U-Boot Bootloader

***Boot Loader** : Microprocessors can execute only code that exists in memory (either ROM or RAM), while operating systems normally reside in large-capacity devices such as hard disks, CD-ROMs, USB disks, network servers, and other permanent storage media.*

When the processor is powered on, the memory doesn't hold an operating system, so special software is needed to bring the OS into memory from the media on which it resides. This software is normally a small piece of code called the boot loader. On a desktop PC, the boot loader resides on the master boot record (MBR) of the hard drive and is executed after the PC's basic input output system (BIOS) performs system initialization tasks. In an embedded system, the boot loader's role is more complicated because these systems rarely have a BIOS to perform initial system configuration. Although the low-level initialization of the microprocessor, memory controllers, and other board-specific hardware varies from board to board and CPU to CPU, it must be performed before an OS can execute.

1. *At a minimum, a boot loader for an embedded system performs these functions:*

- *Initializing the hardware, especially the memory controller*
- *Providing boot parameters for the OS*
- *Starting the OS*

2. *Most boot loaders provide features that simplify developing and updating firmware; for example:*

- *Reading and writing arbitrary memory locations*
- *Uploading new binary images to the board's RAM from mass storage devices*
- *Copying binary images from RAM into flash*

U-Boot** is a like a mini-OS by itself - it has a console, some commands, allows you break the boot process and e.g. modify the kernel command line arguments or even load the kernel from a different location (SD/MMC or USB), run some tests and so on.it is also a **Hybrid firmware bootloader.

Intro:

U-Boot is the most popular bootloader in the linux based embedded devices.it is released as open source under the GNU GPLv2 license.It supports a wide nrange of microprocessors like MIPS, ARM,PPC,Blackfin,AVR32 and X86.It even supports FPGA base nios platforms.if your hardware device is based out any of these processors and if you are looking for a bootloader the best bet is to try u-boot first.It also support different methods of booting which is pretty much needed on fallback situations.The list of filesystems supports by u-boot is more. It has support to boot from USB, NAND, NOR,SD Card.It also had the support to load the linux kernel from the network from the TFTP .
Last but not least it also has the command line interface which give you very easy access to it and try many things before finalizing your design.You can configure U-BOOT for various boot methods like MMC, USB,NFC or NAND based and it allows you to test physical RAM of any issuses.

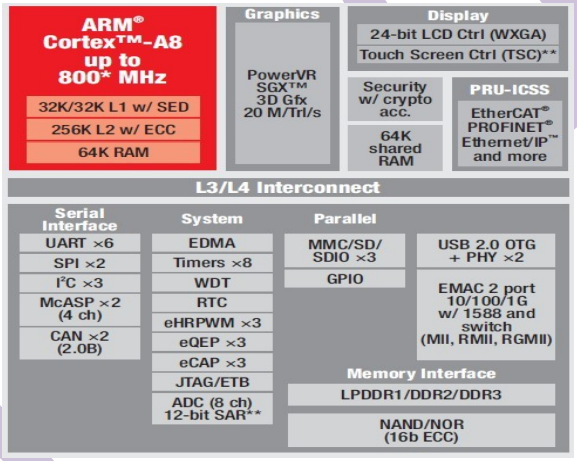
U-Boot Bootloader

Stages in Bootloader :

For Starters U-Boot is First stage and Second stage bootloaders.When U-Boot is compiled we get two images first stage (MLO) and second stage (u-boot.img) images. It is loaded from the system's ROM code from a supported boot device. The ROM code checks for the various boot device that is available.And starts execution of the device which is capable of booting. This can be controlled through jumpers,though some resistor based methods are also exists.since each platform is different.

Stage 1 Bootloader is sometime called Small SPL(Secondary Program Loader).SPL would do inital hardware configuration and loads rest of U-boot.Regard less of whether the SPL is used or not U-Boot will do both the First-stage and secondary-stage booting.

In First stage U-boot initlizates memory controller and SDRAM.This is needed as rest of the execution of the code depends on this. Depending upon the list of the devices it supported by the platform it initlizes the rest. For example if your is supported by the USB to boot and their is no network connectivity then U-Boot will do exactly the same.
If you want to use linux kernel,then setting up of memory controller is only mandatory thing expected by linux kernel. If memory controller is not initilzed properly then linux kernel wont be able to boot.



