Application Note

Introduction

Das U-Boot is a Free Software bootloader available for a wide range of embedded processor architectures. A bootloader is a piece of software that runs when the hardware turns on and starts the operating system. U-Boot is not only capable of boot-strapping a system but is also used as a monitor program. **Some information such as memory map in this document is specific to 2.0 release.**

Contents

1. Hardware Support

The Ubicom port of U-Boot currently supports Ethernet, Switch, SPI, GPIO, Flash, UART (both serial and mailbox console) and reset.

2. Software Support

The following is a list of currently supported commands, which can be executed from U-Boot command line. You can see the list of supported commands by executing the **help** command within the U-Boot command line interface.

```
?       - alias for 'help'
askenv  - get environment variables from stdin
autoscr - DEPRECATED - use "source" command instead
base    - print or set address offset
bcm539x - configure and use the bcm539x switch
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
chpart  - change active partition
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
devtree - print ubicom device tree contents
dhcp    - boot image via network using DHCP/TFTP protocol
echo    - echo args to console
erase   - erase FLASH memory
flinfo  - print FLASH memory information
go      - start application at address 'addr'
gpio    - configure and use gpio pins
help    - print online help
iminfo  - print header information for application image
imls    - list all images found in flash
imxtract- extract a part of a multi-image
itest   - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
md      - memory display
mm      - memory modify (auto-incrementing)
mtdparts- define flash/nand partitions
mtest   - simple RAM test
mw      - memory write (fill)
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
```

```
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
source  - run script from memory
spier   - SPI-ER NAND sub-system
sspi    - SPI utility commands
tftpboot- boot image via network using TFTP protocol
ubi     - ubi commands
ubifsload- load file from an UBIFS filesystem
ubifsls - list files in a directory
ubifsmount- mount UBIFS volume
version - print monitor version
```

You can also get more help on a specific command by executing **help *command*** as follows:

```
Ubicom =>help printenv
printenv
    - print values of all environment variables
printenv name ...
    - print value of environment variable 'name'
```

Most of the commands available within U-Boot command line, except **devtree, gpio** and a few others, are common to all platforms to which U-Boot has been ported. Please have a look at Das U-Boot Manual for more information on the common commands. The command **devtree** displays device tree information which is passed to Linux launched by U-Boot. Below is an example of the output of the **devtree** command:

```
Ubicom =>devtree
Device Tree:
  bffffc8: sendirq=255, recvirq=255, name=board
  bffffd84: sendirq=255, recvirq=255, name=bootargs
  bffffca4: sendirq=255, recvirq=073, name=eth_lan
  bffffc5c: sendirq=255, recvirq=075, name=eth_wan
  bffffc10: sendirq=255, recvirq=093, name=pciej
  bffffb98: sendirq=255, recvirq=255, name=processor
  bffffcec: sendirq=255, recvirq=255, name=storage
  bffffd3c: sendirq=255, recvirq=028, name=traps
```

The command **gpio** allows you to experiment with the general purposes I/O pins of your board. Be careful with this command, because it is possible to permanently damage the board. There are no status LEDs in IP8K RGW boards that are connected with GPIO pins in u-boot to illustrate a safe usage. The example below show a hypothetical usage of this command where pin xx is set as an output bin and then enabled followed by toggling the pins.

```
Ubicom =>help gpio
gpio <pin-nr> [<cmd>] - read the input from <pin-nr> or issue a command
<pin-nr> - 0+pin for Port A pins, 32+pin for port B pins, 64+pin for port
  C...
           Example: pin-nr 98 is Port D, Pin 2 (same as in Linux)
<cmd>    - enable/disable, input/output, toggle/high/low (be careful!!!)

Ubicom =>gpio xx output
Ubicom =>gpio xx enable
Ubicom =>gpio xx toggle
Ubicom =>gpio xx toggle
```

3. Environment Variables

Most of the U-Boot commands make use of environment variables to provide ease of use. The Ubicom port of U-Boot comes with several default environment variables to enable easy development on the Ubicom platform.

