U-Boot overview

Contents

U-Boot overview

Stable: 25.05.2020 - 07:32 / Revision: 25.05.2020 - 07:25

Template:ArticleMainWriter

Template:ArticleApprovedVersion

Contents

1 Das U-Boot

Das U-Boot ("the Universal Boot Loader" or just U-Boot) is an open-source boot loader, which 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: project documentation and https://www.denx.de/wiki/DULG/Manual
- The official **source code** is available with git repository at git.denx.de

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

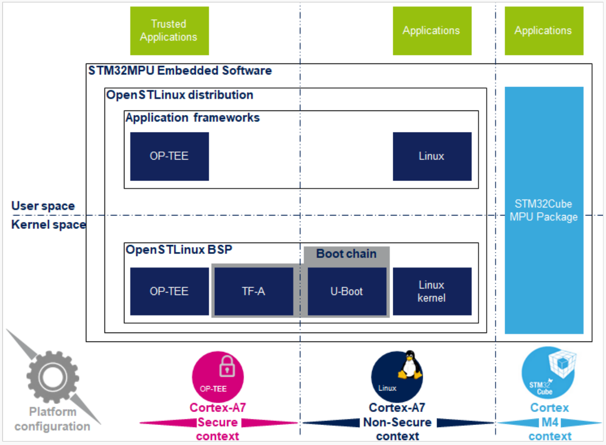
Reading the README file is recommended. It covers the following topics:

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

2 U-Boot overview

The same U-Boot source can generate 2 pieces of firmware used in the STM32 MPU boot chain: SPL and U-Boot

- Trusted boot chain: U-Boot as SSBL
- Basic boot chain: SPL as FSBL and U-Boot as SSBL



Zoom out to STM32MPU Embedded Software

2.1 SPL: FSBL for basic boot

The **U-Boot SPL** or just **SPL** is the first stage boot loader (FSBL) for the basic boot chain.
It is a small binary (bootstrap utility), generated from the U-Boot source, which fits in the internal and limited embedded RAM :

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

2.2 U-Boot: SSBL

U-Boot is the default second stage boot loader (SSBL) for the STM32 MPU platforms for the 2 boot chains, trusted and basic :

- it is configurable and expendable
- it has a simple command line interface (CLI), usually over a serial console port for interaction with the user
- it provides scripting capabilities
- it loads the kernel into RAM and passes control to the kernel
- it manages many internal and external devices like NAND, NOR, Ethernet, USB

- it has many supported features and commands for
  - file systems: FAT, UBI/UBIFS, JFFS
  - IP stack: FTP
  - display: LCD, HDMI, BMP for splashcreen
  - USB: host profile (mass storage) or device profile (DFU stack)

2.3 SPL phases

The **SPL** runs through the main following phases in SYSRAM:

- **board_init_f()**: init drivers up to DDR initialisation (mininimal stack and heap: CONFIG_SPL_STACK_R_MALLOC_SIMPLE_LEN)
- configure heap in DDR (CONFIG_SPL_SYS_MALLOC_F_LEN)
- **board_init_r()**: init other drivers activated in the SPL device tree
- load U-Boot (or Kernel in Falcon mode[1]: README.falcon) and execute it

2.4 U-Boot phases

U-Boot runs through the following main phases in DDR:

- **Pre-relocation** initialization (common/board_f.c): minimal init (cpu, clock, reset, ddr, console,...) running at the load address CONFIG_SYS_TEXT_BASE
- **Relocation**: copy the code to the end of DDR
- **Post-relocation initialization**:(common/board_r.c): init all the drivers
- **Execution of commands**: through autoboot (CONFIG_AUTOBOOT) or console shell
  - execute the boot command (bootcmd=CONFIG_BOOTCOMMAND by default):
    for example, execute the command 'bootm' to:
    - load and check images (kernel, device tree, ramdisk....)
    - fixup kernel device tree
    - install secure monitor (optional)
    - pass control to the Linux kernel (or other 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** defines in include/configs/stm32mp*.h
  the file name is configured by CONFIG_SYS_CONFIG_NAME
  (these flags are progressively migrating to Kconfig)
  for stm32mp157: we use include/configs/stm32mp1.h file
- **DeviceTree** = U-Boot and SPL binaries include a device tree blob which is parsed at run time

