Introduction to U-Boot bootloader

Behan Webster <behanw@converseincode.com>

June, 2020
Behan Webster

- Embedded Linux consultant
- Linux kernel engineer
- Trainer and Course Author for the Linux Foundation
Booting a computer

Multi-stage bootloader

**First stage bootloader**
- Resides on reset vector
- Often on-chip BootROM
- Initializes HW and loads the next stage

**User bootloader**
- First user-controlled code

**Linux kernel**

**Userspace**
U-Boot bootloader

First stage bootloader

U-Boot SPL
- First user-controlled code
- Responsible for additional HW initialization
- Loads U-Boot or Linux directly

U-Boot
- Bootloader with interactive shell
- Boot monitor
- Debug tool

Linux kernel

Userspace
U-Boot example

U-Boot SPL 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29)

Trying to boot from MMC1

U-Boot 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29 -0600)

CPU : AM335X-GP rev 2.1
I2C:  ready
DRAM:  512 MiB
Reset Source: Global warm SW reset has occurred.
Reset Source: Power-on reset has occurred.
MMC:  OMAP SD/MMC: 0, OMAP SD/MMC: 1

Model: BeagleBoard.org PocketBeagle
Net:  usb_ether
Press SPACE to abort autoboot in 2 seconds
=>
U-Boot SPL vs U-Boot

**SPL** – Secondary Program Loader
- Built from the same source as U-Boot
- Significantly reduced size and feature set
- Used to init system and start U-Boot or Linux

**TPL** – Tertiary program loader
- Built from the same source as U-Boot
- Even smaller than SPL
- Almost never used
- Used on severely limited system (ie. OneNAND)
Basic U-Boot commands
U-Boot shell

- There are two shell choices
  1. original old shell (which has no name)
  2. HUSH shell (which has more features)
- For most situations HUSH is the better choice
- Similar to bourne shell
- Persistent environment support
- Scripting support
The 'help' command

- Provides detailed built-in help text
- Can provide further details on specific command

```
=> help
? - alias for 'help'
bdinfo - print Board Info structure
bootm - boot application image from memory
cmp - memory compare
coninfo - print console devices and information
crc32 - checksum calculation
dhcp - boot image via network using DHCP/TFTP protocol
echo - echo args to console
go - start application at address 'addr'
help - print command description/usage
i2c - I2C sub-system
load - load binary file from a filesystem
usb - USB sub-system
```
Running 'help' on a command

▶ Just run help on any command for more information

```
1 => help echo
2 echo - echo args to console
3
4 Usage:
5 echo [args..]
6    - echo args to console; \c suppresses newline
7
8 => help bdinfo
9 bdinfo - print Board Info structure
10
11 Usage:
12 bdinfo
```
help on sub-commands

1 => help usb
2 usb - USB sub-system

Usage:
3 usb start - start (scan) USB controller
4 usb reset - reset (rescan) USB controller
5 usb stop [f] - stop USB [f]=force stop
6 usb tree - show USB device tree
7 usb info [dev] - show available USB devices
8 usb test [dev] [port] [mode] - set USB 2.0 test mode
9 (specify port 0 to indicate the device's upstream port)
10 Available modes: J, K, S[EO_NAK], P[acket], F[orce_Enable]
The 'echo' command

► Useful for printing text
► It does **NOT** interpret control sequences
  (except for `\c` to suppress newline)

```
1 => echo hello world
2 hello world
3 => echo foo\c ; echo bar
4 foobar
```
The 'bdinfo' command

▶ Probing system info

1  => bdinfo
2  arch_number = 0x00000E05
3  boot_params  = 0x80000100
4  DRAM bank    = 0x00000000
5  -> start     = 0x80000000
6  -> size      = 0x20000000
7  eth0name     = usb_ether
8  ethaddr      = 60:64:05:f4:79:7f
9  current eth  = usb_ether
10 ip_addr       = 192.168.1.2
11 baudrate      = 115200 bps
12 TLB addr      = 0x9FFF0000
13 relocaddr     = 0x9FF44000
14 reloc off     = 0x1F744000
15 irq_sp        = 0x9DF23EC0
16 sp start      = 0x9DF23EB0
17 Early malloc usage: 2a8 / 400
Getting further help

- Source, documentation in doc/
  https://git.denx.de/?p=u-boot.git;a=tree;f=doc
- IRC: irc.freenode.net #u-boot
- Mailing list: u-boot@lists.denx.de
U-Boot memory access commands
Memory access commands, 'mw', 'md'

