Template:EcosystemRelease/Components

Contents

Stable: 17.11.2021 - 16:18 / Revision: 18.10.2021 - 15:39

Stable: 17:11:2021 - 10:10 | Revision: 10:10:2021 - 10:00

A quality version of this page, approved on *17 November 2021*, was based off this revision.

Contents

1 Usage

The EcosystemRelease/Components template returns **the version of all software components** delivered by an ecosystem release revision of the flow associated with this wiki (e.g. v1.y.z ,or v2.y.z, or...).

```
Usage: {{EcosystemRelease/Components | revision=<revision>}}
```

Where:

- <revision> is the **mandatory** revision of the ecosystem release, for which the version of all software components is requested.
  - The possible values for this parameter are available in the EcosystemRelease/Revision template.

The returned value is:

- a string that contains the version of all the software components delivered by the ecosystem release revision, in a form that is usable by a tooltip.
- "unknown revision", if <revision> is not supported.

This template relies on the EcosystemRelease/Component template.

2 Basic examples

You type	You get
Components version for vx.0.0 revision requested `{{EcosystemRelease/Components | revision=x.0.0}}`	Main software components: Linux kernel vx.y (example) U-Boot vx.y (example) TF-A vx.y (example) OP-TEE vx.y (example) STM32CubeMP1 Package vx.y (example) OpenEmbedded vx.y (example)
Components version for v3.0.0 revision requested `{{EcosystemRelease/Components | revision=3.0.0}}`	Main software components: Linux kernel v5.10-stm32mp-r1 (v5.10.10) U-Boot v2020.10-stm32mp-r1 TF-A v2.4-stm32mp-r1 OP-TEE v3.12.0-stm32mp-r1 STM32CubeMP1 Package v1.4.0 OpenEmbedded v3.1.5 (Dunfell)
Components version for v3.1.0 revision requested `{{EcosystemRelease/Components | revision=3.1.0}}`	Main software components: Linux kernel v5.10-stm32mp-r2 (v5.10.61) U-Boot v2020.10-stm32mp-r2 TF-A v2.4-stm32mp-r2 OP-TEE v3.12.0-stm32mp-r2 STM32CubeMP1 Package v1.5.0 OpenEmbedded v3.1.11 (Dunfell)

3 More examples

You type	You get
Components version for vx.2.0 revision requested `{{EcosystemRelease/Components | revision=x.2.0}}`	Main software components: Linux kernel vx.y (example) U-Boot vx.y (example) TF-A vx.y (example) OP-TEE vx.y (example) STM32CubeMP1 Package vx.y (example) OpenEmbedded vx.y (example) Android vx.y (example)
Components version for vx.1.0 revision requested `{{EcosystemRelease/Components | revision=x.1.0}}`	Main software components: Linux kernel vx.y (example) U-Boot vx.y (example) TF-A vx.y (example) OP-TEE vx.y (example) STM32CubeMP1 Package vx.y (example) OpenEmbedded vx.y (example) Android vx.y (example)
Unspecified revision `{{EcosystemRelease/Components}}`	Main software components: **unknown revision**
Unknown revision `{{EcosystemRelease/Components | revision=10.20.30}}`	Main software components: **unknown revision**

4 Code

Main software components: **unknown revision**

Linux® is a registered trademark of Linus Torvalds.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm® Cortex®-A

Open Portable Trusted Execution Environment

Stable: 17.11.2021 - 16:20 / Revision: 16.11.2021 - 09:08

A quality version of this page, approved on *17 November 2021*, was based off this revision.

Contents

1 Usage

The EcosystemRelease/Component template returns **the version of a software component** delivered by an ecosystem release revision of the flow associated with this wiki (e.g. v1.y.z ,or v2.y.z, or...).

```
Usage: {{EcosystemRelease/Component | revision=<revision> | component=<component> | name=<name>}}
```

Where:

- <revision> is the **mandatory** revision of the ecosystem release, for which the version of the software component is requested.
  - The possible values for this parameter are available in the EcosystemRelease/Revision template.
- <component> is the **mandatory** software component delivered by the ecosystem release revision, for which the version of the component is requested. The possible values are:
  - **Linux kernel**
  - **U-Boot**
  - **TF-A**
  - **OP-TEE**
  - **STM32CubeMPU**
  - **OpenEmbedded**
  - **Android**
- <name> is the **optional** parameter that allows to select the way the version name is displayed. Per default (if <name> is not set), the long board name (e.g; "Linux kernel vx.y-stm32mp-[...] (vx.y.z)") is displayed. The possible value for <name> is:
  - **short**: the short version name (e.g. "x.y") is displayed

The returned value is:

- a string that contains the version of the software component delivered by the ecosystem release revision.
- an empty string, if the software component is not delivered by the ecosystem release revision.
- "unknown revision", if <<revision> is not supported.
- "unknown component", if <<component> is not supported.