Block Diagram of Target

- *Universal Bootloader shortly called as U-Boot.*
- *Open source ,primary bootloader use din embedded devices.*
- *Initialize and then boots in linux Kernel.*
- *Featured with image extraction ,handles the compresed images*
- *Featured to provide arguments,device tree info to Linux Kernel*
- *Contains First-stage and Second stage bootloader.*
- *First-Stage : Initlizes the DDRAM.*
- *Second Stage : Loades the OS from the memory devices into kernel.*

U-Boot Bootloader

U-Boot Phases

U-Boot boots through the following phases:

TPL

Very early init, as tiny as possible. This loads SPL (or VPL if enabled).

VPL

Optional verification step, which can select one of several SPL binaries, if A/B verified boot is enabled. Implementation of the VPL logic is work-in-progress. For now it just boots into SPL.

SPL

Secondary program loader. Sets up SDRAM and loads U-Boot proper. It may also load other firmware components.

U-Boot

U-Boot is both a first-stage and second-stage bootloader. It is loaded by the system's ROM (e.g. onchip ROM of the ARM CPU) from a supported boot device, such as an SD card, SATA drive, NOR flash or NAND flash. If there are size constraints, U-Boot may be split into two stages: the platform would load a small SPL (Secondary Program Loader), which is a stripped-down version of U-Boot, and the SPL would do some initial hardware configuration (e.g.DRAM initialization using CPU cache as RAM) and load the larger, fully featured version of U-Boot.Regardless of whether the SPL is used, U-Boot performs both first-stage (e.g., configuring memory controllers and SDRAM) and second-stage booting (performing multiple steps to load a modern operating system from a variety of devices that must be configured, presenting a menu for users to interact with and control the boot process, etc.).

U-Boot implements a subset of the UEFI specification as defined in the Embedded Base Boot Requirements (EBBR) specification.UEFI binaries like GRUB or the Linux kernel can be booted via the boot manager or from the command-line interface.

U-Boot runs a command line interface on a console or a serial port. Using the CLI, users can load and boot a kernel, possibly changing parameters from the default. There are also commands to read device information, read and write flash memory, download files (kernels, boot images, etc.) from the serial port or network, manipulate device, and work with environment variables (which can be written to persistent storage, and are used to control U-Boot behavior such as the default boot command and timeout before auto-booting, as well as hardware data such as the Ethernet MAC address).

Unlike PC bootloaders which obscure or automatically choose the memory locations of the kernel and other boot data, U-Boot requires its boot commands to explicitly specify the physical memory addresses as destinations for copying data (kernel, ramdisk, device tree, etc.) and for jumping to the kernel and as arguments for the kernel. Because U-Boot's commands are fairly low-level, it takes several steps to boot a kernel, but this also makes U-Boot more flexible than other bootloaders, since the same commands can be used for more general tasks. It's even possible to upgrade U-Boot using U-Boot, simply by reading the new bootloader from somewhere (local storage, or from the

serial port or network) into memory, and writing that data to persistent storage where the bootloader belongs.

U-Boot has support for USB, so it can use a USB keyboard to operate the console (in addition to input from the serial port), and it can access and boot from USB Mass Storage devices such as SD card readers.

Compatible file systems

U-Boot does not need to be able to read a filesystem in order for the kernel to use it as a root filesystem or initial ramdisk; U-Boot simply provides an appropriate parameter to the kernel, and/or copies the data to memory without understanding its contents.

However, U-Boot can also read from (and in some cases, write to) filesystems. This way, rather than requiring the data that U-Boot will load to be stored at a fixed location on the storage device, U-Boot can read the filesystem to search for and load the kernel, device tree, etc., by pathname.

U-Boot includes support for these filesystems:

- *btrfs*
- *CBFS (corebootfile system)*
- *Cramfs*
- *ext2*
- *ext3*
- *ext4*
- *FAT*
- *FDOS*
- *JJFS2*
- *ReiserFS*
- *Squashfs*
- *UBIFS*
- *ZFS*

Device tree

Device tree is a data structure for describing hardware layout. Using Device tree, a vendor might be able to use a less modified mainline U-Boot on otherwise special purpose hardware. As also adopted by the Linux kernel, Device tree is intended to ameliorate the situation in the embedded industry, where a vast number of product specific forks (of U-Boot and Linux) exist. The ability to run mainline software practically gives customers indemnity against lack of vendor updates.

