want to modify (or add) the file “drivers/mtd/xxx.c” in kernel, let’s do it step by step:

```
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**” makes sure you can use quilt utility with the source code.
2. Enter kernel source folder. It is usually located at: “build_dir/target-<toolchain info>/linux-<platform>-<model>/linux-<version>”

3. Use “**quilt new**” command to create a new patch. After this command, a new empty patch file will be generated under “.patches” folder.
   a) Normally you should choose a number prefix that is 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 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 changes that you are going to made on this file.
5. Edit the file as you wish. You can use what every editors 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 current code, just go back to step 5.
7. Use “**quilt refresh**” to flush all the changes you made 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. Afte updating, you will find a new patch occurs in “target/linux/<platform>/patches”.

4.3.3 Make patch for a package

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

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

4.4 Customize booting sequence

4.4.1 "/etc/inittab"

If you want your init job been 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".

- **respawn** - this works just like you expect it. It starts a process and will respawn it once it has completed.
- **respawnlate** - this works like the respawn but will start the process only when the procd init is completed.
- **askfirst** - this works just like respawn but will print the line "Please press Enter to activate this console." before starting the process
- **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="
- **askconsolelate** - this works like the askconsole but will start the process only when the procd init is completed.
- **sysinit** - this will trigger procd to run the command, given as a parameter, only once. This is usually used to trigger execution of /etc/rc.d/

4.4.2 /etc/init.d and /etc/rcs.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 option "start".

After rcS finishes, system should be up and running.

Here's an 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
```

4.4.3 /etc/rc.local

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

For example, the follow "rc.local" script will print 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
```

5 FAQ

5.1 Q: Which SDK version should I use?

You are not always recommended to use the latest SDK. It depends on which chips you are using.

Here's a summary of recommended SDK release for every SoC chips.

Revision	Latest SDK	OpenWrt SDK Code Base	Linux Kernel	Release Date
MT7620	V3.4.1.0	OpenWrt 14.07	3.10.14	2016-03-24
MT7621	V3.4.1.0	OpenWrt 14.07	3.10.14	2016-03-24
MT7628	V3.4.1.0	OpenWrt 14.07	3.10.14	2016-03-24
MT7623	V4.0.0.0	Lede 17.01	4.4.92	2017-12-31
MT7622	V4.0.0.0	Lede 17.01	4.4.92	2017-12-31

Table 5-1 SoC Chips and Recommended SDK revision

If you have special concerns, you can consult your technical supporting window from MediaTek.

5.2 Q: What is factory.bin, when should I use it?

Sometimes there is a image file with the word ***factory*** in the name. If there is one, you have to flash this one over an OEM firmware (aka factory firmware, which is originally shipped with the product.). After that, you can use normal sysupgrade bin.

This reason for using factory firmware is mainly because:

- OpenWrt may use a different partition layout from OEM firmware, a specialized firmware is necessary to preserve important factory data.
- The original firmware may have some firmware verification that a normal openwrt firmware cannot pass.

5.3 Q: What's the difference between initramfs.bin and squashfs.bin? Which one should I use?

The main differences between squashfs firmware and initramfs firmware are:

- Squashfs firmware put the rootfs on flash and linux kernel will mount it during booting, while initramfs put 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 lost the changes you’ve made after reboot.**
- Squashfs firmware is supposed to be actually written into the flash storage, while initramfs firmware is supposed to be stay in RAM. You should choose the right option in uboot menu.

Normally you should use squashfs, but sometimes you may need the initramfs:

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

5.4 Q: How to install some files into rootfs?

5.4.1 Option 1: Put your file under “base-files“.

Files under “base-files” folder will be copied into rootfs during build. There are several folders named “base-files”, all of them will do.

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

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

5.5 Q: Where is the DTS file?

All dts files are located under “**<linux >/arch/arm64/mediate/dts/**”.

Which one is the one I am using? You should refer to the image make file.

Take MT7622 as an example, check "**<openwrt>/target/linux/mediatek/images/mt7622.mk**"

```
define Device/MTK-AC2600-RFB1
  DEVICE_TITLE := MTK7622 ac2600 rfb1 AP
  DEVICE_DTS := mt7622-ac2600rfb1
  DEVICE_DTS_DIR := $(DTS_DIR)/mediatek
  SUPPORTED_DEVICES := mt7622
  DEVICE_PACKAGES := kmod-usb-core kmod-usb-ohci kmod-usb-storage
endef
TARGET_DEVICES += MTK-AC2600-RFB1
```

The red line points out the dts name. Then you will find the right dts file at:
"**<linux>/arch/arm64/mediate/dts/ mt7622-ac2600rfb1.dts**"