

**MTK U-Boot (MT7621) User’s Manual**

Version: 1.3

Release date: 2020-05-26

© 2008 - 2020 MediaTek Inc.

This document contains information that is proprietary to MediaTek Inc.

Unauthorized reproduction or disclosure of this information in whole or in part is strictly prohibited.

Specifications are subject to change without notice.

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Description** |
| 1.0 | 2018-05-15 | Weijie Gao | Initial Draft. |
|  | 2018-05-16 |  | Default firmware offset correction |
| 1.1 | 2019-01-08 |  | Update to v2018.09 & add more contents |
|  | 2019-01-21 |  | Add clock driver configuration |
|  | 2019-02-15 |  | Add memtest / single core & VPE configuration |
| 1.2 | 2019-06-11 |  | Update commands & Add HTTP server |
|  | 2019-10-29 |  | Update commands & Add HS UART driver |
|  | 2020-01-03 |  | Update commands |
| 1.3 | 2020-05-20 |  | Add NMBM & NAND driver fix |
|  | 2020-05-26 |  | Add dual image support |

Table of Contents

[Document Revision History 2](#_Toc41406783)

[Table of Contents 3](#_Toc41406784)

[1 About MTK U-Boot 5](#_Toc41406785)

[1.1 What is U-Boot 5](#_Toc41406786)

[1.2 MediaTek’s U-Boot 5](#_Toc41406787)

[1.3 U-Boot Revision 5](#_Toc41406788)

[2 Using MTK U-Boot 6](#_Toc41406789)

[2.1 Prepare toolchain 6](#_Toc41406790)

[2.1.1 OpenWrt’s toolchain 6](#_Toc41406791)

[2.1.2 Buildroot’s toolchain 6](#_Toc41406792)

[2.2 U-Boot configuration 9](#_Toc41406793)

[2.2.1 Install essential packages 9](#_Toc41406794)

[2.2.2 Load preset configuration 9](#_Toc41406795)

[2.2.3 Customization 10](#_Toc41406796)

[2.3 Build 26](#_Toc41406797)

[2.4 Generated binaries 26](#_Toc41406798)

[3 U-Boot function introduction 27](#_Toc41406799)

[3.1 SPL 27](#_Toc41406800)

[3.2 MTK provided functionality 27](#_Toc41406801)

[3.2.1 Boot menu 27](#_Toc41406802)

[3.2.2 mtkboardboot command 28](#_Toc41406803)

[3.2.3 mtkupgrade command 28](#_Toc41406804)

[3.2.4 mtkload command 29](#_Toc41406805)

[3.2.5 Save tftp information for mtkupgrade and mtkload 29](#_Toc41406806)

[3.2.6 NMBM (NAND mapping block management) 30](#_Toc41406807)

[4 Tiny HTTP server (Web failsafe) 32](#_Toc41406808)

[4.1 menuconfig of failsafe command 32](#_Toc41406809)

[4.2 Using Web failsafe 32](#_Toc41406810)

[4.3 Web failsafe/HTTP server development 33](#_Toc41406811)

[4.3.1 Add new html files 33](#_Toc41406812)

[4.3.2 HTTP server programming APIs 34](#_Toc41406813)

[4.3.3 Example of URI handler 41](#_Toc41406814)

# About MTK U-Boot

## What is U-Boot

Das U-Boot (subtitled "the Universal Boot Loader" and often shortened to U-Boot) is an open source, primary boot loader used in embedded devices to package the instructions to boot the device's operating system kernel. It is available for a number of computer architectures, including 68k, ARM, Blackfin, MicroBlaze, MIPS, Nios, SuperH, PPC, RISC-V and x86.

## MediaTek’s U-Boot

MediaTek has ported MT7621 to mainline U-Boot started from 2018.

From this version, the U-Boot framework will not be touched. All added components (architectures, targets, commands, drivers) follow the standard of U-Boot.

## U-Boot Revision

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **U-Boot Release** | **MTK Internal git revision** |
| 1.0 | 2018-05-15 | 2018.03 | dccaedb |
| 1.1 | 2019-01-08 | 2018.09 | f126ba0 |
| 1.2 | 2019-06-11 | 2018.09 | 5a70d96 |
|  | 2019-10-29 | 2018.09 | 7b46b78 |
|  | 2020-01-03 | 2018.09 | 9f6f91d |
| 1.3 | 2020-05-20 | 2018.09 | 664cb74 |
|  | 2020-05-26 | 2018.09 | b178829 |

# Using MTK U-Boot

## Prepare toolchain

Generally mainline U-Boot can be compiled with all gcc whose version is newer than or equal to 4.8.

The following versions of gcc have been tested:

gcc-4.8+

gcc-5.x

gcc-6.x

gcc-7.x

gcc-8.1

If you’re using OpenWrt, you can use OpenWrt’s toolchain directly. Otherwise you can build toolchain using buildroot.

It’s recommended to use gcc-5.x or later.

### OpenWrt’s toolchain

OpenWrt’s toolchain is located in **staging\_dir/toolchain-mipsel\_1004kc+dsp\_gcc-4.8-linaro\_uClibc-0.9.33.2** relative to OpenWrt’s root source directory.

The path **staging\_dir/toolchain-mipsel\_1004kc+dsp\_gcc-4.8-linaro\_uClibc-0.9.33.2** is not unique. It’s based on the version of OpenWrt you’re using. You should view the source code to get the actual path.

For MTK’s MIPS-based WiSoCs, any toolchain targets to MIPS32 Release 2 is OK to compile U-Boot. This means either toolchain built for MT7620/MT7621/MT7628 can be used to compile this U-Boot.

The compiler binary is located in the **bin** folder, with a prefix **mipsel-openwrt-linux-** or **mipsel-openwrt-linux-uclibc-**. The **uclibc** in the prefix triplet depends on the C library used by the toolchain.

### Buildroot’s toolchain

You can also create toolchain using buildroot. With buildroot you can choose a customized gcc version.

#### Download buildroot source code

Download from <https://buildroot.org/download.html>.

For example download the buildroot-2018.02.9.tar.bz2 .

#### Configure and build buildroot

Uncompress the tar ball:

**tar -jxvf buildroot-2018.02.9.tar.bz2**

Go to buildroot’s root source directory:

**cd buildroot-2018.02.9**

Open configure menu:

**make menuconfig**

Go to submenu *Target options*. Set *Target Architecture* to **MIPS (little endian)**. Set *Target Architecture Variant* to **Generic MIPS32R2**. Choose **Use soft-float**.