3.1. Displaying Environment Variables

Environment variables can be easily displayed either individually or in a list by using the command **printenv**. Below is the output of both usages of the command:

```
Ubicom =>printenv
bootcmd=bootm
bootdelay=2
baudrate=115200
ethaddr=02:03:64:de:fa:dd
serverip=192.168.0.1
bootfile=vmlinux.ub
bootargs=mtdparts=ubicom32fc.0:512k(bootloader)ro,0(kernel),14080k(rootfs),15
  36k(jffs2),256k(fw_env);ubi32-nand-spi-er.0:128m(user)
mtdparts=mtdparts=ubicom32fc.0:512k(bootloader)ro,0(kernel),14080k(rootfs),15
  36k(jffs2),256k(fw_env);ubi32-nand-spi-er.0:128m(user)
ipaddr=192.168.0.100
stdin=serial
stdout=serial
stderr=serial

Environment size: 435/262140 bytes
```

Note: The size for the kernel is listed as '0'. If a partition has size 0 the kernel will look at the underlying memory and it finds a u-boot partition. If it does then it auto-sizes the zero sized partition to the correct size and steals that size from the following partition. So after booting you get this for example.

```
/ # cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00040000 "bootloader"
mtd1: 00400000 00040000 "kernel"
mtd2: 009c0000 00040000 "rootfs"
mtd3: 00180000 00040000 "jffs2"
mtd4: 00040000 00040000 "fw_env"
mtd5: 08000000 00020000 "user"
```

That feature allows us to install debug vs release kernels or different configured kernels without having to adjust the kernel command line.

3.2. Changing Environment Variables

Environment variables can easily be changed by using the command **setenv**. For example, to change the IP address of the board, type the following:

```
Ubicom =>setenv ipaddr 10.10.30.251
```

To confirm that the environment variable was indeed changed simply type:

```
Ubicom =>printenv ipaddr
ipaddr=10.10.30.251
```

3.3. Saving Environment Variables to Flash

The changes made in environment variables will not be permanent unless you save them. To see the new values of environment variables the next time you turn on your board, you need to make them permanent by saving them. The **saveenv** command is used to store environment variables permanently on flash:

```
Ubicom =>saveenv
Saving Environment to Flash...
Protect off B0FC0000 ... B0FFFFFF
Un-Protecting sectors 63..63 in bank 1
Un-Protected 1 sectors
Erasing Flash...Erase Flash from 0xb0fc0000 to 0xb0ffffff in Bank # 1
Erasing sector 63
Erased 1 sectors
Writing to Flash... done
Protecting sectors 63..63 in bank 1
Protected 1 sectors
Ubicom =>
```

4. Boot Process

When you turn on your Ubicom board, it is not the U-Boot that runs first. Before U-Boot, Ultra will launch. Ultra is a firmware specific to Ubicom boards. Ultra is in charge of initializing hardware components on boards and is alive as long as the board is up, unlike U-Boot whose ultimate task is to launch the operating system (Linux in this case). After being launched by U-Boot, the operating system replaces U-Boot, while Ultra continues to run.

When you turn on your board you will see the following message on the console:

```
Press SPACE to abort autoboot in 2 seconds
Ubicom =>
```

Unless you press SPACE within the specified number of seconds, U-Boot will try to execute the default boot command, which is equivalent to **bootm** by default and stored in **bootcmd** environment variable. You can set **bootcmd** to any one of the supported U-Boot commands or to a list of such commands, and they will run when U-Boot starts. The following commands are all valid.

```
Ubicom =>setenv bootcmd tftpboot
Ubicom =>setenv bootcmd printenv;devtree;tftpboot
```

Moreover, the default time value to press SPACE before executing **bootcmd** is 2 seconds and stored in **bootdelay** environment variable. You can set it to the value you like using **setenv** command.

On the other hand, if you press SPACE within the specified amount of time, the autoboot process is aborted, and you will see the U-Boot command line. You can manually carry on booting the operating system as well as carrying out other operations U-Boot command line supports.