2 Basic examples

You type	You get
All components for the v3.1.0 revision requested `Long version names: ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=Linux kernel}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=U-Boot}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=TF-A}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=OP-TEE}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=STM32CubeMPU}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=OpenEmbedded}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=Android}}` `Short version names: ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=Linux kernel | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=U-Boot | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=TF-A | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=OP-TEE | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=STM32CubeMPU | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=OpenEmbedded | name=short}} ` `{{EcosystemRelease/Component |` `revision=3.1.0 | component=Android | name=short}}`	Long version names: Linux kernel v5.10-stm32mp-r2 (v5.10.61) U-Boot v2020.10-stm32mp-r2 TF-A v2.4-stm32mp-r2 OP-TEE v3.12.0-stm32mp-r2 STM32CubeMP1 Package v1.5.0 OpenEmbedded v3.1.11 (Dunfell) Short version names: 5.10 2020.10 2.4 3.12.0 1.5.0 3.1

You type	You get
All components for the v3.0.0 revision requested `Long version names: {{EcosystemRelease/Component	revision=3.0.0	component=Linux kernel}} {{EcosystemRelease/Component	revision=3.0.0	component=U-Boot }} {{EcosystemRelease/Component	revision=3.0.0	component=TF-A}} {{EcosystemRelease/Component	revision=3.0.0	component=OP-TEE }} {{EcosystemRelease/Component	revision=3.0.0	component=STM32CubeMPU}} {{EcosystemRelease/Component	revision=3.0.0	component=OpenEmbedded}} {{EcosystemRelease/Component	revision=3.0.0	component=Android}} Short version names: {{EcosystemRelease/Component	revision=3.0.0	component=Linux kernel	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=U-Boot	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=TF-A	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=OP-TEE	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=STM32CubeMPU	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=OpenEmbedded	name=short}} {{EcosystemRelease/Component	revision=3.0.0	component=Android	name=short}}` 	Long version names: Linux kernel v5.10-stm32mp-r1 (v5.10.10) U-Boot v2020.10-stm32mp-r1 TF-A v2.4-stm32mp-r1 OP-TEE v3.12.0-stm32mp-r1 STM32CubeMP1 Package v1.4.0 OpenEmbedded v3.1.5 (Dunfell) Short version names: 5.10 2020.10 2.4 3.12.0 1.4.0 3.1
Linux kernel version for vx.0.0 revision requested	

You type	You get

<pre>{{EcosystemRelease/Component | revision=x.0.0 | component=Linux kernel}}
 {{EcosystemRelease/Component | revision=x.0.0 | component=Linux kernel | name=short}}</pre>
	Linux kernel vx.y (example)
	x.y

U-Boot version for vx.0.0 revision requested
<pre>{{EcosystemRelease/Component | revision=x.0.0 | component=U-Boot }}
 {{EcosystemRelease/Component | revision=x.0.0 | component=U-Boot | name=short}}</pre>
	U-Boot vx.y (example)
	x.y

TF-A version for vx.0.0 revision requested
<pre>{{EcosystemRelease/Component | revision=x.0.0 | component=TF-A}}
 {{EcosystemRelease/Component | revision=x.0.0 | component=TF-A | name=short}}</pre>
	TF-A vx.y (example)
	x.y

OP-TEE version for vx.0.0 revision requested
<pre>{{EcosystemRelease/Component | revision=x.0.0 | component=OP-TEE }}
 {{EcosystemRelease/Component | revision=x.0.0 | component=OP-TEE | name=short}}</pre>
	OP-TEE vx.y (example)
	x.y

STM32CubeMPU version for vx.0.0 revision requested
<pre>{{EcosystemRelease/Component | revision=x.0.0 | component=STM32CubeMPU}}
 {{EcosystemRelease/Component | revision=x.0.0 | component=STM32CubeMPU | name=short}}</pre>
	STM32CubeMP1 Package vx.y (example)
	x.y

You type	You get
OpenEmbedded version for vx.0.0 revision requested `{{EcosystemRelease/Component | revision=x.0.0 | component=OpenEmbedded}}  {{EcosystemRelease/Component | revision=x.0.0 | component=OpenEmbedded | name=short}}`	OpenEmbedded vx.y (example) x.y
Android version for vx.1.0 revision requested `{{EcosystemRelease/Component | revision=x.1.0 | component=Android}}  {{EcosystemRelease/Component | revision=x.1.0 | component=Android | name=short}}`	Android vx.y (example) x.y