/home/user/buildroot-2018.02.9/.config - Buildroot 2018.02.9 Configuration ‌

> Target options ───────────────────────────────────────────────────────────── ‌

┌──────────────────────────── Target options ─────────────────────────────┐ ‌

│ Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty │ ‌ ‌

│ submenus ----). Highlighted letters are hotkeys. Pressing <Y> │ ‌ ‌

│ selects a feature, while <N> excludes a feature. Press <Esc><Esc> to │ ‌ ‌

│ exit, <?> for Help, </> for Search. Legend: [\*] feature is selected │ ‌ ‌

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ Target Architecture (MIPS (little endian)) ---> │ │ ‌ ‌

│ │ Target Binary Format (ELF) ---> │ │ ‌ ‌

│ │ Target Architecture Variant (Generic MIPS32R2) ---> │ │ ‌ ‌

│ │ [\*] Use soft-float (NEW) │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

├─────────────────────────────────────────────────────────────────────────┤ ‌ ‌

│ <Select> < Exit > < Help > < Save > < Load > │ ‌ ‌

└─────────────────────────────────────────────────────────────────────────┘ ‌ ‌

‌ ‌

‌

(Optional) Go to submenu *Toolchain*. Choose a prefered version of gcc and binutils.

/home/user/buildroot-2018.02.9/.config - Buildroot 2018.02.9 Configuration ‌

> Toolchain ────────────────────────────────────────────────────────────────── ‌

┌─────────────────────────────── Toolchain ───────────────────────────────┐ ‌

│ Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty │ ‌ ‌

│ submenus ----). Highlighted letters are hotkeys. Pressing <Y> │ ‌ ‌

│ selects a feature, while <N> excludes a feature. Press <Esc><Esc> to │ ‌ ‌

│ exit, <?> for Help, </> for Search. Legend: [\*] feature is selected │ ‌ ‌

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ () Additional binutils options (NEW) │ │ ‌ ‌

│ │ \*\*\* GCC Options \*\*\* │ │ ‌ ‌

│ │ GCC compiler Version (gcc 6.x) ---> │ │ ‌ ‌

│ │ () Additional gcc options (NEW) │ │ ‌ ‌

│ │ [ ] Enable C++ support (NEW) │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

├─────────────────────────────────────────────────────────────────────────┤ ‌ ‌

│ <Select> < Exit > < Help > < Save > < Load > │ ‌ ‌

└─────────────────────────────────────────────────────────────────────────┘ ‌ ‌

‌ ‌

‌

Save configuratrion.

Build buildroot:

**make**

#### Using buildroot toolchain

Buildroot’s toolchain is located in **output/host** relative to buildroot’s root source directory. However this can be changed in its menuconfig.

The compiler binary is located in the **bin** folder, with a prefix **mipsel-linux-** or **mipsel-buildroot-linux-uclibc-**.

## U-Boot configuration

### Install essential packages

The mainline U-Boot requires the following packages to be installed before compiling:

swig python-dev

### Load preset configuration

Although mainline U-Boot uses menuconfig for the whole configuration now, it still provides many preset configuration files for convenience.

First uncompress the u-boot tar ball and enter its root source directory.

To use default configuration for MT7621 RFB board boot from SPI-NOR:

**make mt7621\_rfb\_defconfig**

To use default configuration for MT7621 RFB board boot from NAND:

**make mt7621\_nand\_rfb\_defconfig**

To use default configuration for MT7621 RFB 802.11ax board boot from SPI-NOR:

**make mt7621\_ax\_rfb\_defconfig**

To use default configuration for MT7621 RFB 802.11ax board boot from NAND:

**make mt7621\_nand\_ax\_rfb\_defconfig**

To use default configuration for MT7621 RFB board boot from NAND with NMBM enabled:

**make mt7621\_nmbm\_rfb\_defconfig**

To use default configuration for MT7621 RFB 802.11ax board boot from NAND with NMBM enabled:

**make mt7621\_nmbm\_ax\_rfb\_defconfig**

For details about NMBM (NAND bad block management), please refer to [section 3.2.6](#_NMBM_(NAND_mapping).

### Customization

Under U-Boot’s root source directory, execute:

**make menuconfig**

You will get a menu that lists out all configurable features of the U-Boot.

┌───────────────────── U-Boot 2018.09 Configuration ──────────────────────┐ ‌

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ Architecture select (MIPS architecture) ---> │ │ ‌ ‌

│ │ MIPS architecture ---> │ │ ‌ ‌

│ │ General setup ---> │ │ ‌ ‌

│ │ Boot images ---> │ │ ‌ ‌

│ │ API ---> │ │ ‌ ‌

│ │ Boot timing ---> │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

├─────────────────────────────────────────────────────────────────────────┤ ‌ ‌

│ <Select> < Exit > < Help > < Save > < Load > │ ‌ ‌

└─────────────────────────────────────────────────────────────────────────┘ ‌ ‌

‌ ‌

#### MT7621 platform configuration

Submenu *MIPS architecture ---> MediaTek MT7621 platform*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Target select (Support MediaTek MT7621) ---> │ │ ‌ ‌

│ │ MediaTek MT7621 platform ---> │ │ ‌ ‌

│ │ Endianness selection (Little endian) ---> │ │ ‌ ‌

│ │ CPU selection (MIPS32 Release 2) ---> │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

‌

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ Board select (MT7621 reference board) ---> │ │ ‌ ‌

│ │ (0x10000) SPL alignment size │ │ ‌ ‌

│ │ (0x30000) Maximum U-Boot size │ │ ‌ ‌

│ │ [ ] Do optional memtest after DRAM initialization │ │ ‌ ‌

│ │ [ ] Force to use single MIPS core │ │ ‌ ‌

│ │ [ ] Force to use single VPE per core │ │ ‌ ‌

│ │ CPU & DDR configuration ---> │ │ ‌ ‌

│ │ [ ] Enable NAND bad block mapping management │ │ ‌ ‌

│ │ (0x50000) Default kernel offset in the NOR │ │ ‌ ‌

│ │ (3) Delay for mtkautoboot command (seconds) │ │ ‌ ‌

│ │ [ ] Enable dual image support │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

* *Board select*:

Currently two board available:

**MT7621 reference board**: for boot from SPI-NOR

**MT7621 reference board (NAND)**: for boot from NAND

You can add your custom board.

* *SPL alignment size*:

This is used for SPL padding. The SPL part will be padded to be aligned with this size. Typically this value is set to flash’s block size:

0x1000 for SPI NOR flash with 4KiB sector support

0x10000 for SPI NOR flash with 64KiB erase block

0x20000 for NAND flash with 128KiB block size

* *Maximum U-Boot size*

This is used by the SPL when searching for the main U-Boot image. This value defines a range [0..MAX\_U\_BOOT\_SIZE] on the flash. The SPL will serch for the U-Boot image within this range. The SPL will enter emergency failsafe mode is U-Boot image is not found.