Types of device trees are ***dts,dtc,dtb,dtsi,EDT,FDT****.*

U-Boot Bootloader

The Directories present in u-boot are

- ***api*** *: Network,display and storage related APIs.*
- ***Arch*** *: CPU Architecture specific files.*
- ***Board*** *: Hardware Board specific files.*
- ***Common*** *: CPU/Board independent files like Uboot commands.*
- ***Config*** *: Hardware board configuration files.*
- ***Disk*** *: Disk drive relater files handles the disk partition.*
- ***Doc*** *: Bunch of READMEof CPU,SOC,Periphereal documentation.*
- ***Drivers*** *:supported device driver files.*
- ***Dts*** *: control related with DTS/open firmware specification.*
- ***Examples*** *: some examples using U-Boot loader APIs.*
- ***Fs*** *: file system related files.*
- ***Include*** *: U-Boot related common header files.*
- ***Lib*** *: CPU/Board independent library files.*
- ***Licenes*** *: Contain U-Boot loader license information.*
- ***Net*** *: Netwaork supported driver and software stack files.*
- ***Scripts*** *: General scripts to accommodate sequence commands.*
- ***Spl*** *: first stage bootloader*
- ***test*** *: Some of the test utility files.*
- ***Tools*** *: host tools nedded to bulid images,etc.*

Boot Flow

Boot ROM -> First stage bootloader -> Initalizes memory and loading -> Second Stage Bootloader

ROM code - > SPL - > U-Boot - > Kernel.

U-Boot Commands

U-Boot has a set of built-in commands for booting the system, managing memory, and updating an embedded system's firmware. By modifying U-Boot source code, you can create your own built-in commands.

U-Boot has different types of commands based on their usage they are categoried into

- ***Information commands*** *: used for the information of the related hardware ,software and others.*
- ***Environment commands :*** *To read, write, and save environment variables .*
- ***MII commands :*** *To access the Ethernet PHY.*
- ***I2C Commands :*** *These commads are used for interfacing with I2C interface.*
- ***Serial Port Commands :*** *Theses Commands Works with sesrial lines.*
- ***Network Commands :*** *these command**Tools** s are used for the network related.*
- ***USB Commands :*** *these commands are used for the usb subsystem controls.*
- ***Memory Commands :*** *memory commands manages the RAM memory.*

U-Boot Bootloader

Environment variables:

U-Boot uses environment variables to tailor its operation. The environment variables configure settings such as the baud rate of the serial connection, the seconds to wait before auto boot, the default boot command, and so on.These variables must be stored in either non-volatile memory (NVRAM) such as an EEPROM or a protected flash partition.
The factory default variables and their values also are stored in the U-Boot binary image itself. In this way, you can recover the variables and their values at any time with the envreset command. Environment variables are stored as strings (case sensitive). Custom variables can be created as long as there is enough space in the NVRAM.

Running process of the U-boot

The bootloader used for Embedded Artists COM boards is U-boot, also known as Universal Boot Loader or Das U-Boot. This is an open-source bootloader commonly used on many different architectures and platforms.
U-boot's main responsibility is to load the Linux kernel, select and load the device tree and hand it over the device tree to the kernel. In order to do this the U-boot has to do some initial hardware initialization such as basic processor (CPU) setup, initialize clocks and timers, initialize console, optionally the display and board specific initialization. ighlights some of the functions part of the initialization flow

_main	*arm/lib/crt0.S* *arm/lib/crt0_64.S*	*Main function called by C runtime.*
board_init_f	*common/board_f.c*	*Prepares the hardware for execution. Will for example call arch_cpu_init to initialize CPU, but also the board specific function board_early_init_f.*
board_init_r	*common/board_r.c*	*SDRAM is initialized and global variables are available when this function is called. SDRAM is initialized. It will call functions such as board_init, initr_mmc,console_init_r and finally run_main_loop.*
main_loop	*common/main.c*	*It is in this function U-boot will start to process commands, such as the commands defined in the U-boot environment. For auto booting U-boot will run the command(s) defined in the configuration variable CONFIG_BOOTCOMMAND.*