- Useful for reading/writing memory and registers
- Support for byte/word/long/quad register access using suffixes (.b, .w, .l, .q)
- Default access width is long, 32bit (md = md.l)
- Support for reading multiple units at a time (default 0x40)
- Default for read is updated if number of units specified
- Can read subsequent addresses if no address specified

```
1  => mw 0x81000000 0x1234abcd
2  => md.l 0x81000000 0x8
3  81000000: 1234abcd 00000000 00000000 00000000 ..4............
4  81000010: 00000000 00000000 00000000 00000000 ................
5  => md.w 0x81000000 0x8
6  81000000: abcd 1234 0000 0000 0000 0000 0000 0000 0000 0000 0000 ..4............
7  => md.b 0x81000000 0x8
8  81000000: cd ab 34 12 00 00 00 00 ..4..
9  =>
10 81000008: 00 00 00 00 00 00 00 00 ...........
```
Memory access commands, 'mw', 'md'

► Try toggling GPIOs the hard way
► Note on used bitfields:
  |xxxx|xxx0|000x|xxxx|xxxx|xxxx|xxxx|xxxx|xxxx|
► Expected result: Two Blue LEDs ON/OFF

1 => echo "Try toggling GPIOs the hard way"
2 => md 0x4804c130 4
3 4804c130: 00000002 ffffffff f0000300 00000000 ................
4 => mw 0x4804c134 0xfe1fffff
5 => mw 0x4804c13c 0x00a00000
6 => mw 0x4804c13c 0x01400000
7 => md 0x4804c130 4
8 4804c130: 00000002 fe1fffff f140300 01400000 ...........@...@.
Memory modification commands, 'mm', 'nm'

▶ Useful for interactively modifying registers
▶ Same properties as for md/mw apply
▶ mm autoincrements address, nm does not
▶ Use 'q' to drop back to U-Boot shell
▶ Use '-' to return to previous address
▶ Press 'Enter' without value to skip current address

```
1 => mm 0x4804c134
2 4804c134: fffffff ? fe1fffff
3 4804c138: f0002300 ?
4 4804c13c: 00000000 ? 00400000
5 4804c140: 00000000 ? q
7 =>
```
Memory access commands, 'cp', 'cmp'

- cp – copy memory
- cmp – compare memory
- Same properties as md/mw above apply

1. => mw 0x81000000 0x1234abcd 0x10
2. => cp 0x81000000 0x82000000 0x8
3. => cmp 0x81000000 0x82000000 0x8
4. Total of 8 word(s) were the same
5. => cmp 0x81000000 0x82000000 0x9
6. word at 0x81000020 (0x1234abcd) != word at 0x82000020 (0xea000003)
7. Total of 8 word(s) were the same
U-Boot environment and scripting commands
U-Boot environment

- The environment uses key-value storage
- Can contain values or scripts
- Default env built into U-Boot binary
- Optional custom env loaded from storage
- Live copy in RAM
- Can be accessed as variables
- Can be modified
- Can be made persistent
The 'printenv' command

- For printing the environment
- Legacy alias for 'env print'

```bash
1  => env print
2  arch=arm
3  ...
4  Environment size: 26907/131068 bytes
5  =>
6  => env print arch
7  arch=arm
8  => printenv arch
9  arch=arm
10 => echo "$arch"
11 arm
```
The 'setenv'/’askenv’/’editenv’ command

- For modifying the environment
- Legacy alias for ‘env set’/’env ask’/’env edit’

```
1 => env set foo bar
2 => env print foo
3 foo=bar

5 => env ask quux "Set quux to ?"
6 Set quux to ? 1234
7 => env print quux
8 quux=1234

10 => env edit quux
11 edit: 24
12 => env print quux
13 quux=24
```
Removing variables

▶ Set variables to empty to remove them from the environment

```bash
1 => env print foo
2 ## Error: "foo" not defined
3 => env set foo bar
4 => env print foo
5 foo=bar
6 => env set foo
7 => env print foo
8 ## Error: "foo" not defined
```
The 'saveenv' command

- For environment persistency
- Environment is not persistent across reboots by default
- Any changes to environment are done to the live copy

```
1 => env set foo bar
2 => env print foo
3  foo=bar
4 => reset
5 => env print foo
6 ## Error: "foo" not defined
7
8 => env set foo bar
9 => saveenv
10 => reset
11 => env print foo
12  bar
```
The 'run' command