* *Do optional memtest after DRAM initialization*

Enable a prompt after DRAM initialization to allow user to start a full memory test

* *Force to use single MIPS core / Force to use single VPE per core*

Force to use single core/VPE even if the chip has dual-core.

* *Enable NAND bad block mapping management*

This option is described in [section 2.2.3.9](#_NMBM_(NAND_mapping_1).

* *Default kernel offset in the NOR/NAND*:

This configuration is from the MT7621 reference board.

This is the fallback value when the mtkboardboot command fails to find a firmware partition offset from the builtin mtdparts.

* *Delay for mtkautoboot command:*

Display time for the bootmenu.

* *Enable dual image support*

This option is described in [section 2.2.3.11](#_Dual_image_support).

Submenu *MIPS architecture ---> MediaTek MT7621 platform ---> CPU & DDR configuration:*

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ (880) CPU Frequency (MHz) │ │ ‌ ‌

│ │ DRAM Frequency (1200MHz) ---> │ │ ‌ ‌

│ │ [ ] Use user-prefered DRAM initialization parameters first │ │ ‌ ‌

│ │ [ ] Reset the chip if failed to initialize DRAM │ │ ‌ ‌

│ │ [ ] Enable verbose output during DRAMC initialization │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

* *CPU Frequency*:

The CPU frequency may be rounded up based on the crystal frequency.

For 20/40MHz crystal, the frequency is a multiple of 20MHz.

For 25MHz crystal, the frequency is a multiple of 25MHz.

* *DRAM Frequency*:

Four fixed value

* *Use user-prefered DRAM initialization parameters first*:

By default U-Boot will automatically determine the DRAM size, and uses default DDR AC Timing settings.

You can let U-Boot to initialize DRAM using a specific DDR AC Timing setting. If the actual DRAM size after initialization is not equal to the size belong to the specific DDR AC Timing setting, U-Boot will reinitialize the DRAM using a correct DDR AC Timing setting.

* *Do auto probing if user-prefered parameters fails:*

This option depends on *Use user-prefered DRAM initialization parameters first*.

Sometimes the user-preferred DRAM parameters may not match the real board (e.g. uses 128MB parameter on a board with 256MB DRAM). The DRAM initialization may fail. U-Boot will detect this situation whether DRAM initialization fails or not. When this situation occurs, U-Boot can fallback to automatic determination.

* *Reset the chip if failed to initialize DRAM*

If U-Boot failed to initialize the DDR controller and DDR chip, reset the board and retry

* *Enable verbose output during DRAMC initialization*:

This is only used for debug purpose.

Submenu *Boot media*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Support for booting from NAND flash │ │ ‌ ‌

│ │ [ ] Support for booting from ONENAND │ │ ‌ ‌

│ │ [ ] Support for booting from QSPI flash │ │ ‌ ‌

│ │ [ ] Support for booting from SATA │ │ ‌ ‌

│ │ [ ] Support for booting from SD/EMMC │ │ ‌ ‌

│ │ [\*] Support for booting from SPI flash │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Support for booting from NAND* flash if the U-Boot is boot from NAND.

Select *Support for booting from SPI flash* if the U-Boot is boot from SPI-NOR.

**This option must match the selected board.**

**Do not select multiple items.**

Submenu *Device Tree Control*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Run-time configuration via Device Tree │ │ ‌ ‌

│ │ [ ] Board-specific manipulation of Device Tree │ │ ‌ ‌

│ │ [ ] Enable run-time configuration via Device Tree in SPL │ │ ‌ ‌

│ │ [ ] Enable run-time configuration via Device Tree in TPL │ │ ‌ ‌

│ │ [ ] Enable use of a live tree │ │ ‌ ‌

│ │ Provider of DTB for DT control (Embedded DTB for DT control) │ │ ‌ ‌

│ │ (mt7621\_nor\_rfb) Default Device Tree for DT control │ │ ‌ ‌

│ │ [ ] Support embedding several DTBs in a FIT image for u-boot │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

*Default Device Tree for DT control*: Select the dts file which will be used.

dts files are located in arch/mips/dts.

**mt7621\_nor\_rfb** for boot from SPI-NOR.

**mt7621\_nand\_rfb** for boot from NAND.

You can add your custom dts file.

Submenu *Environment*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Environment in an MMC device (NEW) │ │ ‌ ‌

│ │ [ ] Environment in a NAND device │ │ ‌ ‌

│ │ [ ] Environment in a non-volatile RAM │ │ ‌ ‌

│ │ [ ] Environment in a NMBM upper MTD layer │ │ ‌ ‌

│ │ [ ] Environment is in OneNAND │ │ ‌ ‌

│ │ [ ] Environment is in remote memory space │ │ ‌ ‌

│ │ [\*] Environment is in SPI flash │ │ ‌ ‌

│ │ [ ] Environment in a UBI volume │ │ ‌ ‌

│ │ [ ] Create default environment from file │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Environment in a NAND device* if environment is stored in NAND

Select *Environment in a NMBM upper MTD layer* if environment is stored in NAND with NMBM enabled

Select *Environment is in SPI flash* if environment is stored in SPI-NOR

**This option must match the selected board.**

Environment block offset and size is defined in

include/configs/mt7621.h for SPI-NOR

include/configs/mt7621\_nand.h for NAND (w/ NMBM)

#### MT7621 device driver configuration

The following drivers are provided by MTK but not enabled in the preset configurations:

* USB driver:

Submenu *Device Drivers ---> USB support*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ --- USB support │ │ ‌ ‌

│ │ [\*] Enable driver model for USB │ │ ‌ ‌

│ │ \*\*\* USB Host Controller Drivers \*\*\* │ │ ‌ ‌

│ │ [\*] xHCI HCD (USB 3.0) support │ │ ‌ ‌

│ │ [ ] DesignWare USB3 DRD Core Support (NEW) │ │ ‌ ‌

│ │ [ ] DesignWare USB3 DRD Generic OF Simple Glue Layer (NEW) │ │ ‌ ‌

│ │ [ ] Support for PCI-based xHCI USB controller (NEW) │ │ ‌ ‌

│ │ [ ] Support for NXP Layerscape on-chip xHCI USB controller (N│ │ ‌ ‌

│ │ [\*] Support for MediaTek MT7621 on-chip xHCI USB controller (│ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Enable driver model for USB*

Select *xHCI HCD (USB 3.0) support*

Select *Support for MediaTek MT7621 on-chip xHCI USB controller*

To enable USB Mass Storage deivce support:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ \*\*\* USB peripherals \*\*\* │ │ ‌ ‌