3 More examples

You type	You get
Linux kernel version for vx.2.0 revision requested `{{EcosystemRelease/Component | revision=x.2.0 | component=Linux kernel}}`	Linux kernel vx.y (example)
Linux kernel version for vx.1.0 revision requested `{{EcosystemRelease/Component | revision=x.1.0 | component=Linux kernel}}`	Linux kernel vx.y (example)
Android version for the vx.0.0 revision requested: Android is not supported by this ecosystem release revision `{{EcosystemRelease/Component | revision=x.0.0 | component=Android}}`	
U-Boot version for an unspecified revision requested `{{EcosystemRelease/Component | component=U-Boot}}`	unknown revision
TF-A version for an unknown revision requested `{{EcosystemRelease/Component | revision=10.20.30 | component=TF-A}}`	unknown revision
Unspecified component	

You type	You get
`{{EcosystemRelease/Component | revision=x.0.0}}`	unknown component
Unknown component `{{EcosystemRelease/Component | revision=x.0.0 | component=not supported}}`	unknown component

4 Code

unknown component

Linux® is a registered trademark of Linus Torvalds.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm® Cortex®-A

Open Portable Trusted Execution Environment

Stable: 17.11.2021 - 16:18 / Revision: 15.11.2021 - 08:04

A quality version of this page, approved on *17 November 2021*, was based off this revision.

Contents

1 Usage

The EcosystemRelease/Revision template returns **the status for an ecosystem release revision** in the flow associated with this wiki (e.g. v1.y.z ,or v2.y.z, or...), or **the revision (*x.y.z*) of the latest ecosystem release**.

```
Usage: {{EcosystemRelease/Revision | revision=<revision>}}
```

Where:

- <revision> is the **mandatory** revision of the ecosystem release for which a status is requested. The possible values are:
  - **latest** to get the revision (*x.y.z*) of the latest ecosystem release.
  - **2.0.0** (June 2020 ecosystem release revision).
  - **2.1.0** (November 2020 ecosystem release revision).
  - **3.0.0** (March 2021 ecosystem release revision).
  - **3.1.0** (November 2021 ecosystem release revision).
  - x.0.0, x.1.0, x.2.0 and 0.y.0 values are reserved to the examples and the helper files.

The returned value is:

- ***x.y.z*** (revision of the latest ecosystem release), if <revision> is set to **latest**.
- **latest**, if <revision> is the latest one, in the flow associated with this wiki.
- **legacy**, if <revision> is a legacy one, in the flow associated with this wiki.
- **next**, if <revision> is a planned one, in the flow associated with this wiki.
- **former**, if <revision> is part of a former flow that is not the one associated with this wiki.
- **unknown**, if <revision> is not supported.

2 Basic examples

You type	You get
vx.0.0 revision (legacy) `{{EcosystemRelease/Revision | revision=x.0.0}}`	legacy
Revision (*x.y.z*) of the latest ecosystem release `{{EcosystemRelease/Revision | revision=latest}}`	3.1.0
Revision (*x.y.z*) of the next ecosystem release `{{EcosystemRelease/Revision | revision=3.1.0}}`	latest
Revision (*x.y.z*) of the next ecosystem release `{{EcosystemRelease/Revision | revision=next}}`	unknown
Revision (*x.y.z*) of the latest ecosystem release `{{EcosystemRelease/Revision | revision=3.0.0}}`	legacy
Revision (*x.y.z*) of the latest ecosystem release `{{EcosystemRelease/Revision | revision=2.1.0}}`	former
Revision (*x.y.z*) of the latest ecosystem release	

You type	You get
`{{EcosystemRelease/Revision | revision=2.0.0}}`	former
Revision (*x.y.z*) of the latest ecosystem release `{{EcosystemRelease/Revision | revision=1.0.0}}`	former

3 More examples

You type	You get
vx.2.0 revision (next) `{{EcosystemRelease/Revision | revision=x.2.0}}`	next
vx.1.0 revision (latest) `{{EcosystemRelease/Revision | revision=x.1.0}}`	latest
v0.y.0 revision (former) `{{EcosystemRelease/Revision | revision=0.y.0}}`	former
Unknown revision `{{EcosystemRelease/Revision | revision=10.20.30}}`	unknown
Unspecified revision `{{EcosystemRelease/Revision}}`	unknown

4 Code

unknown
Stable: 19.10.2021 - 13:54 / Revision: 19.10.2021 - 13:54

A quality version of this page, approved on *19 October 2021*, was based off this revision.

Contents

1 Das U-Boot

Das U-Boot ("the Universal Boot Loader" or U-Boot) is an open-source bootloader that can be used on ST boards to initialize the platform and load the Linux® kernel.