- For running scripts in the environment
- Chaining commands with ';' is possible
- Note that ';' ignores return value

1 => env set foo 'echo hello'
2 => run foo
3 hello
4
5 => env set foo 'echo hello ; echo world'
6 => run foo
7 hello
8 world
Variables in environment

- Proper escaping is important on U-Boot shell
- Be careful with variable expansion

1 => env set foo bar
2 => env set quux echo $foo
3 => env set foo baz
4 => run quux
5 bar
6 => env print quux
7 quux=echo bar
8
9 => env set quux echo \$foo
10 => env print quux
11 => env set quux 'echo $foo'
12 => env print quux
Special variables

Certain variables have special meaning/function

- `ver` – U-Boot version
- `stdin`, `stdout`, `stderr` – Redirection of STDIO. Setting these has immediate impact, also see `coninfo` command.
- `loadaddr` – Default load address
- `filesize` – Size of the loaded file
- `bootargs` – Boot arguments passed to Linux command line
- `bootcmd` – Default boot command (see boot command and autoboot)
- `preboot` – Script executed before autoboot
- `ipaddr`, `netmask`, `serverip`, `gatewayip` – Network settings
- `ethaddr`, `eth1addr`, ... – Ethernet MAC address
The 'setexpr' command

- Environment manipulation multi-tool
- Supports loading memory content into variables
- Supports arithmetic operations on both variables and memory (AND, OR, XOR, +, -, *, /, MOD)
- Supports basic regex manipulation on strings and variables

1 => md 0x9ff4e000 1
2 9ff4e000: ea0000b8
3 => setexpr foo *0x9ff4e000
4 => env print foo
5  foo=ea0000b8
6
7 => env set foo 1 ; env set bar 2
8 => setexpr baz $foo + $bar
9 => env print baz
10 baz=3
11
12 => setexpr foo gsub ab+ x "aabbcc"
13 foo=axcc
U-Boot shell conditional expressions and loops
The 'true'/false' commands

- Return 0 (true) / non-zero (false) return values
- U-Boot supports handling return values of commands
- Automatic variables are supported too

```bash
1 => true
2 => echo $? 
3 0
4 => false
5 => echo $? 
6 1
```
Conditional expressions

- The if conditional is supported
- Shorthand || and && expressions also supported
- Warning, the "if ! foo ; then ... fi" is not supported, use ie. "if foo ; then false ; else ... fi" as a workaround

```bash
1 => if true ; then echo "hello" ; else echo "bye" ; fi
2 hello
3 => false || echo "false!"
4 false!
5
6 => env set foo 'true && echo "true!''
7 => run foo
8 true!
```
The 'test' command

▶ Minimal test command from HUSH

```
1 => env set i 4
2 => test $i -lt 5
3 => echo $?
4 0
5 => env set i 6
6 => test $i -lt 5
7 => echo $?
8 1
9
10 => env set i 6
11 => if test $i -lt 5 ; then echo "Less then 5" ; \
12   else echo "More than 5" ; fi
13 More than 5
```
The 'for' loop

- The for loop over a list of elements

```bash
for i in a b c d; do echo "$i"; done
```

a
b
c
d
The 'while' loop

- The while loop with a condition
- Ctrl-c can be used to break an endless loop

```
while true ; do
  echo hello ;
done
```

```
hello
hello
hello
```

```
Ctrl-c
```

```
=>
```
U-Boot data loading commands
Loading from storage

- U-Boot supports loading from various storage types
  - SD/MMC – mmc command
  - USB – usb command
  - SATA – sata command
  - NAND – nand command
  - ...

- Both RAW storage and filesystems are supported
  - Universal FS access – ’ls’, ’load’ commands
  - ExtFS – legacy ’extls’/’extload’ command
  - VFAT – legacy ’fatls’/’fatload’ command
  - UBI/UBIFS – ’ubi’ command
  - ...
Loading from SD card

1. => mmc rescane
2. => mmc part

Partition Map for MMC device 0 -- Partition Type: DOS

<table>
<thead>
<tr>
<th>Part</th>
<th>Start Sector</th>
<th>Num Sectors</th>
<th>UUID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8192</td>
<td>6955008</td>
<td>1147c091-01</td>
<td>83 Boot</td>
</tr>
</tbody>
</table>

3. => ls mmc 0:1
4. <DIR> 4096 .
5. <DIR> 4096 ..
6. 40 ID.txt

7. ...