│ │ [\*] USB Mass Storage support │ │ ‌ ‌

│ │ [ ] USB Keyboard support (NEW) │ │ ‌ ‌

│ │ [ ] USB Gadget Support (NEW) ---- │ │ ‌ ‌

│ │ [ ] USB to Ethernet Controller Drivers (NEW) ---- │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *USB Mass Storage support*

* SD driver:

Submenu *Device Drivers ---> MMC host controller support*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] MMC/SD/SDIO card support │ │ ‌ ‌

│ │ [\*] support for MMC/SD write operations (NEW) │ │ ‌ ‌

│ │ [ ] Poll for broken card detection case │ │ ‌ ‌

│ │ [\*] Enable MMC controllers using Driver Model │ │ ‌ ‌

│ │ [ ] ARM AMBA Multimedia Card Interface and compatible support (NE│ │ ‌ ‌

│ │ [ ] Enable quirks │ │ ‌ ‌

│ │ [ ] Support for HW partitioning command(eMMC) │ │ ‌ ‌

│ │ [ ] Support eMMC replay protected memory block (RPMB) (NEW) │ │ ‌ ‌

│ │ [ ] Support IO voltage configuration (NEW) │ │ ‌ ‌

│ │ [ ] Support IO voltage configuration in SPL (NEW) │ │ ‌ ‌

│ │ [ ] enable HS200 support (NEW) │ │ ‌ ‌

│ │ [ ] enable HS200 support in SPL (NEW) │ │ ‌ ‌

│ │ [\*] Output more information about the MMC (NEW) │ │ ‌ ‌

│ │ [ ] MMC debugging (NEW) │ │ ‌ ‌

│ │ [ ] Tiny MMC framework in SPL (NEW) │ │ ‌ ‌

│ │ [ ] Synopsys DesignWare Memory Card Interface (NEW) │ │ ‌ ‌

│ │ [ ] Freescale i.MX21/27/31 or MPC512x Multimedia Card support (NE│ │ ‌ ‌

│ │ [ ] Support for MMC controllers on PCI (NEW) │ │ ‌ ‌

│ │ [ ] TI OMAP High Speed Multimedia Card Interface support (NEW) │ │ ‌ ‌

│ │ [ ] Secure Digital Host Controller Interface support (NEW) │ │ ‌ ‌

│ │ [ ] Ftsdc010 SD/MMC controller Support (NEW) │ │ ‌ ‌

│ │ [\*] MediaTek SD/MMC Card Interface support │ │ ‌ ‌

│ │ [ ] Freescale/NXP eSDHC controller support │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *MMC/SD/SDIO card support*

Select *Enable MMC controllers using Driver Model*

Deselect *Enable quirks*

Deselect *Support for HW partitioning command(eMMC)*

Select *MediaTek SD/MMC Card Interface support*

#### Addition device drivers

If USB Mass Stroage or SD/MMC support is enabled, the following drivers must be also selected:

Submenu *Device Drivers*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Enable SCSI interface to SATA devices │ │ ‌ ‌

│ │ SATA/SCSI device support ---> │ │ ‌ ‌

│ │ [ ] AXI bus drivers ---- │ │ ‌ ‌

│ │ [\*] Support block devices │ │ ‌ ‌

│ │ -\*- Enable Legacy Block Device │ │ ‌ ‌

│ │ [\*] Use block device cache (NEW) │ │ ‌ ‌

│ │ [ ] Support IDE controllers │ │ ‌ ‌

│ │ [ ] Enable support for checking boot count limit ---- │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Support block devices*

Submenu *Device Drivers ---> Clock*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Enable clock driver support │ │ ‌ ‌

│ │ [ ] Enable cgu clock driver for HSDK │ │ ‌ ‌

│ │ [ ] AT91 clock drivers │ │ ‌ ‌

│ │ [ ] Enable ICS8N3QV01 VCXO driver │ │ ‌ ‌

│ │ │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Enable clock driver support*

#### File system drivers

If USB Mass Stroage or SD/MMC support is enabled, the file system drivers may be enabled to support file read/write.

Submenu *File systems*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Enable BTRFS filesystem support │ │ ‌ ‌

│ │ [ ] Enable CBFS (Coreboot Filesystem) │ │ ‌ ‌

│ │ [ ] Enable ext4 filesystem support │ │ ‌ ‌

│ │ [ ] Enable FAT filesystem support │ │ ‌ ‌

│ │ [ ] Enable JFFS2 filesystem support │ │ ‌ ‌

│ │ [ ] UBIFS silence verbose messages │ │ ‌ ‌

│ │ [ ] Enable CRAMFS filesystem support │ │ ‌ ‌

│ │ [ ] YAFFS2 filesystem support │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Choose the file systems to be used (mostly ext4 and FAT).

#### Partition types

If USB Mass Stroage or SD/MMC support is enabled, the partition type must be enabled.

Submenu *Partition Types*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Enable Partition Labels (disklabels) support │ │ ‌ ‌

│ │ [ ] Enable Apple's MacOS partition table (NEW) │ │ ‌ ‌

│ │ [ ] Enable Apple's MacOS partition table for SPL (NEW) │ │ ‌ ‌

│ │ [\*] Enable MS Dos partition table (NEW) │ │ ‌ ‌

│ │ [ ] Enable MS Dos partition table for SPL │ │ ‌ ‌

│ │ [ ] Enable ISO partition table │ │ ‌ ‌

│ │ [ ] Enable ISO partition table for SPL (NEW) │ │ ‌ ‌

│ │ [ ] Enable AMIGA partition table (NEW) │ │ ‌ ‌

│ │ [ ] Enable AMIGA partition table for SPL (NEW) │ │ ‌ ‌

│ │ [\*] Enable EFI GPT partition table │ │ ‌ ‌

│ │ (128) Number of the EFI partition entries (NEW) │ │ ‌ ‌

│ │ (0) Offset (in bytes) of the EFI partition entries (NEW) │ │ ‌ ‌

│ │ [\*] Enable EFI GPT partition table for SPL (NEW) │ │ ‌ ‌

│ │ [\*] Enable support of UUID for partition (NEW) │ │ ‌ ‌

│ │ [\*] Enable support of UUID for partition in SPL (NEW) │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Enable MS Dos partition table* for most common usage.

Select *Enable EFI GPT partition table* if necessary

#### Common commands configuration

Most common commands are already enabled in preset configurations.