4.1. SDK 2.0 Specific

As explained in the above paragraph, there are two distinct parts to booting. Before main kernel is active and while kernel is active. Ultra code executing during these stages is also different.

- Bootexec – Ultra, before kernel is active. (While U-Boot is active)
- Mainexec – Ultra, while kernel is active.

While both Bootexec and Mainexec perform similar activities with respect to hardware initialization and thread support, this portioning helps up helps in operations such as firmware upgrade.

5. Boot Methods

5.1. Booting from Memory

You can use the **bootm** command to boot using a locally stored operating system image:

```
Ubicom =>help bootm
bootm [addr [arg ...]]
    - boot application image stored in memory
  passing arguments 'arg ...'; when booting a Linux kernel,
  'arg' can be the address of an initrd image
```

The arguments to **bootm** are optional. The first argument to **bootm** is the memory address (in RAM, ROM, or flash memory) where the image is stored, followed by optional arguments that depend on the OS. In case of no address argument, **bootm** uses the value of **loadaddr** environment variable as the image address.

5.2. Booting from the Network

Remote booting requires a valid network configuration. To do so, make sure that the following environment variables are properly set:

- ethaddr
- ipaddr
- serverip
- bootfile

If the **serverip** is in a different network than your **ipaddr** you will also need to specify:

- netmask
- gatewayip

Otherwise, you might come across an error message like the following:

```
Ubicom =>tftpboot
  *** ERROR: `ipaddr' not set
```

U-Boot supports booting using an OS image located at a TFTP server. Recommended servers are **tftpd32** by Ph. Jounin on Windows and the **tftp-hpa** package on Linux. On windows you have to disable the DHCP, DNS and SNTP services of the application if you are in a network environment where they might conflict with other servers or Windows connection sharing.

```
Ubicom =>help tftpboot
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
```

The **tftpboot** comand has optional arguments too. When executed with no arguments, **tftpboot** will use the environment variables **loadaddr, serverip,** and **bootfile**.

Make sure that **loadaddr** points to a valid location in RAM. It points, by factory default, to a location in flash which is the start address of the OS image in flash.

The output of a successful boot via TFTP looks like the following:

```
Ubicom =>tftpboot 0xC4000000
SynopGMAC: Initializing synopsys GMAC interfaces eth_lan: (port = 0xba01c000,
   irq = 73)
SynopGMAC: GMAC reset completed in 369 clocks
SynopGMAC: synopGMAC_linux_open called
SynopGMAC: GMAC reset completed in 149 clocks
TFTP from server 192.168.0.6; our IP address is 192.168.0.100
Filename 'upgrade.ub'.
Load address: 0xc4000000
Loading: SynopGMAC: synopGMAC_task_poll:: Interrupt due to GMAC LINE module
#################################################################
    #################################################################
    #################################################################
    #################################################################
    #################################################################
    #################################################################
    #################################################################
    ############################
done
Bytes transferred = 7077888 (6c0000 hex)
SynopGMAC: synopGMAC_linux_close
```

In the case of U-Boot, **tftpboot** is more of a “tftpload” rather than a “tftpboot”. The **tftpboot** loads the remote image over the network into RAM starting at the specified **loadaddr**.