- Official website: https://www.denx.de/wiki/U-Boot
- Official manual: U-Boot project documentation and https://www.denx.de/wiki/DULG/Manual
- Official **source code** is available under git repository at [1]

Read the README file before starting using U-Boot. It covers the following topics:

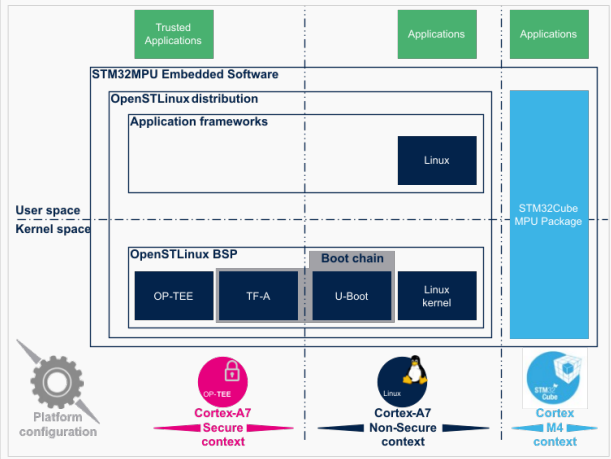
- source file tree structure
- description of CONFIG defines
- instructions for building U-Boot
- brief description of the Hush shell
- list of common environment variables

Do go further, read the documentations available in doc/ and the documentation generated by `make htmldocs` [1].

2 U-Boot overview

The STM32 MPU boot chain uses Trusted Firmware-A (TF-A) as FSBL and U-Boot as SSBL.

The same U-Boot source can also generate an alternate FSBL named SPL. The boot chain becomes: SPL as FSBL and U-Boot as SSBL.



Zoom out to STM32MPU Embedded Software

Warning

This alternate boot chain with SPL cannot be used for product development.

2.1 SPL: alternate FSBL

2.1.1 SPL description

The **U-Boot SPL** or **SPL** is an alternate first stage bootloader (FSBL).
It is a small binary (bootstrap utility) generated from the U-Boot source and stored in the internal limited-size embedded RAM.
SPL main features are the following:

- It is loaded by the ROM code.
- It performs the initial CPU and board configuration (clocks and DDR memory).
- It loads the SSBL (U-Boot) into the DDR memory.

2.1.2 SPL restrictions

Warning

SPL cannot be used for product development.

SPL is provided only as an example of the simplest FSBL with the objective to support upstream U-Boot development. However, several known limitations have been identified when SPL is used in conjunction with the minimal secure monitor provided within U-Boot for basic boot chain. These limitations apply to:

- power management
- secure access to registers
- limited features (STM32CubeProgrammer / boot from NAND Flash memory)
- SCMI support for clock and reset (not compatible with latest Linux kernel device tree)

There is no workaround for these limitations.

2.1.3 SPL execution sequence

SPL executes the following main steps in SYSRAM:

- **board_init_f()**: driver initialization including DDR initialization (mininimal stack and heap: CONFIG_SPL_STACK_R_MALLOC_SIMPLE_LEN)
- configuration of heap in DDR memory (CONFIG_SPL_SYS_MALLOC_F_LEN)
- **board_init_r()**: initialization of the other drivers activated in the SPL device tree
- loading and execution of U-Boot (or Kernel in Falcon mode[2]: README.falcon).

2.2 U-Boot: SSBL

2.2.1 U-Boot description

U-Boot is the second-stage bootloader (SSBL) of boot chain for STM32 MPU platforms.

SSBL main features are the following:

- It is configurable and expendable.
- It features a simple command line interface (CLI), allowing users to interact over a serial port console.
- It provides scripting capabilities.
- It loads the kernel into RAM and gives control to the kernel.
- It manages several internal and external devices such as NAND and NOR Flash memories, Ethernet and USB.
- It supports the following features and commands:
  - File systems: FAT, UBI/UBIFS, JFFS
  - IP stack: FTP
  - Display: LCD, HDMI, BMP for splashcreen
  - USB: host (mass storage) or device (DFU stack)

2.2.2 U-Boot execution sequence

U-Boot executes the following main steps in DDR memory:

- **Pre-relocation** initialization (common/board_f.c): minimal initialization (such as CPU, clock, reset, DDR and console) running at the CONFIG_SYS_TEXT_BASE load address.
- **Relocation**: copy of the code to the end of DDR memory.
- **Post-relocation initialization**:(common/board_r.c): initialization of all the drivers.
- **Command execution** through autoboot (CONFIG_AUTOBOOT) or console shell.
  - Execution of the boot command (by default bootcmd=CONFIG_BOOTCOMMAND):

  for example, execution of the command `bootm` to:
    - load and check images (such as kernel, device tree and ramdisk)
    - fixup the kernel device tree
    - install the secure monitor (optional) or
    - pass the control to the Linux kernel (or to another target application)

3 U-Boot configuration

The U-Boot binary configuration is based on

- **Kbuild infrastructure** (as in Linux Kernel, you can use `make menuconfig` in U-Boot)

The configurations are based on:

  - options defined in Kconfig files (CONFIG_ compilation flags)
  - the selected configuration file: configs/stm32mp*_defconfig

- **other compilation flags** defined in include/configs/stm32mp*.h (these flags are progressively migrated to Kconfig)

The file name is configured through CONFIG_SYS_CONFIG_NAME.