Some optional commands can be enabled if necessary:

Submenu *Command line interface ---> Device access commands*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] mmc │ │ ‌ ‌

│ │ [ ] Enable support for RPMB in the mmc command (NEW) │ │ ‌ ‌

│ │ [ ] mmc swrite (NEW) │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *mmc* if SD/MMC support is enabled

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] universe - Command to set up the Turndra Universe controller │ │ ‌ ‌

│ │ [\*] usb │ │ ‌ ‌

│ │ [ ] sdp │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *usb* if USB support is enabled

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] nand │ │ ‌ ‌

│ │ [\*] nmbm │ │ ‌ ‌

│ │ [ ] mmc\_spi - Set up MMC SPI device │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *nmbm* if NMBM support is enabled

Submenu *Command line interface ---> Network commands*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] nfs │ │ ‌ ‌

│ │ [\*] mii │ │ ‌ ‌

│ │ [ ] ping │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *mii* if you want to read MII registers of MT7530’s internal PHYs.

Submenu *Command line interface ---> Filesystem commands*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Enable the 'btrsubvol' command │ │ ‌ ‌

│ │ [ ] ext2 command support │ │ ‌ ‌

│ │ [ ] ext4 command support │ │ ‌ ‌

│ │ [ ] FAT command support │ │ ‌ ‌

│ │ [ ] filesystem commands │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *ext2 command support* and/or *ext4 command support* if ext4 filesystem is enabled

Select *FAT command support* if FAT filesystem is enabled

Select *filesystem commands* if you want to use a universal file system command like ls and fsload.

#### Serial baudrate configuration

Submenu *Device Drivers* ---> *Serial drivers*:

These rates are also supported: 230400, 460800 and 921600.

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ (115200) Default baudrate │ │ ‌ ‌

│ │ [\*] Require a serial port for console │ │ ‌ ‌

│ │ [\*] Specify the port number used for console │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

#### FIT Image support

FIT Image is enabled by default in preset configurations.

There are two key options must be enabled:

Submenu *MIPS architecture ---> OS boot interface*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ -\*- Hand over legacy command line to Linux kernel │ │ ‌ ‌

│ │ -\*- Hand over legacy environment to Linux kernel │ │ ‌ ‌

│ │ [\*] Hand over a flattened device tree to Linux kernel │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Hand over a flattened device tree to Linux kernel*

Submenu *Boot images*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Enable support for Android Boot Images │ │ ‌ ‌

│ │ [\*] Support Flattened Image Tree │ │ ‌ ‌

│ │ [\*] Support SHA256 checksum of FIT image contents │ │ ‌ ‌

│ │ [ ] Enable signature verification of FIT uImages │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Support Flattened Image Tree*

#### NMBM (NAND mapping block management)

Submenu *Device Drivers* ---> *MTD Support*

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [ ] Altera Generic Quad SPI Controller │ │ ‌ ‌

│ │ [\*] Enable NAND mapping block management │ │ ‌ ‌

│ │ Default log level (1 - Info) ---> │ │ ‌ ‌

│ │ [\*] Enable MTD based NAND mapping block management │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

Select *Enable NAND mapping block management* to enable NMBM driver, which allows U-Boot to manage NAND bad blocks, including remapping bad blocks found during factory production and remapping new bad blocks during use.

Select Enable MTD based NAND mapping block management to enable registering NMBM layer to MTD device.

Submenu *MIPS architecture ---> MediaTek MT7621 platform*:

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Enable NAND bad block mapping management │ │ ‌ ‌

│ │ (1) Default ratio for reserved block for management (N/16) │ │ ‌ ‌

│ │ (256) Maximum blocks allowed for reserved block for management │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

* *Enable NAND bad block mapping management*

This configuration enables NMBM for NAND board. When enabled NMBM will be automatically

attached to the raw NAND device. All NAND operations should be done by using NMBM MTD device, or nmbm command.

* *Default ratio for reserved block for management (N/16)*

This option determines how many blocks at the high adress of NAND can be used for NMBM. For a large size NAND, 1/16 of total blocks are still too large for NMBM. For this situation please set CONFIG\_NMBM\_MAX\_BLOCKS to a proper value to limit the maximum reserved blocks.

* *Maximum blocks allowed for reserved block for management*

This option is applied after NMBM\_MAX\_RATIO to ensure maximum blocks reserved block for NMBM will not exceed the value set by this option.

#### MTD partition

MTD partition is set in preset configurations. It’s used to record the firmware offset to bootup, and used to record the offset and size of firmware/bootloader partition when doing firmware/bootloader upgrading.

Submenu *Command line interface ---> Filesystem commands*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] MTD partition support │ │ ‌ ‌

│ │ (nor0=raspi) Default MTD IDs │ │ ‌ ‌

│ │ (mtdparts=raspi:192k(u-boot),64k(u-boot-env),64k(factory),-(firmw│ │ ‌ ‌

│ │ [ ] Padd partition size to take account of bad blocks │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

For boot from SPI-NOR:

Default MTD IDs set to

nor0=raspi

Default MTD partition scheme for reference board is

mtdparts=raspi:192k(u-boot),64k(u-boot-env),64k(factory),-(firmware)

Default MTD partition scheme for reference board (using 802.11ax) is

mtdparts=raspi:256k(u-boot),64k(u-boot-env),256k(factory),-(firmware)

For boot from NAND:

Default MTD IDs set to

nand0=ranand

Default MTD partition scheme for reference board is

mtdparts=ranand:512k(u-boot),512k(u-boot-env),256k(factory),-(firmware)

Default MTD partition scheme for reference board (using 802.11ax) is

mtdparts=ranand:512k(u-boot),512k(u-boot-env),512k(factory),-(firmware)

For boot from NAND with NMBM enabled:

Default MTD IDs set to

nmbm0=nmbm0

Default MTD partition scheme for reference board is

mtdparts=nmbm0:512k(u-boot),512k(u-boot-env),256k(factory),-(firmware)

Default MTD partition scheme for reference board (using 802.11ax) is

mtdparts=nmbm0:512k(u-boot),512k(u-boot-env),512k(factory),-(firmware)

#### Dual image support

Dual image support is a mechanism for image/firmware backup and recovery.

This feature requires two MTD partitions: the main firmware partition, which store the firmware to be booted. And the backup firmware partition, which only stores the backup firmware.

When u-boot tries to boot firmware, it will first check the integrity of the firmware in both partitions.

* If both firmware are correct, u-boot will boot the firmware in main partition.
* If one of the firmware is damaged, it will be replaced by the firmware from another partition.
* If both firmware are damaged, u-boot will not the firmware.

Just for notice: currently only the kernel part is capable for integrity checking. Rootfs part has only a simple data checking.