All the configuration flags (CONFIG_) are described in the source code: the README file or documentation directory
example: CONFIG_SPL => activate the SPL compilation.
Hence to compile U-Boot, you need to select the <target> and the device tree for the board to select a predefined configuration.
See #U-Boot_build for examples.

3.1 Kbuild

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

- select pre-defined configuration (defconfig file, in configs directory) and generate the first **.config**

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

- change U-Boot compile configuration (modify .config) using one of the 5 make command

```
PC $> make menuconfig --> menu based program
PC $> make config  --> line-oriented configuration
PC $> make xconfig --> QT program[2]
PC $> make gconfig --> GTK program
PC $> make nconfig --> ncurse menu based program
```

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

Warning: modification is only done locally in the build directory, it is lost after a "make distclean"

So if you want to use your configuration as defconfig:

```
PC $> make savedefconfig
```

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

The other makefile targets are :

```
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

See doc/README.fdt-control

The board device tree, with the same binding as the kernel, is integrated with the SPL and U-Boot binaries:

- appended at the end of the code by default (CONFIG_OF_SEPARATE)
- embedded in binary (CONFIG_OF_EMBED): useful for debug, allows easy elf file loading

A default device tree is defined in the defconfig file (with CONFIG_DEFAULT_DEVICE_TREE).

You can also select another supported device tree with the make flag DEVICE_TREE
for stm32mp32 boards the file are: arch/arm/dts/stm32mp*.dts

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

or you can provide a precompiled device tree blob (with EXT_DTB option)

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

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

To do this, U-Boot uses some specific device-tree flags to specify 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-spl** => present in SPL

In the device tree used by U-Boot, these flags **need to be added in each node** used in SPL or in U-Boot before relocation but also for each used handle (clock, reset, pincontrol).

4 U-Boot command line interface (CLI)

see U-Boot Command Line Interface

If CONFIG_AUTOBOOT is activated, to enter in this console, you have CONFIG_BOOTDELAY seconds (2s by default) before bootcmd execution (CONFIG_BOOTCOMMAND) by pressing any key when the line below is displayed.

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

4.1 Commands

The commands are defined cmd/*.c , they are activated under associated configuration flag **CONFIG_CMD_***.

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

List of commands extracted from U-Boot Manual (**not-exhaustive**):

- Information Commands
  - bdinfo - print Board Info structure
  - coninfo - print console devices and informations
  - flinfo - print FLASH memory information
  - iminfo - print header information for application image
  - help - print online help
- Memory Commands
  - base - print or set 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 - print FLASH memory information
  - erase - erase FLASH memory
  - protect - enable or disable FLASH write protection
  - mtdparts - define a Linux compatible MTD partition scheme
- Execution Control Commands
  - source - run script from memory
  - bootm - boot application image from memory
  - go - start application at address 'addr'
- Download Commands
  - bootp - boot image via network using BOOTP/TFTP protocol
  - dhcp - invoke DHCP client to obtain IP/boot params
  - loadb - load binary file over serial line (kermit mode)
  - loads - load S-Record file over serial line
  - rarpboot- boot image via network using RARP/TFTP protocol
  - tftpboot- boot image via network using TFTP protocol
- Environment Variables Commands
  - printenv- print environment variables
  - saveenv - save environment variables to persistent storage
  - setenv - set environment variables
  - run - run commands in an environment variable
  - bootd - boot default, i.e., run 'bootcmd'

- Flattened Device Tree support
    - fdt addr - select FDT to work on
    - fdt list - print one level
    - fdt print - recursive print
    - fdt mknode - create new nodes
    - fdt set - set node properties
    - fdt rm - remove nodes or properties
    - fdt move - move FDT blob to new address
    - fdt chosen - fixup dynamic info
- Special Commands
    - i2c - I2C sub-system
- Storage devices
- Miscellaneous Commands
    - echo - echo args to console
    - reset - Perform RESET of the CPU
    - sleep - delay execution for some time
    - version - print monitor version

To add a new command, see doc/README.commands

4.2 U-Boot environment variables

The U-Boot behavior is configured with environment variables.

see Manual and README / Environment Variables

By default the env is NOT saved (CONFIG_ENV_IS_NOWHERE), only the default environment is used (saveenv command is not working)

You can modify this default environment by changing the content of CONFIG_EXTRA_ENV_SETTINGS in your configuration file (for example ./include/configs/stm32mp1.h) (see README / - Default Environment).

You can also choose one location with configuration flags:

- CONFIG_ENV_IS_IN_MMC
- CONFIG_ENV_IS_IN_FLASH
- CONFIG_ENV_IS_IN_SPI
- CONFIG_ENV_IS_IN_FAT
- CONFIG_ ENV_IS_IN_NAND
- CONFIG_ENV_IS_IN_UBI
- CONFIG_ENV_IS_IN_EEPROM

4.2.1 bootcmd

Autoboot command: defines the command executed when U-Boot starts (CONFIG_BOOTCOMMAND).

But you can change this variable in CONFIG_EXTRA_ENV_SETTINGS (after BOOTENV macro needed for #Generic Distro configuration).

