U-boot Porting guide

Saurin Suthar

eInfochips

U-BOOT PORTING GUIDE

❖ U-BOOT OVERVIEW

u-boot(Universal Bootloader) is an open source, multi platform bootloader. u-boot supports interactive commands, environment variables, command scripting and booting from external media. u-boot supports a large variety of popular CPUs and CPU families used today, and a much larger collection of reference boards based on these processors.

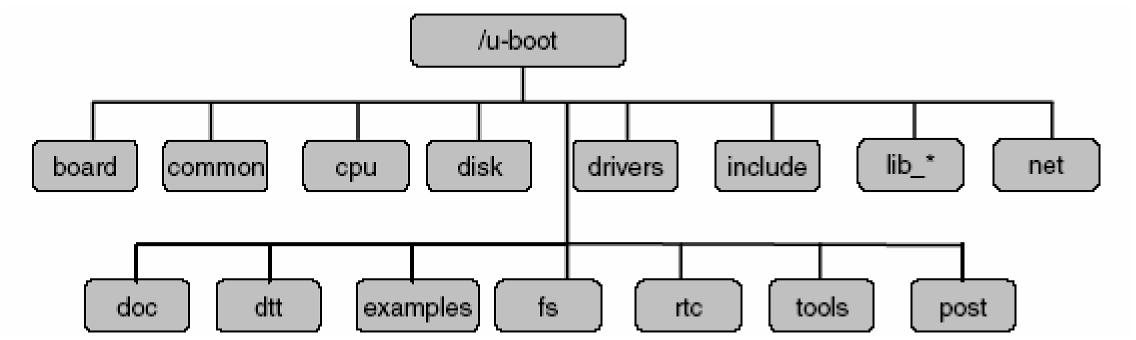
u-boot configures different hardware blocks on the board and brings them out of reset into a sane state. It can load and start an OS automatically (auto-boot) or, alternatively, it allows the user to run commands to start the OS. The subset of default commands that are part of u-boot also provide capability to the user to perform memory, network, flash operations, and more prior to OS boot-up.

Generally, u-boot resides in the beginning area of the flash. The exact sector or block is defined by the processor. u-boot initialize the CPU and several peripherals located on board, create some critical data structures which will be used by kernel, and load itself into the top of the memory.

When control is transferred to u-boot, it will initialize the interrupts and the rest peripherals. Then it will wait for commands from user. If u-boot receives the command to boot the kernel image or if it is used to boot the kernel Image directly, it will uncompress the kernel, load it to the memory, and give the control to the kernel. Then kernel will continue execution without the intervention of u-boot.

u-boot provides additional functions than just booting device and initializing kernel. Usually it comes with a command-line interface

❖ DIRECTORY STRUCTURE



Directory Name	**Description**
board	Platform, board-level files; i.e. davinci, Sandpoint, cds, Marvell. The board directory contains all the specific board initialization functions, which are called from lib_<arch>/board.c.
board/<boardname>/	Specific board info for specifice board; i.e. serial, asm_init.S config.mk, flash.c etc.
common	All the commands; i.e. cmd_boot.c, cmd_date.c, environment, env.c, main.c, etc.
cpu	CPU specific information; i.e. 74xx_7xx, c824xcpu/74xx_7xx contains cpu.c, cpu_init.c, cache.S, interrupts.c, start.S, traps.S, etc
disk	Partition and device information for disks
drivers	Various device drivers; i.e. ns9750_eth, rt18019, serial, usbtty, etc.
include	Various header files; i.e. console.h, version.h, usb.h, pci.h. version.h has the version display header #define u-boot_VERSION "xxx" include/configs contains the configurations for all the various boards
lib_generic	Generic libraries; i.e. bzlib.c vsprintf, string.c, etc.
lib_*	processor specific libraries; i.e. bat_rw.c, board.c (memory sizes, baud rates, and calls board specific routines in /u-boot/board)
net	Ethernet files; i.e. eth.c, net.[ch], nfs.[ch], bootp.c [ch], etc.
doc	Various readme
dtt	Temperature sensor code
examples	Some stand-alone example code; i.e. hello_world.c, etc.
fs	File system directories and code; i.e. fat, fdos, jffs2
rtc	Real time clock code; i.e. date.c, mpc8xx.c, etc.
tools	Various tool directories and files; i.e. env, gdb, logos, scripts, mkimage.c, etc
post	Directories and files; i.e. post.c codec.c, cache.c, memory.c, uart.c, etc.
board	Platform, board-level files; i.e. davinci, Sandpoint, cds, Marvell. The board directory contains all the specific board initialization functions, which are called from lib_<arch>/board.c.
board/<boardname> /	Specific board info for specifice board; i.e. serial, asm_init.S config.mk, flash.c etc.
common	All the commands; i.e. cmd_boot.c, cmd_date.c, environment, env.c, main.c, etc.
cpu	CPU specific information; i.e. 74xx_7xx, mpc824xcpu/74xx_7xx contains cpu.c, cpu_init.c, cache.S, interrupts.c, start.S, traps.S, etc
disk	Partition and device information for disks
drivers	Various device drivers; i.e. ns9750_eth, rt18019, serial, usbtty, etc.
include	Various header files; i.e. console.h, version.h, usb.h, pci.h. version.h has the version display header #define u-boot_VERSION "xxx" include/configs contains the configurations for all the various boards
lib_generic	Generic libraries; i.e. bzlib.c vsprintf, string.c, etc.

lib_*	processor specific libraries; i.e. bat_rw.c, board.c (memory sizes, baud rates, and calls board specific routines in /u-boot/board)
net	Ethernet files; i.e. eth.c, net.[ch], nfs.[ch], bootp.c [ch], etc.
doc	Various readme
dtt	Temperature sensor code
examples	Some stand-alone example code; i.e. hello_world.c, etc.
fs	File system directories and code; i.e. fat, fdos, jffs2
rtc	Real time clock code; i.e. date.c, mpc8xx.c, etc.
tools	Various tool directories and files; i.e. env, gdb, logos, scripts, mkimage.c, etc
post	Directories and files; i.e. post.c codec.c, cache.c, memory.c, uart.c, etc.