Submenu *MIPS architecture ---> MediaTek MT7621 platform*:

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ [\*] Enable dual image support │ │ ‌ ‌

│ │ (firmware) MTD partition name of the main image │ │ ‌ ‌

│ │ (firmware\_backup) MTD partition name of the backup image │ │ ‌ ‌

│ │ [\*] Check squashfs rootfs integrity │ │ ‌ ‌

│ │ [ ] Restore kernel only │ │ ‌ ‌

│ └────┴(+)─────────────────────────────────────────────────────────────┘ │ ‌ ‌

To make dual image usable, mtdparts should be at least like this:

mtdparts=ranand:512k(u-boot),512k(u-boot-env),512k(factory),**32768k(firmware),32768k(firmware\_backup)**

* *Restore kernel only*

Select this only if the rootfs is pad to NAND erase boundary.

## Build

Under U-Boot root source directory, execute:

**make CROSS\_COMPILE=<toolchain-path-and-prefix>**

**<toolchain-path-and-prefix>** is the path and prefix of the toolchain you’re using. It points to the **bin** directory of the toolchain, with the toolchain prefix.

For example:

/home/user/buildroot-2018.02.2/output/host/bin/mipsel-linux-

## Generated binaries

Intermediate files:

u-boot.img: compressed u-boot.bin with uImage header

Final binary:

**u-boot-mt7621.bin**: spl/u-boot-spl-mtk-pad.bin + u-boot.img

**Final binary** can be burnt directly into flash.

**u-boot.img** can be used for SPL emergency recovery or be booted from memory. This file can not be booted by bootm command. Only mtkload command is capable for booting it.

# U-Boot function introduction

## SPL

SPL is a U-Boot feature. It split original u-boot image into two separate parts. The first part is used to initialize DRAM and other necessary devices, and then load the second part into memory and then run it. The second part contains all remaining u-boot functionalities. The second part is called secondary program, and the first part is called secondary program loader.

For MTK U-Boot, the SPL part has been padded to flash block boundary so that the SPL part and the secondary image can be upgraded separately. The secondary image is compressed using LZMA to reduce size.

MTK provided a specific SPL functionality, a emergency recovery method. When SPL failed to load secondary image from flash, it will try to load secondary image from serial console using ymodem protocol and then boot it directly.

This is defined in board/relink/common/spl.c.

**SPL must be enabled for all MT7621 boards. All preset configurations are SPL enabled.**

## MTK provided functionality

### Boot menu

The boot menu items are defined in board/ralink/common/cmd\_mtkautoboot.c.

It provides easy ways to upgrade bootloader/firmware via TFTP client or serial.

‌

\*\*\* U-Boot Boot Menu \*\*\* ‌

‌

1. Startup system (Default) ‌

2. Upgrade firmware ‌

3. Upgrade bootloader ‌

4. Upgrade bootloader (advanced mode) ‌

5. Load image ‌

0. U-Boot console ‌

‌

The three upgrade items are actual three mtkupgrade commands, and will be introduced in 3.2.3

### mtkboardboot command

The mtkboardboot command is defined in

board/ralink/{mt7621\_rfb,mt7621\_nand\_rfb}/cmd\_mtkboardboot.c.

This command provides the ability to bootup firmware in two ways: from MTD partition first, then pre-defined address.

The MTD partition is used by both mtkboardboot and mtkupgrade.

mtkboardboot will try to get the offset defined by MTD partition ***firmware*** first, and try to boot firmware from this place. If failed, it will try to boot firmware from a address defined by *Default kernel offset in the NOR/NAND*.

### mtkupgrade command

The mtkupgrade command is defined in board/ralink/common/cmd\_mtkupgrade.c.

It provides the ability to upgrade bootloader/firmware by prompting user.

The command’s usage:

mtkupgrade [<type>]

type - upgrade file type

bl - Bootloader

bladv - Bootloader (Advanced)

fw - Firmware

Note:

If <type> is bl, and the bootloader to be upgraded is a combined SPL image, the command will try to upgrade the secondary image only.

If <type> is bladv, and the bootloader to be upgraded is a combined SPL image, the command will prompt the user to determine whether to upgrade the whole bootloader or the secondary image only.

If bootloader is to be upgraded, the command will try to get the partition size defined by MTD partition ***u-boot*** or ***Bootloader***.

If firmware is to be upgraded, the command will try to get the partition size and offset defined by MTD partition ***firmware***.

<type> is optional in command line. If <type> is not provided, the command will prompt the user to select which part to be upgraded.

### mtkload command

The mtkload command is also defined in board/ralink/common/cmd\_mtkupgrade.c.

It provides the ability to load image (bootloader/firmware) into memory by prompting user and optionally run the image.

It’s very useful to test a initramfs based firmware or a memory-bootable bootloader.

The command has no command line parameter. It will prompt the user to input all necessary information.

Note:

If the target image is firmware or memory-bootable bootloader, this command will try to boot it directly. If the target image is a full SPL based U-Boot, this command will try to extract the memory part of U-Boot and boot it.

### Save tftp information for mtkupgrade and mtkload

The mtkupgrade and mtkload commands can use tftp as the way for uploading files. The tftp information (IP addresses, netmask, filename) can be recorded into environment. However this feature is not enabled by default.

To enable this feature, set environment variable mtkupgrade.save\_tftp\_info to yes:

**env set mtkupgrade.save\_tftp\_info yes**

**env save**

To disable this feature, just set mtkupgrade.save\_tftp\_info to anything other than yes, or just delete it:

**env del mtkupgrade.save\_tftp\_info**

**env save**

Restoring environment to default will clear all recorded information:

**env default -a**

**env save**

### NMBM (NAND mapping block management)

NMBM is used to solve to common issues with NAND:

* Skipped bad block(s) during factory manufacturing:

In factory manufacturing, U-Boot and kernel image will be combined into one single raw image.

The raw image will be written into NAND from the beginning. If bad block detected during writing, the bad block will be skipped, and the data will be shifted to the next block.

This means all data is shifted based on the beginning of the NAND.

However, in Linux kernel, NAND will be split into several partitions. For each MTD partition, the data shift is based on the beginning of current partition, not the NAND. When creating MTD partitions, bad blocks prior the partition will not be taken in to account for data shift. This will cause data in MTD partition incorrect, and often lead to bootup failure.

* New bad block(s) produced during normal use.

NMBM creates an upper layer that represent the raw image data in factory manufacturing, and mask all bad blocks. This is done by creating a block mapping table that maps upper layer blocks to raw NAND blocks.



NAND without NMBM



NAND with NMBM

To operate with NMBM upper mtd, [nmbm command](#_NMBM_(NAND_mapping_1) must be enabled.