To actually boot the image, the **bootm** command needs to be issued as below:

```
Ubicom =>bootm
*  kernel: default image load address = 0xc4000000
## Booting kernel from Legacy Image at c4000000 ...
   Image Name:   Unknown - BOARD
   Image Type:   UBICOM32 Linux Multi-File Image (gzip compressed)
   Data Size:    7017466 Bytes =  6.7 MB
   Load Address: c0100000
   Entry Point:  c0100000
   Contents:
      Image 0: 25362 Bytes = 24.8 kB
      Image 1: 20 Bytes =  0 kB
      Image 2: 6992066 Bytes =  6.7 MB
   Verifying Checksum ... OK
   kernel data at 0xc4000050, len = 0x00006312 (25362)
## Loading init Ramdisk from multi component Legacy Image at c4000000 ...
   ramdisk start = 0xc4006364, ramdisk end = 0xc4006378
   Uncompressing (GZIP) Multi-File Image ... OK
   kernel loaded at 0xc0100000, end = 0xc010c20c
## Transferring control to Linux (at address c0100000) ...

Starting kernel ...

SynopGMAC: Now Calling network_unregister
[    0.000000] Linux version 2.6.28.10 (esawma@esawma-desktop) (gcc version
  4.4.1 20100807 (stable) (GCC) ) #1 Tue Aug 10 16:05:24 PDT 2010
[    0.000000] processor dram c0100000-d0000000, expecting c0100000-c2000000
[    0.000000] processor dram c0100000-d0000000, expecting c0100000-c2000000
[    0.000000] processor ocm bffc0b00-bfffdb00, expecting bffc0b00-bfffdb00
[    0.000000] IP8K Processor, Ubicom, Inc. <www.ubicom.com>
[    0.000000] Device Tree:
[    0.000000]    bffffc8: sendirq=255, recvirq=255,  name=board
[    0.000000]    bffff904: sendirq=255, recvirq=255,  name=bootargs
```

6. Programming the Flash

With two simple steps, it is also possible to write the image that is loaded into memory by **tftpboot** to flash: erase the flash and copy the image from RAM to flash.

Flash can be erased with the command **erase**. In the following example, we will use some of the values that we obtained during the **tftpboot** process as well as the command **flinfo**.

We only need to erase the amount of flash up to the size of the to-be-programmed image that resides in RAM. At the end of a successful transfer, the **tftpboot** command tells us the exact size of the image in bytes:

```
…….
Load address: 0xc4000000
Loading: SynopGMAC: synopGMAC_task_poll:: Interrupt due to GMAC LINE module
#################################################################
   #################################################################
   #################################################################
   #################################################################
   #################################################################
   #################################################################
   #################################################################
   ############################
done
Bytes transferred = 7077888 (6c0000 hex)
```

We need the actual flash address to which we are going to copy the image. Information about the flash can be obtained in U-Boot by issuing the command **flinfo**:

```
Ubicom =>flinfo

Bank # 1: flash_print_info
flash bank size is 16777216
sector count is 64
flash id is 26
  Sector Start Addresses:
    B0000000 (RO) B0040000 (RO) B0080000      B00C0000      B0100000
    B0140000      B0180000      B01C0000      B0200000      B0240000
    B0280000      B02C0000      B0300000      B0340000      B0380000
    B03C0000      B0400000      B0440000      B0480000      B04C0000
    B0500000      B0540000      B0580000      B05C0000      B0600000
    B0640000      B0680000      B06C0000      B0700000      B0740000
    B0780000      B07C0000      B0800000      B0840000      B0880000
    B08C0000      B0900000      B0940000      B0980000      B09C0000
    B0A00000      B0A40000      B0A80000      B0AC0000      B0B00000
    B0B40000      B0B80000      B0BC0000      B0C00000      B0C40000
    B0C80000      B0CC0000      B0D00000      B0D40000      B0D80000
    B0DC0000      B0E00000      B0E40000      B0E80000      B0EC0000
    B0F00000      B0F40000      B0F80000      B0FC0000
Ubicom =>
```

The example output shows flash space starts at B0000000. It also tells us that the first two sectors are read only (RO). They are reserved for the bootloader. This can be either the bootexec (Ultra) or bootexec and u-boot. In the latter case, the bootexec and u-boot are stored contiguously in the flash starting in the beginning of the flash. The