For STM32MP15x lines, the include/configs/stm32mp1.h file is used.

- DeviceTree: U-Boot binaries include a device tree blob that is parsed at runtime

All the configuration flags (prefixed by CONFIG_) are described in the source code, either in the README file or in the documentation directory .

For example, CONFIG_SPL activates the SPL compilation.

Hence to compile U-Boot, select the <target> and the device tree for the board in order to choose a predefined configuration.

Refer to #U-Boot_build for examples.

3.1 Kbuild

Like the kernel, the U-Boot build system is based on configuration symbols (defined in Kconfig files). The selected values are stored in a **.config** file located in the build directory, with the same makefile target. .

Proceed as follows:

- Select a predefined configuration (defconfig file in configs directory) and generate the first **.config**:

```
PC $> make <config>_defconfig.
```

- Change the U-Boot compile configuration (modify .config) by using one of the following five `make` commands:

```
PC $> make menuconfig --> menu based program
PC $> make config  --> line-oriented configuration
PC $> make xconfig --> QT program[3]
PC $> make gconfig --> GTK program
PC $> make nconfig --> ncurse menu based program
```

You can then compile U-Boot with the updated .config.

Warning: the modification is performed locally in the build directory. It will be lost after a `make distclean`.

Save your configuration to be able to use it as a defconfig file:

```
PC $> make savedefconfig
```

This target saves the current config as a defconfig file in the build directory. It can then be compared with the predefined configuration (configs/stm32mp*defconfig).

The other makefile targets are the following:

```
 PC $> make help
....
Configuration targets:
  config          - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based
                    program
  menuconfig          - Update current config utilising a menu based program
  xconfig          - Update current config utilising a Qt based front-end
  gconfig          - Update current config utilising a GTK+ based front-end
  oldconfig          - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  defconfig          - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig           - New config where all options are answered with no
  allyesconfig           - New config where all options are accepted with yes
  allmodconfig           - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig          - New config with random answer to all options
  listnewconfig   - List new options
  olddefconfig           - Same as oldconfig but sets new symbols to their
                   default value without prompting
```

3.2 Device tree

Refer to doc/README.fdt-control for details.

The board device tree has the same binding as the kernel. It is integrated within the U-Boot binaries: u-boot.bin

- By default, it is appended at the end of the code (CONFIG_OF_SEPARATE).
- It can be embedded in the U-Boot binary (CONFIG_OF_EMBED). This is particularly useful for debugging since it enables easy .elf file loading.

The U-Boot device tree (u-boot.dtb) can be also provided as external file loaded by FSBL when U-Boot code is started (u-boot-nodtb.bin: code without device tree): device tree address is provided as boot parameter (in r2 register).

A default device tree is available in the defconfig file (by setting CONFIG_DEFAULT_DEVICE_TREE).

You can either select another supported device tree using the DEVICE_TREE make flag. For stm32mp boards, the corresponding file is <dts-file-name>.dts in arch/arm/dts/stm32mp*.dts , with <dts-file-name> set to the full name of the board:

```
PC $> make DEVICE_TREE=<dts-file-name>
```

or provide a device tree blob (dtb file) resulting from the dts file compilation, by using the EXT_DTB option:

```
PC $> make EXT_DTB=boot/<dts-file-name>.dtb
```

The SPL device tree is also generated from this device tree. However to reduce its size, the U-Boot makefile uses the fdtgrep tool to parse the full U-Boot DTB and identify all the drivers required by SPL.

To do this, U-Boot uses specific device-tree flags to determine if the associated driver is initialized prior to U-Boot relocation and /or if the associated node is present in SPL :

- **u-boot,dm-pre-reloc** => present in SPL, initialized before relocation in U-Boot
- **u-boot,dm-pre-proper** => initialized before relocation in U-Boot
- **u-boot,dm-spl** => present in SPL

In the device tree used by U-Boot, these flags **need to be added in all the nodes** used in SPL or in U-Boot before relocation, and for all used handles (clock, reset, pincontrol).

To obtain a device tree file **<dts-file-name>.dts** that is identical to the Linux kernel one, these U-Boot properties are only added for ST boards in the add-on file **<dts-file-name>-u-boot.dtsi**. This file is automatically included in **<dts-file-name>.dts** during device tree compilation (this is a generic U-Boot Makefile behavior).

4 U-Boot command line interface (CLI)

Refer to U-Boot Command Line Interface.