Once NMBM has enabled, don’t use nand command as it operates with the lower raw NAND, which is now managed by NMBM.

# Tiny HTTP server (Web failsafe)

MediaTek provides a tiny HTTP server to recover firmware via Web browser. It can be triggered by run “httpd” command, or after bootm failure.

## menuconfig of failsafe command

The submenu *Enable Failsafe Web UI* is placed at the bottom of the root menuconfig.

┌───────────────────── U-Boot 2018.09 Configuration ──────────────────────┐ ‌ ‌

│ Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty │ ‌ ‌

│ submenus ----). Highlighted letters are hotkeys. Pressing <Y> │ ‌ ‌

│ includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to │ ‌ ‌

│ exit, <?> for Help, </> for Search. Legend: [\*] built-in [ ] │ ‌ ‌

│ ┌────^(-)─────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ Partition Types ---> │ │ ‌ ‌

│ │ Device Tree Control ---> │ │ ‌ ‌

│ │ (dtc) Path to dtc binary for use within mkimage │ │ ‌ ‌

│ │ Environment ---> │ │ ‌ ‌

│ │ [\*] Networking support ---> │ │ ‌ ‌

│ │ Device Drivers ---> │ │ ‌ ‌

│ │ File systems ---> │ │ ‌ ‌

│ │ Library routines ---> │ │ ‌ ‌

│ │ [ ] Unit tests ---- │ │ ‌ ‌

│ │ [ ] Enable Failsafe Web UI ---- │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

├─────────────────────────────────────────────────────────────────────────┤ ‌ ‌

│ <Select> < Exit > < Help > < Save > < Load > │ ‌ ‌

└─────────────────────────────────────────────────────────────────────────┘ ‌ ‌

│ ┌─────────────────────────────────────────────────────────────────────┐ │ ‌ ‌

│ │ --- Enable Failsafe Web UI │ │ ‌ ‌

│ │ [ ] Start Failsafe Web UI on autoboot failure (NEW) │ │ ‌ ‌

│ └─────────────────────────────────────────────────────────────────────┘ │ ‌ ‌

## Using Web failsafe

To start Web failsafe manually, run httpd command in U-Boot console. Once Web failsafe is running, the console can only accept Ctrl + C to terminate it. The console will output the URI accessed by user when running.

To access Web failsafe UI, set the computer’s IP address and netmask to the same subnet of u-boot (serverip & netmask in environment), but IP address should not be the same as u-boot’s IP address (ipaddr in environment). And then access u-boot’s IP address in a Web browser.

The default Web UI is very simple, and only provides basic function to upgrade firmware. The final upgrade procedure calls an internal subfunction of mtkupgrade, to make sure it has the same behavior of mtkupgrade.

## Web failsafe/HTTP server development

The TCP stack provided by MediaTek is net/{tcp.c,tcp.h} and include/net/tcp.h.

The HTTP server provided by MediaTek if net/httpd.c and include/httpd.h.

All Web failsafe files are located in failsafe/.

### Add new html files

Put files to be added into failsafe/fsdata

Edit failsafe/fsdata/Makefile, and add the following lines:

obj-y += real\_file.o

FILE\_real\_file.o := real\_file.ext

FSPATH\_real\_file.o := virtual\_path

The real\_file.ext is the real file name to be added.

The real\_file.o is the object file name from compiled real file name with the extension replaced with .o.

The virtual\_path is the path to be used in U-Boot failsafe. It can be any legal path name.

### HTTP server programming APIs

#### Structure types

struct httpd\_form\_value {

const char \*name;

const char \*data;

const char \*filename;

size\_t size;

};

**httpd\_form\_value** is used to record a field from a HTTP request. The field comes from HTML form.

**name** and **data** are always valid. If the field type is file, **filename** and **size** record the file’s name and actual size, and **data** points to the raw file data.

struct httpd\_form\_values {

u32 count;

struct httpd\_form\_value values[MAX\_HTTP\_FORM\_VALUE\_ITEMS];

};

**httpd\_form\_values** is used to record multiple fields from a HTTP request. **count** records the total number of fields. **values** is the list of accepted fields. The maximum fields accepted is defined by **MAX\_HTTP\_FORM\_VALUE\_ITEMS**.

struct httpd\_instance;

**httpd\_instance** is used internally by HTTP server. It is used to identify a HTTP instance as the U-Boot can have multiple HTTP instance running at different port. Users should not change it.

struct httpd\_uri\_handler {

const char \*uri;

httpd\_uri\_handler\_cb cb;

};

**httpd\_uri\_handler** is provided by user. It records the URI to be handled and the handler function pointer. It will also be passed to the handler to identify which URI is requested. **uri** points to the URI to be handler. **cb** points to the handler function.

struct httpd\_request {

enum httpd\_request\_method method;

const struct httpd\_uri\_handler \*urih;

struct httpd\_form\_values form;

};

**httpd\_request** records all useful information of the HTTP request. **methos** records whether the request is GET or POST. **urih** points to the URI handler structure registered by user. **form** records fields provided by the request.

struct http\_response\_info {

u32 code;

const char \*content\_type;

int content\_length;

const char \*location;

int connection\_close;

int chunked\_encoding;

int http\_1\_0;

};

**http\_response\_info** records all information required for making a HTTP response header. **code** is the response code (e.g. 200 OK). **content\_type** is the MIME of the payload (e.g. text/html). **content\_length** is the length of payload. **location** is used to specify the redirect location. **connection\_close** means the connection should be closed after this response being sent. **chunked\_encoding** means this connection have variable payload length. **http\_1\_0** means use HTTP/1.0 which is needed by **chunked\_encoding**.

struct httpd\_response {

enum httpd\_response\_status status;

struct http\_response\_info info;

const char \*data;

u32 size;

void \*session\_data;

};

**httpd\_response** is provided by the URI handler, and is used for HTTP server to generating the HTTP response. **status** specifies how the HTTP server should process this response. **info** is used for making the HTTP responser header. **info** is used by HTTP server only when status is **HTTP\_RESP\_STD**. **data** points to the payload. **size** is the length of the data. **session\_data** is used by the URI handler to record its private data.

struct fs\_desc {

const char \*path;

unsigned int size;

const void \*data;

};

**fs\_desc** records information of an embedded file. **path** is the virtual\_path specified in the Makefile. **size** is the size of file. **data** points to the data of the file.

#### Enumration types

enum httpd\_response\_status {

HTTP\_RESP\_NONE,

HTTP\_RESP\_STD,

HTTP\_RESP\_CUSTOM

};

**httpd\_response\_status** determines how the HTTP server should process the response provided by the URI handler.

**HTTP\_RESP\_NONE**: no response provided and the response is finished.

**HTTP\_RESP\_STD**: URI handler provided all information and data in a single response. HTTP server should generate the HTTP response header and send the header with payload. The response is finished when payload is fully sent.