❖ PORTING U-BOOT

The major changes required in u-boot for porting to any board, changes are related to a new processor and other peripheral devices. A common practice is to use an existing configuration for some similar board and create a new configuration for the new board. The NOR flash, complies with the Common Flash Interface (CFI) standards and flash operations can, therefore, use the existing driver by adding flash specific definitions.

When designing a new system with the processor, follow these steps:

a. Define a physical memory map with the sections like flash memory, dram memory and peripheral memory.

b. Define a memory link map with the following sections
   - .text – Program executable and data sections of u-boot
   - .env – Reserve space for environmental variables
   - Global data – This data is reserved for global variables
   - Monitor – Reserved space in ram for the u-boot bootloader
   - Malloc – Reserve space for dynamically allocated memory
   - Bootmap – Reserve space for operating system image

c. Configure u-boot by modifying the <boardname>.h header file

First, create a new set of directories and files to mimic the existing configuration. The necessary directories and files are:

- u-boot/board/<boardname>
- u-boot/include/configs/<boardname>.h
- u-boot/Makefile

Then copy files, and make changes to Makefile and platform specific files to retarget the code for board. **include, cpu, lib_arch, and board** are the directories dependent upon a particular architecture and the board. All other directories remain the same for any architecture. For porting u-boot for specific processor and board, we need to add or modify the following files:

- **The include directory:**
  The include directory contains all header files globally used by u-boot. In this directory we need to generate or change following files for a specific cpu and board:
  1) **Configs/<boardname>.h**
     This file includes hardware configuration for memory map and peripherals. It contains the chip configuration, flash boot configuration, nor and NAND flash configuration, SDRAM memory configuration, serial configuration, u-boot configuration, I2C configuration, Linux configuration, Network and Ethernet configuration etc.

  2) **<core>.h** for example **arm926ejs.h**
     This file contains the processor specific definitions required for u-boot to access the CPU. This includes the register definitions.

- **The cpu directory:**
  The cpu directory contains initialization routines for internal peripherals.

  1) **<core>/cpu.c**
     This file contains the cpu specific code. It performs operations like read and write to control register, reset cpu, I/D cache enable/disable, Setup the stacks for IRQ and FIQ etc.

  2) **<core>/interrupt.c**
     This file contains the function related to interrupt and timer. Like enable interrupts, disable interrupts, timer related operations etc.

  3) **<core>/start.S**
     This file contains startup code for ARM926EJS cpu core.

- **The lib_<arch> directory :**

  4) **board.c**
     This file performs the following functions:
     - memory map initialization
       - Logical memory map configuration
       - dynamic memory allocation routines(malloc)
     - u-boot peripheral initialization

  5) **<arch>linux.c for example armlinux.c**
     This file contains Implementation of bootm command, that is used to places the kernel image into RAM. This command passes the architecture number and boot parameters to the kernel and starts the kernel execution by giving a control to kernel image.

  6) **div0.c**
     This file contains division-by zero exception handler

- **The board directory:**
  An individual directory for each board configuration supported by u-boot is found in the board directory. This sub-directory includes board specific configuration routines.

7) **< boardname >/flash.c**
   This file contains the function related to flash memory like flash Initialization, Reset flash, print flash information, Erase flash, Write to flash

8) **< boardname >/ < boardname >_emac.c**
   This file initializes EMAC and performs EMAC related operations like receive packet and send packet.

9) **< boardname >/ < boardname >.c**
   This file contains functions like board initialization routine, Initialize Timer configuration register, Serial Port initialization, NAND flash initialization etc.

10) **< boardname >/ soc.h**
   This file contains peripheral base addresses, Interrupt event ids etc.

11) **< boardname >/ platform.S**
   This file contains the PLL, power domain initialization, SDRAM Memory configuration register like SDRAM control register, refresh rate control register, initialization or SDRAM etc.

Modify the top level makefile to specify the new configuration. All the configurations are listed under various titles.

```
<boardname>_config :    unconfig
        @./mkconfig $(@:_config=) <architecture> <core> <boardname>
```

e.g, For davinci EVM board,

```
davinci_config :    unconfig
        @./mkconfig $(@:_config=) arm arm926ejs davinci
```

❖ U-BOOT BUILD

Toolchain

Before building and installing u-boot you need a cross-development toolchain for your target architecture. You are probably familiar with the toolchain on your desktop Linux PC. That toolchain runs on an x86 platform and generates binaries for an x86 platform. A cross-development toolchain executes particular CPU architecture, but generates binaries for a different architecture.

U-boot Build procedure

The following command sequence is used to build the u-boot for the particular board.

```
$ make distclean
$ make <boardname>_config
$ make
```

The result of these command sequence will build the following targets and there should be no compilation errors.

1. **u-boot** is an ELF file of the code and it can also used to generate the disassembly.
   ```
   $ objdump -D u-boot > u-boot.dis
   ```
2. **u-boot.bin** is the binary image. (with ELF headers striped out) used for download and debugging
3. **u-boot.srec** is the S-Record image.
4. **u-boot.map** has the global addresses for symbols, etc.

Note: Header information will be added to the image for the bootROM to decide the data transfer parameters.