If CONFIG_AUTOBOOT is activated, you have CONFIG_BOOTDELAY seconds (2s by default, 1s for ST configuration) to enter the console by pressing any key, after the line below is displayed and bootcmd is executed (CONFIG_BOOTCOMMAND):

```
Hit any key to stop autoboot:  2
```

4.1 Commands

The commands are defined in cmd/*.c . They are activated through the corresponding **CONFIG_CMD_*** configuration flag.

Use the `help` command in the U-Boot shell to list the commands available on your device:

```
Board $> help
```

Below the list of all commands extracted from U-Boot Manual (**not-exhaustive**):

- Information Commands
  - bdinfo - prints Board Info structure
  - coninfo - prints console devices and information
  - flinfo - prints Flash memory information
  - iminfo - prints header information for application image
  - help - prints online help
- Memory Commands
  - base - prints or sets the address offset
  - crc32 - checksum calculation
  - cmp - memory compare
  - cp - memory copy
  - md - memory display
  - mm - memory modify (auto-incrementing)
  - mtest - simple RAM test
  - mw - memory write (fill)
  - nm - memory modify (constant address)
  - loop - infinite loop on address range
- Flash Memory Commands
  - cp - memory copy
  - flinfo - prints Flash memory information
  - erase - erases Flash memory
  - protect - enables or disables Flash memory write protection
  - mtdparts - defines a Linux compatible MTD partition scheme
- Execution Control Commands
  - source - runs a script from memory
  - bootm - boots application image from memory

- go - starts application at address 'addr'
- Download Commands
  - bootp - boots image via network using BOOTP/TFTP protocol
  - dhcp - invokes DHCP client to obtain IP/boot params
  - loadb - loads binary file over serial line (kermit mode)
  - loads - loads S-Record file over serial line
  - rarpboot- boots image via network using RARP/TFTP protocol
  - tftpboot- boots image via network using TFTP protocol
- Environment Variables Commands
  - printenv- prints environment variables
  - saveenv - saves environment variables to persistent storage
  - setenv - sets environment variables
  - run - runs commands in an environment variable
  - bootd - default boot, that is run 'bootcmd'
- Flattened Device Tree support
  - fdt addr - selects the FDT to work on
  - fdt list - prints one level
  - fdt print - recursive printing
  - fdt mknode - creates new nodes
  - fdt set - sets node properties
  - fdt rm - removes nodes or properties
  - fdt move - moves FDT blob to new address
  - fdt chosen - fixup dynamic information
- Special Commands
  - i2c - I2C sub-system
- Storage devices
- Miscellaneous Commands
  - echo - echoes args to console
  - reset - performs a CPU reset
  - sleep - delays the execution for a predefined time
  - version - prints the monitor version

To add a new command, refer to doc/README.commands .

4.2 U-Boot environment variables

The U-Boot behavior is configured through environment variables.

Refer to Manual and README / Environment Variables.

On the first boot, U-Boot uses a default environment embedded in the U-Boot binary. You can modify it by changing the content of CONFIG_EXTRA_ENV_SETTINGS in your configuration file (for example ./include/configs/stm32mp1.h) (see README / - Default Environment).

This environment can be modified and saved in the boot device. When it is present, it is loaded during U-Boot initialization:

- To boot from *e*•MMC/SD card (CONFIG_ENV_IS_IN_MMC): at the end of the partition indicated by config field "u-boot,mmc-env-partition" in device-tree (for ST boards: partition named "fip" in ecosystem release v3.0.0 with FIP support or partition named "ssbl" without FIP support).

- To boot from NAND Flash memory (CONFIG_ENV_IS_IN_UBI): in the two UBI volumes "config" (CONFIG_ENV_UBI_VOLUME) and "config_r" (CONFIG_ENV_UBI_VOLUME_REDUND).
- To boot from NOR Flash memory (CONFIG_ENV_IS_IN_SPI_FLASH): the u-boot_env mtd parttion (at offset CONFIG_ENV_OFFSET).

4.2.1 env command

The `env` command allows displaying, modifying and saving the environment in U-Boot console.

```
Board $> help env
env - environment handling commands

Usage:
env default [-f] -a - [forcibly] reset default environment
env default [-f] var [...] - [forcibly] reset variable(s) to their default values
env delete [-f] var [...] - [forcibly] delete variable(s)
env edit name - edit environment variable
env exists name - tests for existence of variable
env print [-a | name ...] - print environment
env print -e [name ...] - print UEFI environment
env run var [...] - run commands in an environment variable
env save - save environment
env set -e name [arg ...] - set UEFI variable; unset if 'arg' not specified
env set [-f] name [arg ...]
```

Example: proceed as follows to restore the default environment and save it. This is useful after a U-Boot upgrade:

```
Board $> env default -a
Board $> env save
```

You can also use the command activated by CONFIG_CMD_ERASEENV:

```
Board $> env erase
```

4.2.2 bootcmd

"bootcmd" variable is the autoboot command. It defines the command executed when U-Boot starts (CONFIG_BOOTCOMMAND).