8. => load mmc 0:1 $loadaddr ID.txt
9. => md.b $loadaddr $filesize

10. 82000000: 42 65 61 67 6c 65 42 6f 61 72 ... BeagleBoard.org
11. 82000010: 44 65 62 69 61 6e 20 49 6d 61 ... Debian Image 201
Loading from network

- U-Boot network stack is UDP-only (no TCP)
- Support for TFTP, NFS (over UDP), DHCP/BOOTP, ...
- ping – ICMP Echo
- tftp – TFTP download (tftpput for upload)
- dhcp – Obtain settings from DHCP and ev. load file

```
1 => env set ethaddr 00:aa:bb:cc:dd:ee  # optional!
2 => env set ipaddr 192.168.1.300
3 => env set netmask 255.255.255.0
4 => env set serverip 192.168.1.1
5 => ping $serverip
6 => tftp $loadaddr $serverip:somefile
7 => dhcp $loadaddr $serverip:somefile
```
Loading over serial port

- When nothing else, UART is available
- U-Boot supports X/Y modem, Srecord and kermit protocol

U-Boot> loady
<send file over ymodem protocol, e.g. sb -T>

Example with GNU screen

$ screen /dev/ttyUSB0 115200
$ => loady
$ ctrl-a:exec !! sb -T yourbinary.bin

or from another shell on the same host computer:

$ screen -x -r -X exec \!!\!! sb -T yourbinary.bin
Booting the kernel
Booting the kernel

There are many image formats

- (z)Image
  - Linux binary (with decompressor)
  - No protection against bitrot
  - Just set up registers and jump to it
  - Optional separate Device Tree

- uImage
  - Legacy since forever
  - Wrapper around arbitrary binary
  - CRC32 checksum and small amount of metadata
  - Wraps single file only
  - Optional separate Device Tree

- fitImage – multi-component image
  - Based on Device Tree
  - Supports multiple files
  - Configurable checksum algorithm per entry
  - Supports digital signatures
Booting kernel image

- bootz – (z)Image
- booti – ARM64 Image
- bootm – fitImage, ulImage
- $bootcmd – default boot command

1 => help bootz
2 bootz – boot Linux zImage image from memory

Usage:

3 bootz [addr [initrd[:size]] [fdt]]
4    - boot Linux zImage stored in memory
5          The argument 'initrd' is optional... The optional arg
6      ':size' allows specifying the size of RAW initrd.
7
8 When booting a Linux kernel which requires a flat
9    device-tree a third argument is required which is
10   the address of the device-tree blob.
Booting kernel image

1 => env set bootargs console=tty00,115200
2 => load mmc 0:1 0x82000000 boot/zImage-4.9.82-ti-r102
3 9970640 bytes read in 673 ms (14.1 MiB/s)
4 => load mmc 0:1 0x88000000 boot/dtbs/4.9.82-ti-r102/
   \ am335x-pocketbeagle.dtb
5 132769 bytes read in 180 ms (719.7 KiB/s)
6 => bootz 0x82000000 - 0x88000000
7 ## Flattened Device Tree blob at 88000000
8   Booting using the fdt blob at 0x88000000
9   Loading Device Tree to 8ffdc000, end 8ffff6a0 ... OK
10
11 Starting kernel ...
12
13 [ 0.000000] Booting Linux on physical CPU 0x0
14 [ 0.000000] Linux version 4.9.82-ti-r102 \n15 (root@b2-am57xx-beagle-x15-2gb) (gcc version 6.3.0 20170516
16 (Debian 6.3.0-18) ) #1 SMP PREEMPT Thu Feb 22 01:16:12 UTC 2
17 [ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7