```
#define CONFIG_EXTRA_ENV_SETTINGS \
        "stdin=serial\0" \
        "stdout=serial\0" \
```

```
        "stderr=serial\0" \
        "kernel_addr_r=0xc2000000\0" \
        "fdt_addr_r=0xc4000000\0" \
        "scriptaddr=0xc4100000\0" \
        "pxefile_addr_r=0xc4200000\0" \
        "splashimage=0xc4300000\0"  \
        "ramdisk_addr_r=0xc4400000\0" \
        "fdt_high=0xffffffff\0" \
        "initrd_high=0xffffffff\0" \
        BOOTENV \
        "bootcmd=run bootcmd_mmc0\0"
```

4.3 Generic Distro configuration

see doc/README.distro

This feature is activated for ST boards (CONFIG_DISTRO_DEFAULTS):

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

With DISTRO the default command executed: include/config_distro_bootcmd.h

```
bootcmd=run distro_bootcmd
```

This script will try any device found in the variable 'boot_targets' and execute the associated bootcmd.

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

```
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 searchs for a configuration file **extlinux.conf** in a bootable device, this file defines the kernel configuration to use:

- bootargs
- kernel + device tree + ramdisk files (optional)
- FIT image

4.4 U-Boot scripting capabilities

"Script files" are command sequences that will be executed by U-Boot's command interpreter; this feature is especially useful when you configure U-Boot to use a real shell (hush) as command interpreter.

See U-Boot script manual for example.

5 U-Boot build

5.1 Prerequisites

You need:

- a PC with Linux and tools:
  - see PC_prerequisites
  - #ARM cross compiler
- U-Boot source code
  - the latest STMicroelectonics U-Boot version
    - tar.xz file from Developer Package (for example STM32MP1)
    - from GITHUB[3], with git command

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

- from the Mainline U-Boot in official GIT repository [4]

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

5.1.1 ARM cross compiler

You need to have a cross compiler [5] installed on your Host (X86_64, i686, ...) for the targeted Device architecture = ARM, the environment variables ($PATH and $CROSS_COMPILE) need to be configured in your shell.