linux image is stored at B0080000. This can be verified by running **ubicom32-elf-objdump** on the **upgrade.elf** file and looking at the **.image** address, which in the case below is shown on line 20:

```
ubicom-linux-dist-2.0$ toolchain/bin/ubicom32-elf-objdump -h bin/upgrade.elf

bin/upgrade.elf:     file format elf32-ubicom32

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .fixed_text   00000010  c0100000  c0100000  00004000  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .syscall_text 00000018  bffc0030  c0100010  00004030  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .ocm_text     0000f0d8  bffc0b00  c0100028  00004b00  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .ocm_module_text 00010428  bffcfbd8  bffcfbd8  0086c004  2**0
                  CONTENTS
  4 .ocm_data     000012d0  bffe0000  c010f100  00014000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .text         00323c30  c01103d0  c01103d0  000183d0  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  6 .data         00024000  c0434000  c0434000  0033c000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .init         0050c000  c0458000  c0458000  00360000  2**2
                  CONTENTS, ALLOC, LOAD, CODE
  8 .eh_frame     00000004  c0964000  c0964000  0086c000  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  9 .bss          00042b14  c0968000  c0968000  0086c004  2**14
                  ALLOC
 10 .comment      00007f20  00000000  00000000  0087c42c  2**0
                  CONTENTS, READONLY
 11 .debug_aranges 0001d768  00000000  00000000  0088434c  2**0
                  CONTENTS, READONLY, DEBUGGING
 12 .debug_pubnames 0002fa57  00000000  00000000  008a1ab4  2**0
                  CONTENTS, READONLY, DEBUGGING
 13 .debug_info   027caaec  00000000  00000000  008d150b  2**0
                  CONTENTS, READONLY, DEBUGGING
 14 .debug_abbrev 001107af  00000000  00000000  0309bff7  2**0
                  CONTENTS, READONLY, DEBUGGING
 15 .debug_line   003db560  00000000  00000000  031ac7a6  2**0
                  CONTENTS, READONLY, DEBUGGING
 16 .debug_frame  0005b220  00000000  00000000  03587d08  2**2
                  CONTENTS, READONLY, DEBUGGING
 17 .debug_str    000ca6f1  00000000  00000000  035e2f28  2**0
                  CONTENTS, READONLY, DEBUGGING
 18 .debug_loc    0025d002  00000000  00000000  036ad619  2**0
                  CONTENTS, READONLY, DEBUGGING
```

```
19 .debug_ranges 000c0d70  00000000  00000000  0390a61b  2**0
                 CONTENTS, READONLY, DEBUGGING
20 .image        006c0000  b0080000  b0080000  039cb38b  2**0
                 CONTENTS
21 .downloader   00000df8  d0000000  d0000000  0408b38b  2**0
                 CONTENTS
```

Note that the size we recorded under the **tftpboot** transfer matches the size of the **.image** section obtained through **ubicom32-elf-objdump**. With the information gathered in the above steps, we can now erase the flash to prepare it for programming:

```
Ubicom =>erase B0080000 +6c0000
Erase Flash from 0xb0080000 to 0xb073ffff in Bank # 1
Erasing sector 2
Erasing sector 3
Erasing sector 4
Erasing sector 5
Erasing sector 6
Erasing sector 7
Erasing sector 8
Erasing sector 9
Erasing sector 10
Erasing sector 11
Erasing sector 12
Erasing sector 13
Erasing sector 14
Erasing sector 15
Erasing sector 16
Erasing sector 17
Erasing sector 18
Erasing sector 19
Erasing sector 20
Erasing sector 21
Erasing sector 22
Erasing sector 23
Erasing sector 24
Erasing sector 25
Erasing sector 26
Erasing sector 27
Erasing sector 28
Erased 27 sectors
```

Once the flash is erased, the image residing in RAM can be copied to flash using the **cp.b** command. The **.b** indicates that the amount (count) to be copied is specified in bytes.

```
Ubicom =>cp.b 0xC4000000 0xB0080000 0x6c0000
Copy to Flash... done
```

To test the new image copied to flash, you can issue the **reset** command to reboot the system. If the default boot method in U-Boot is set to **bootm**, then as soon as the system is reset, the newly programmed image will be booted.

6.1. SDK 2.0 Specific

In SDK 2.0, the packing is better: boot decompressor, bootexec and u-boot are all packed in the first two sectors (the ones that are covered by the `bootloader' partition. The size of that bootloader partition can be set with an LPJ define (and overwritten in individual BRD files).

7. Useful Links

U-Boot: http://www.denx.de/wiki/U-Boot