Device Tree

- Data structure describing the hardware
- Usually passed to OS to provide information about HW topology where it cannot be detected or probed
- An acyclic graph, made of named nodes containing properties
  - Nodes can contain properties and child nodes
  - Properties are a name-value pairs
  - See https://en.wikipedia.org/wiki/Device_tree
- Device Tree properties can refer to other nodes by the used of phandles (references to other nodes)
  - phandles provide simple references to device node labels (e.g. "<&L2>" is reference to L2 cache node)
  - phandles can be used to reference nodes anywhere in the device tree
Device Tree example

```
1 /dts-v1/;
2 #include "arm-realview-eb-mp.dtsi"
3 /
4 model = "ARM RealView EB Cortex A9 MPCore";
5 [
6   cpus {
7     #address-cells = <1>;
8     #size-cells = <0>;
9     enable-method = "arm,realview-smp";
10    A9_0: cpu@[0 {
11       device_type = "cpu";
12       compatible = "arm,cortex-a9";
13       reg = <0>;
14       next-level-cache = &L2;
15    }]
16 [
17   pmu: pmu@[0 {
18     interrupt-affinity = &A9_0, &A9_1, &A9_2, &A9_3;
19    }]
20  }
```
fitImage example (1)

```c
/dts-v1/;
/
 {

description = "Linux kernel and FDT blob for sockit";

images {

    kernel@1 {
        description = "Linux kernel";
        data = /incbin/("./arch/arm/boot/zImage");
        type = "kernel";
        arch = "arm";
        os = "linux";
        compression = "none";
        load = <0x00008000>;
        entry = <0x00008000>;
        hash@1 {
            algo = "crc32";
        };
    };
```
fitImage example (2)

```c
fdt@1 {  
    description = "Flattened Device Tree blob";
    data = /incbin/("./arch/arm/boot/dts/socfpga....dtb");
    type = "flat_dt";
    arch = "arm";
    compression = "none";
    hash@1 {  
        algo = "crc32";
    };
};
```
configurations {
    default = "conf@1";
    conf@1 {
        description = "Boot Linux kernel with FDT blob";
        kernel = "kernel@1";
        fdt = "fdt@1";
        hash@1 {
            algo = "crc32";
        }
    }
};

Compile with

mkimage -f fit-image.its fitImage
fitImage commands

- bootm $fitimageaddr – boot fitImage/uImage
- iminfo – print image information
- imxtract – extract file from fitImage/uImage

1 => iminfo $loadaddr
2
3 ## Checking Image at 82000000 ...
4 FIT image found
5 FIT description: Linux kernel and FDT blob for am335x-pocketbeagle
6 Created: 2018-09-03 0:46:36 UTC
7 Image 0 (kernel@1)
8 Description: Linux kernel (Mon Sep 3 02:46:36 CEST 2018)
9 Created: 2018-09-03 0:46:36 UTC
10 Type: Kernel Image
11 Compression: uncompressed
12 Data Start: 0x82000154
13 Data Size: 5665328 Bytes = 5.4 MiB
14 Architecture: ARM
15 OS: Linux
16 Load Address: 0x80008000
17 Entry Point: 0x80008000
18 Hash algo: crc32
19 Hash value: 1a1062ee
20 ...
21 => imxtract $loadaddr kernel@1 0x8a000000
22 ## Copying 'kernel@1' subimage from FIT image at 82000000 ...
23 crc32+ Loading part 0 ... OK
24 => md 0x8a000000
25 8a000000: e1a00000 e1a00000 e1a00000 e1a00000 ...............
26 8a0000010: e1a00000 e1a00000 e1a00000 e1a00000 ...............
27 8a0000020: ea000005 016f2818 00000000 00567230 .....(o......0rV.
The 'fdt' command

- FDT manipulation
- `fdt addr` – Tell U-Boot where the FDT is
- `fdt resize` – Add extra space to FDT
- `fdt print` – Print DT path
- `fdt set` – Add or change DT entry

```
1 => load mmc 0:1 0x88000000 boot/dtbs/4.9.82-ti-r102/am335x-pocketbeagle.dtb
2 132769 bytes read in 180 ms (719.7 KiB/s)
3 => fdt addr 0x88000000
4 => fdt resize
5 => fdt print /chosen
6  chosen {
7      stdout-path = "/ocp/serial@44e09000";
8  }
9 => fdt set /chosen/ foo bar
10 => fdt print /chosen
11 chosen {
12      foo = "bar";
13      stdout-path = "/ocp/serial@44e09000";
14  }
15 => bootz 0x82000000 - 0x88000000
```
Miscellaneous U-Boot commands
The 'gpio' command

- Useful for toggling/sampling GPIOs
- GPIO input sets return value
- gpio input - to read a gpio
- gpio set - to set a gpio
- gpio clear - to clear a gpio
- gpio toggle - to toggle a gpio

1 => gpio input 45
2 gpio: pin 45 (gpio 45) value is 1
3 => echo $?
4 1
5 => gpio set 53
6 gpio: pin 53 (gpio 53) value is 1
The 'i2c' command