**HTTP\_RESP\_CUSTOM**: URI handler provide all data to be sent. HTTP server send whatever data URI handler provided. HTTP response header must be provided by URI handler. The response is in progress and the URI handler will be called again when data is sent.

enum httpd\_request\_method {

HTTP\_GET,

HTTP\_POST

};

**httpd\_request\_method** records the method of a HTTP request.

enum httpd\_uri\_handler\_status {

HTTP\_CB\_NEW,

HTTP\_CB\_RESPONDING,

HTTP\_CB\_CLOSED

};

**httpd\_uri\_handler\_status** indicates the current status when calling URI handler.

**HTTP\_CB\_NEW**: indicates this is a new HTTP request.

**HTTP\_CB\_RESPONDING**: indicated the last response is sent. This is valid only when **status** of the response is **HTTP\_RESP\_CUSTOM**.

**HTTP\_CB\_CLOSED**: indicates the request is finished. URI handler can free any resources is allocates for this request (e.g. **session\_data**).

#### Callback type

typedef void(\*httpd\_uri\_handler\_cb)(enum httpd\_uri\_handler\_status status,

struct httpd\_request \*request,

struct httpd\_response \*response);

This is the function prototype of the URI handler.

**status** specifies the current status when calling the URI handler.

**request** records the information of the HTTP request. It is also valid in the respond stage.

**response** points to the **httpd\_response** structure and should be filled by the URI handler.

#### Global variable

extern u32 upload\_id;

The HTTP server supports store only one upload. **upload\_id** is used to identify an upload. It will change everytime a new upload is coming. This is also used to determine whether the expected upload is still valid.

#### Functions

struct httpd\_instance \*httpd\_find\_instance(u16 port);

**httpd\_find\_instance** is used to find an existing HTTP server instance by looking for the listening port.

**port**: the port that the HTTP server instance is listening.

Return NULL if not found.

struct httpd\_instance \*httpd\_create\_instance(u16 port);

**httpd\_create\_instance** is used to create a new HTTP server instance.

**port**: the port to be listened.

Return NULL if port is occupied or other error occurred (e.g. out of memory).

void httpd\_free\_instance(struct httpd\_instance \*httpd\_inst);

**httpd\_free\_instance** is used to destroy a HTTP server instance and release its resources.

**httpd\_inst**: points to a valid httpd\_instance structure.

int httpd\_free\_instance\_by\_port(u16 port);

**httpd\_free\_instance\_by\_port** is used to destroy a HTTP server instance and release its resources.

**port**: the port that the HTTP server instance is listening.

Return 0 if succeeded, -ve if not found.

int httpd\_register\_uri\_handler(struct httpd\_instance \*httpd\_inst,

const char \*uri,

httpd\_uri\_handler\_cb cb,

struct httpd\_uri\_handler \*\*returih);

**httpd\_register\_uri\_handler** is used to register a URI handler to a HTTP server instance.

**httpd\_inst**: points to a valid **httpd\_instance** structure.

**uri**: URI path to be handled.

**cb**: URI handler function pointer.

**returih**: optional. Return URI handler pointer, can be used for **httpd\_unregister\_uri\_handler**.

Return 0 if succeeded, -ve if error occurred.

Note: If more than one URI handler with the same uri path registered. Only the first URI handler will be called.

int httpd\_unregister\_uri\_handler(struct httpd\_instance \*httpd\_inst,

struct httpd\_uri\_handler \*urih);

**httpd\_unregister\_uri\_handler** is used to unregister a URI handler from a HTTP server instance.

**httpd\_inst**: points to a valid **httpd\_instance** structure.

**urih**: points to a valid **httpd\_uri\_handler** structure.

Return 0 if succeeded, -ve if error occurred.

struct httpd\_uri\_handler \*httpd\_find\_uri\_handler(

struct httpd\_instance \*httpd\_inst, const char \*uri);

**httpd\_find\_uri\_handler** is used to find an existing URI handler from a HTTP server instance.

**httpd\_inst**: points to a valid **httpd\_instance** structure.

**uri**: URI path to be matched.

Return NULL if not found.

u32 http\_make\_response\_header(struct http\_response\_info \*info, char \*buff,

u32 size);

**http\_make\_response\_header** is used to make HTTP response header. It’s usually used in customized response.

**info**: points to a valid http\_response\_info structure which contains information of the response header.

**buff**: a buffer to store the generated response header.

**size**: size of the buffer.

Return the actual size of the header. If the buffer can not hold the entire header, the header is truncated and the return size is also the actual size stored in the buffer.

struct httpd\_form\_value \*httpd\_request\_find\_value(

struct httpd\_request \*request, const char \*name);

**httpd\_request\_find\_value** is used in URI handler to fetch a field from a request.

**request**: pointer to a **httpd\_request** structure provided to the handler.

**name**: name of the field.

Return NULL if the field does not exist.

const struct fs\_desc \*fs\_find\_file(const char \*path);

**fs\_find\_file** is used to find an embeeded file.

**path**: virtual\_path specified in Makefile.

Return NULL if not found.

### Example of URI handler

For full examples, please refer to failsafe/failsafe.c.

#### Overall structure

static void uri\_handler(enum httpd\_uri\_handler\_status status,

struct httpd\_request \*request,

struct httpd\_response \*response)

{

if (status == HTTP\_CB\_NEW) {

/\* do sth. \*/

return;

}

if (status == HTTP\_CB\_RESPONDING) {

/\* do sth. \*/

return;

}

if (status == HTTP\_CB\_CLOSED) {

/\* do sth. \*/

return;

}

}

#### Output static page

static void index\_handler(enum httpd\_uri\_handler\_status status,

struct httpd\_request \*request,

struct httpd\_response \*response)

{

const struct fs\_desc \*file;

if (status == HTTP\_CB\_NEW) {

file = fs\_find\_file("index.html");

response->status = HTTP\_RESP\_STD;

response->info.code = 200;

response->info.connection\_close = 1;

response->info.content\_type = "text/html";

response->data = file->data;

response->size = file->size;

return;

}

}

#### Redirect

static void uri\_handler(enum httpd\_uri\_handler\_status status,

struct httpd\_request \*request,

struct httpd\_response \*response)

{

if (status == HTTP\_CB\_NEW) {

response->status = HTTP\_RESP\_STD;

response->info.code = 302;

response->info.connection\_close = 1;

response->info.location = "/about.html";

return;

}

}