You can use gcc for ARM, available in:

1. the SDK toolchain
   See Cross-compile with OpenSTLinux SDK, PATH and CROSS_COMPILE are automatically updated.
2. an existing package (for example, on Ubuntu/Debian: (**PC $>** sudo apt-get install gcc-arm-linux-gnueabihf)
3. an existing toolchain:
   - gcc v8 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: **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 it in $HOME,
and you need to update your environment:

```
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-
```

5.2 Compilation

In the U-Boot source directory, you need to select the <target> and the <device tree> for your configuration and then execute the "make all" command.

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

KBUILD_OUTPUT can be used optionally to change the output directory if you want to compile several targets or don't compile in the source directory, for example:

```
PC $> export KBUILD_OUTPUT=../build/basic
```

DEVICE_TREE can be also exported to your environment when you support only one board, for example:

```
PC $> export DEVICE_TREE=stm32mp157c-ev1
```

For all the stm32mp15 family, we manage 3 configurations:

- stm32mp15_trusted_defconfig: trusted boot chain, U-Boot (without SPL) is unsecure and uses Secure monitor from TF-A
- stm32mp15_optee_defconfig: trusted boot chain, U-Boot (without SPL) is unsecure and uses Secure monitor from SecureOS = OP-TEE
- stm32mp15_basic_defconfig: basic boot chain, with an SPL as FSBL, U-BOOT is secure and installs monitor with PSCI

The board diversity is only managed with the device tree.

Examples from STM32MP15 U-Boot:

```
PC $> export KBUILD_OUTPUT=../build/basic
PC $> make stm32mp15_basic_defconfig
PC $> make DEVICE_TREE=stm32mp157c-<board> all
```

```
PC $> export KBUILD_OUTPUT=../build/trusted
PC $> make stm32mp15_trusted_defconfig
PC $> make DEVICE_TREE=stm32mp157c-<board> all
```

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

Use help to list other targets:

```
PC $> make help
```

5.3 Output files

The resulting U-Boot files are present in your build directory (U-Boot or KBUILD_OUTPUT) and SPL Images are in the spl subdirectory.

STM32 image format (*.stm32) is managed by mkimage U-Boot tools and is requested by boot ROM (for basic boot chain) or by TF-A (for trusted boot chain).

- **u-boot.stm32** : U-Boot binary with STM32 image header => SSBL for Trusted boot chain
- **u-boot.img** : U-Boot binary with uImage header => SSBL for Basic boot chain
- u-boot : elf file, used to debug with gdb
- **spl/u-boot-spl.stm32** : SPL binary with STM32 image header => FSBL for Basic boot chain
- spl/u-boot-spl : elf file, used to debug with gdb

6 References

- 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
- http://git.denx.de/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)

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

Second Stage Boot Loader

First 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 forvolatile random-accesssemiconductor memories.)

Read Only Memory

Central processing unit

Doubledata rate (memory domain)

Microprocessor Unit

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)

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

MultimediaCard

Serial Peripheral Interface

Electrically-erasable programmable read-only memory

Initial ramdisk (https://en.wikipedia.org/wiki/Initial_ramdisk)

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

Trusted Firmware for Arm Cortex-A

Power State Coordination Interface

Permission error

Stable: 25.06.2020 - 07:35 / Revision: 19.06.2020 - 07:24

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 22.04.2020 - 13:12 / Revision: 22.04.2020 - 13:10

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 04.02.2020 - 07:47 / Revision: 04.02.2020 - 07:34

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 29.06.2020 - 09:25 / Revision: 23.06.2020 - 09:27

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 23.07.2020 - 14:54 / Revision: 07.07.2020 - 12:09

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 16.01.2020 - 14:27 / Revision: 16.01.2020 - 14:23

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 13.05.2020 - 08:56 / Revision: 13.05.2020 - 08:54

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

Permission error

Stable: 25.05.2020 - 07:31 / Revision: 25.05.2020 - 07:29

You do not have permission to read this page, for the following reason:

The action "Read pages" for the draft version of this page is only available for the groups ST_editors, ST_readers, Selected_editors, sysop, reviewer