- Useful for accessing I2C bus
- `i2c bus` – lists available I2C busses
- `i2c dev` – select an I2C bus
- `i2c md` – read registers from I2C device
- `i2c mw` – write registers to I2C device
- `i2c probe` – probe for devices on I2C
- `i2c speed` – set I2C bus speed

1. `=> i2c dev 2`
2. Setting bus to 2
3. `=> i2c probe`
4. Valid chip addresses: 1C
5. `=> i2c md 0x1c 0x0 0x8`
6. 0000: 00 41 ac 01 fc 7f 10 00 .A......
Compiling U-Boot from source
U-Boot sources

- Git master at:
  http://git.denx.de/?p=u-boot.git;a=summary
- on Github:
  https://github.com/u-boot/u-boot
- Custodian subtrees at:
  http://git.denx.de/?p=u-boot.git;a=forks
- Available via Git and HTTP protocols
Building the sources

1 $ git clone git://git.denx.de/u-boot.git
2 $ cd u-boot
3 $ export CROSS_COMPILE=arm-linux-gnueabihf-  # optional, set cross compiler
4 $ make am335x_evm_defconfig
5 $ make

▶ U-Boot sandbox target (sandbox_defconfig)
U-Boot running as userspace application

▶ U-Boot QEMU targets, (qemu_defconfig)
U-Boot running in QEMU as "BIOS"
qemu-system-arm -M virt -bios u-boot.bin
Practical labs

- Examples work with a PocketBeagle and Techlab
- https://beagleboard.org/pocket
- https://beagleboard.org/techlab
Task 0

Enter U-Boot prompt

▶ HINT: Press SPACE to stop autoboot
Task 0

```
1 Model: BeagleBoard.org PocketBeagle
2 <ethaddr> not set. Validating first E-fuse MAC
3 Net: No ethernet found.
4 Press SPACE to abort autoboot in 2 seconds
5 =>
```
Task 1

Boot kernel from SD card

- Check if SD card contains zImage and DTB.
- Load both into memory
- Set $bootargs
- Boot the kernel with DT
- HINT: mmc rescan, ls, load, bootz commands
Task 1

```
1 => env set bootargs root=/dev/mmcblk0p1 rootfstype=ext4 rootwait \
    console=tty00,115200
2 => mmc rescan
3 => load mmc 0:1 0x82000000 boot/vmlinuz-4.14.91-ti-r90
4 => load mmc 0:1 0x88000000 boot/dtbs/4.14.91-ti-r90/
    am335x-pocketbeagle-techlab.dtb
5 => bootz 0x82000000 - 0x88000000
6 9970640 bytes read in 6594 ms (1.4 MiB/s)
7 132769 bytes read in 123 ms (1 MiB/s)
8 ## Flattened Device Tree blob at 88000000
9   Booting using the fdt blob at 0x88000000
10  Loading Device Tree to 8ffdc000, end 8ffff6a0 ... OK
11
12 Starting kernel ...
13
14   [0.000000] Booting Linux on physical CPU 0x0
```
Task 2

Boot kernel from SD card with adjusted DT

► Change /model property in DT and boot kernel with it
► HINT: mmc rescan, load, fdt addr, fdt set, bootz commands
► HINT: in Linux see cat /proc/device-tree/model
Task 2

1 => env set bootargs root=/dev/mmcblk0p1 rootfstype=ext4 rootwait \       
2       console=tty00,115200
3 => mmc rescan
4 => load mmc 0:1 0x82000000 boot/vmlinuz-4.14.91-ti-r90
5 => load mmc 0:1 0x88000000 boot/dtbs/4.14.91-ti-r90/\       
6       am335x-pocketbeagle-techlab.dtb
7 => fdt addr 0x88000000
8 => fdt set / model "Something"
9 => fdt list
10 / {  
11       ...
12       compatible = "ti,am335x-pocketbeagle", "ti,am335x-bone", "ti,am33xx";
13       model = "Something";
14       chosen {
15       ...
16    };
17 => bootz 0x82000000 - 0x88000000
18 ...
19 [ 0.000000] OF: fdt: Machine model: Something
20 ...
21 beaglebone login:debian
22 debian@beaglebone:~$ dmesg | grep model
23 [ 0.000000] OF: fdt: Machine model: Something
Task 3

**Button input**

- **HINT:** 'gpio input' command
- **HINT:** 0x4804c138 is the offset of the GPIO input register
- **HINT:** gpio 45 is the USR button GPIO
Task 3