For stm32mp, CONFIG_BOOTCOMMAND="run bootcmd_stm32mp":

```
Board $> env print bootcmd
 bootcmd=run bootcmd_stm32mp
```

"bootcmd_stm32mp" is a script that selects the command to be executed for each boot device (see ./include/configs/stm32mp1.h), based on generic distro scripts:

- To boot from a serial/usb device: execute the `stm32prog` command.
- To boot from an *e*•MMC, SD card: boot only on the same device (bootcmd_mmc...).
- To boot from a NAND Flash memory: boot on ubifs partition on the NAND memory (bootcmd_ubi0).
- To boot from a NOR Flash memory: use the SD card (on SDMMC 0 on ST boards with bootcmd_mmc0)

```
Board $> env print bootcmd_stm32mp
```

You can then change this configuration:

- either permanently in your board file
  - default environment by CONFIG_EXTRA_ENV_SETTINGS (see ./include/configs/stm32mp1.h)
  - change CONFIG_BOOTCOMMAND value in your defconfig

```
CONFIG_BOOTCOMMAND="run bootcmd_mmc0"
```

```
CONFIG_BOOTCOMMAND="run distro_bootcmd"
```

- or temporarily in the saved environment:

```
Board $> env set bootcmd run bootcmd_mmc0
Board $> env save
```

Note: To reset the environment to its default value:

```
Board $> env default bootcmd
Board $> env save
```

4.3 Generic Distro configuration

Refer to doc/README.distro for details.

This feature is activated by default on ST boards (CONFIG_DISTRO_DEFAULTS):

- one boot command (bootmcd_xxx) exists for each bootable device.
- U-Boot is independent from the Linux distribution used.
- bootcmd is defined in ./include/config_distro_bootcmd.h

When DISTRO is enabled, the command that is executed by default is include/config_distro_bootcmd.h :

```
bootcmd=run distro_bootcmd
```

This script tries any device found in the 'boot_targets' variable and executes the associated bootcmd.

Example for mmc0, mmc1, mmc2, pxe and ubifs devices:

```
bootcmd_mmc0=setenv devnum 0; run mmc_boot
bootcmd_mmc1=setenv devnum 1; run mmc_boot
bootcmd_mmc2=setenv devnum 2; run mmc_boot
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcmd_ubifs0=setenv devnum 0; run ubifs_boot
```

U-Boot searches for an **extlinux.conf** configuration file for each bootable device. This file defines the kernel configuration to be used with `bootm` command:

- bootargs
- files to start the OS:
  - kernel (uImage) + device tree + ramdisk files (optional)

- FIT image, including all these needed files (for details see doc/uImage.FIT/howto.tx)

4.4 U-Boot scripting capabilities

"Script files" are command sequences that are executed by the U-Boot command interpreter. This feature is particularly useful to configure U-Boot to use a real shell (hush) as command interpreter.

See U-Boot script manual for an example.

5 U-Boot build

See U-Boot Documentation.

5.1 Prerequisites

- a PC with Linux and tools:
  - see PC_prerequisites
  - #ARM cross compiler
- U-Boot source code
  - the latest STMicroelectronics U-Boot version
    - tar.xz file from Developer Package (for example STM32MP1) or from latest release on ST github [4]
    - from GITHUB[5], with `git` command

```
PC $> git clone https://github.com/STMicroelectronics/u-boot
```

  - from the Mainline U-Boot in official GIT repository [6]

```
PC $> git clone https://source.denx.de/u-boot/u-boot.git
```

5.2 ARM cross compiler

A cross compiler [7] must be installed on your Host (X86_64, i686, ...) for the ARM targeted Device architecture. In addition, the $PATH and $CROSS_COMPILE environment variables must be configured in your shell.

You can use gcc for ARM, available in:

- the SDK toolchain (see Cross-compile with OpenSTLinux SDK)

PATH and CROSS_COMPILE are automatically updated.

- an existing package

For example, install gcc-arm-linux-gnueabihf on Ubuntu/Debian: (**PC $>** sudo apt-get.