1 == if gpio input 45 ; then
2     echo "Button pressed" ;
3     else
4     echo "Button not pressed" ;
5     fi
Blink USR LED using direct HW IO

▶ HINT: for or while commands
▶ HINT: 0x4804c134 is the offset of the GPIO direction register
  Use the following to set the four pins as outputs
  mw 0x4804c134 0xfe1fffff
▶ HINT: 0x4804c13c is the offset of the GPIO value register
  Use the following to set LED 0 on
  mw 0x4804c13c 0x00200000
▶ HINT: sleep 1 waits 1 second
▶ HINT: look at 'base' command
Task 4

1  =>  mw 0x4804c134 0xfe1fffff
2  =>  while true ; do
3       mw 0x4804c13c 0x00200000 ;
4       sleep 1 ;
5       mw 0x4804c13c 0x00000000 ;
6       sleep 1 ;
7     done
Task 5

Implement moving light using USR LEDs using GPIO command

▶ HINT: for or while commands
▶ HINT: LEDs are GPIO 53, 54, 55, 56
▶ HINT: sleep 1 waits 1 second
▶ HINT: look at 'base' command
Task 5

```bash
while true ; do
    for i in 53 54 55 56 ; do
        gpio set $i ;
        sleep 1 ;
        gpio clear $i ;
    done
done
```
Task 6

Conveniently load custom environment using Ymodem

▶ HINT: loady and env import commands
Task 6

```bash
linux$ cat << EOF > /tmp/env.txt
  > hello=world
  > foo=bar
  > EOF

=> loady

## Ready for binary (ymodem) download to 0x82000000 at 115200 bps...
ctrl-a:exec !! sb -T /tmp/env.txt
C## Total Size = 0x00000014 = 20 Bytes
=> md.b $loadaddr $filesize
82000000: 68 65 6c 6c 6f 3d 77 6f 72 6c 64 0a 66 6f 6f 3d hello=world.foo=
82000010: 62 61 72 0a bar.
=> env import $loadaddr $filesize
## Warning: defaulting to text format
=> env print hello
hello=world
```
Task 7

Compiling U-Boot to run in sandbox mode

▶ Clone U-Boot sources, configure them for sandbox, compile U-Boot.

▶ HINT: On your host machine

```
1 $ git clone git://git.denx.de/u-boot.git
2 $ cd u-boot
3 $ make sandbox_defconfig
4 $ make -j $(nproc)
5 $ ./u-boot
```
Task 7

$ make sandbox_defconfig
HOSTCC  scripts/basic/fixedef
...
#
# configuration written to .config
#

$ make -j $(nproc)
scripts/kconfig/conf  --syncconfig Kconfig
CHECK  include/config.h
UPD    include/config.h
...
CFGCHK  u-boot.cfg
$ ./u-boot

U-Boot 2018.11-rc1-00033-ga16accc9fb (Oct 07 2018 - 17:13:29 +0200)
Model: sandbox
DRAM: 128 MiB
... =>
Barcode reader example

- Imagine that an ethernet MAC address is entered into U-Boot from a barcode reader which is formatted incorrectly.
- Filter the MAC address out and ignore the separators.
- The input should read with "env ask" and be something like "00xaaxbbxccxdxexe".
- Use setexpr to change the input into a proper MAC address (i.e. "00:aa:bb:cc:dd:ee").
- Assume the list of separators is known and fixed as "xyz".
1 => env ask mac 'MAC address ?'
2  MAC address ? 00xaaxbbxccxddxee
3 => setexpr myethaddr gsub '\((.\))\[xyz\]' '\1': $mac
4 myethaddr=00:aa:bb:cc:dd:ee
Task 9

Playing with the Techlab Accelerometer

▶ Read out the MMA8452Q accelerometer data
▶ HINT: i2c commands
▶ HINT: Accelerometer is on bus 2, select bus 2

1 => i2c dev 2

▶ HINT: Accelerometer has I2C address 0x1c, try

1 => i2c md 0x1c 0 0x10

▶ HINT: Accelerometer is in standby, wake it up with

1 => i2c mw 0x1c 0x2a 0x1

then try reading samples at offset 0x1..0x6 again
Task 9

1 => i2c dev 2
2 => i2c md 0x1c 0 0x10
3 => i2c mw 0x1c 0x2a 0x1
4 => while true ; do i2c md 0x1c 0x2a 0x3; done

bjs