- an existing toolchain:
  - latest gcc toolchain provided by arm (https://developer.arm.com/open-source/gnu-toolchain/gnu-a/downloads/)
  - gcc v7 toolchain provided by linaro: (https://www.linaro.org/downloads/)

For example, to use *gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf.tar.xz* from arm, extract the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf/bin:$PATH
PC $> export CROSS_COMPILE=arm-none-linux-gnueabihf-
```

For example, to use **gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz**
from https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabi/
Unzip the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin:$PATH
PC $> export CROSS_COMPILE=arm-linux-gnueabi-
```

5.3 Compilation

In the U-Boot source directory, select the defconfig for the <target> and the <device tree> for your board and then execute the `make all` command:

```
PC $> make <target>_defconfig
PC $> make DEVICE_TREE=<device tree> all
```

Use help to list other targets than all:

```
PC $> make help
```

Optionally

- **KBUILD_OUTPUT** can be used to change the output build directory in order to compile several targets in the source directory. For example:

```
PC $> export KBUILD_OUTPUT=<path>
```

- **DEVICE_TREE** can also be exported to your environment when only one board is supported. For example:

```
PC $> export DEVICE_TREE=<device-tree>
```

The result is the following:

```
PC $> export KBUILD_OUTPUT=<path>
PC $> export DEVICE_TREE=<device tree>
PC $> make <target>_defconfig
PC $> make all
```

Examples from STM32MP15 U-Boot:

The boot chain for STM32MP15x lines use stm32mp15_trusted_defconfig:

```
PC $> make stm32mp15_trusted_defconfig
PC $> make DEVICE_TREE=stm32mp157f-dk2 all
```

```
PC $> export KBUILD_OUTPUT=../build/stm32mp15_trusted
PC $> export DEVICE_TREE=stm32mp157c-ev1
PC $> make stm32mp15_trusted_defconfig
PC $> make all
```

5.4 Output files

The resulting U-Boot files are located in your build directory (U-Boot or KBUILD_OUTPUT).

Since ecosystem release v3.0.0 , two U-Boot files are used by ST boards to generate FIP used by FSBL TF-A, with or without OP-TEE support:

- BL33_CFG=u-boot.dtb : the U-Boot device tree, selected by DEVICE_TREE, loaded by TF-A BL2 and amended by secure monitor (SPMIN or OP-TEE)
- BL33=u-boot-nodtb.bin: the U-Boot executable, loaded by TF-A BL2 started by secure monitor with BL33_CFG as parameter

Nota: All the compiled device tree are available in $KBUILD_OUTPUT/arch/arm/dts/*.dtb.
You can select them as BL33_CFG without U-Boot recompilation.

See TF-A_overview for FIP details.

The file used to debug with gdb is

- u-boot : elf file for U-Boot

For ecosystem release v2.1.0 : u-boot.stm32 : U-Boot binary with STM32 image header, including device tree selected by DEVICE_TREE, loaded by TF-A

This behavior can be restored if you activate CONFIG_STM32MP15x_STM32IMAGE in your defconfig of ecosystem release v3.0.0

This temporary option is only introduced to facilitate the FIP migration but it will be removed in the next EcosystemRelease.

The STM32 image format (*.stm32) is managed by mkimage U-Boot tools and Signing_tool. It is requested by ROM code and TF-A without FIP support (see STM32 header for binary files for details).

6 References

- https://u-boot.readthedocs.io/en/stable/index.html
- https://www.denx.de/wiki/pub/U-Boot/MiniSummitELCE2013/2013-ELCE-U-Boot-Falcon-Boot.pdf
- https://en.wikipedia.org/wiki/Xconfig
- https://github.com/STMicroelectronics/u-boot/releases
- https://github.com/STMicroelectronics/u-boot
- https://source.denx.de/u-boot/u-boot.git or https://github.com/u-boot/u-boot
- https://en.wikipedia.org/wiki/Cross_compiler

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Linux® is a registered trademark of Linus Torvalds.

First Stage Boot Loader

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Second Stage Boot Loader

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Read Only Memory

Central processing unit

Doubledata rate (memory domain)

Flash memories combine high density and cost effectiveness of EPROMs with the electrical erasability of EEPROMs. For this reason, the Flash memory market is one of the most exciting areas of the semiconductor industry today and new applications requiring in system reprogramming, such as cellular telephones, automotive engine management systems, hard disk drives, PC BIOS software for Plug & Play, digital TV, set top boxes, fax and other modems, PC cards and multimedia CD-ROMs, offer the prospect of very high volume demand.

System control and management interface

Microprocessor Unit

Unsorted block images

High-Definition Multimedia Interface (HDMI standard)

Device Firmware Upgrade

Device Tree Binary (or Blob)

Memory Technology Device

Trivial File Transfer Protocol (https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol)

Dynamic Host Configuration Protocol (See https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol for more details)

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

MultimediaCard

SD memory card (https://www.sdcard.org)

Firmware Image Package is a packaging format used by TF-A

Serial Peripheral Interface

Operating System

Flattened uImage Tree is a packaging format used by U-Boot

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Trusted Firmware for Arm® Cortex®-A

Open Portable Trusted Execution Environment

Boot Loader stage 3-3

Boot Loader stage 2