Introduction to U-Boot bootloader

Marek Vašut <marek.vasut@gmail.com>

March, 2018

Marek Vasut

- ► Software engineer
- ► Versatile Linux kernel hacker
- ► Custodian at U-Boot bootloader
- ► OE-core contributor (Yocto...)
- ► FPGA enthusiast

Booting a computer

- Multi-stage bootloader
  - First stage on reset vector
  - Often a BootROM
  - Inits HW, loads next stage
- OS kernel
- Userspace

U-Boot bootloader

- ▶ Boot loader
    - ▶ First[1]-ish code that runs on a system
    - ▶ Responsible for some HW initialization and starting OS
- ▶ Boot monitor
- ▶ Debug tool

---

[1]There are exceptions, ie. Boot ROMs

U-Boot example

```
1 U-Boot SPL 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29)
2 Trying to boot from MMC1
3
4
5 U-Boot 2018.01-00002-g9aa111a004 (Jan 20 2018 - 12:45:29 -0600)
6
7 CPU  : AM335X-GP rev 2.1
8 I2C:   ready
9 DRAM:  512 MiB
10 Reset Source: Global warm SW reset has occurred.
11 Reset Source: Power-on reset has occurred.
12 MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
13
14 Model: BeagleBoard.org PocketBeagle
15 Net:   usb_ether
16 Press SPACE to abort autoboot in 2 seconds
17 =>
```

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
- TPL – Tertiary program loader
- TPL is almost never used
- Even smaller than SPL

The 'echo' command

- ▶ Useful for printing text
- ▶ Does NOT interpret control sequences (except for c to suppress newline)

```
1 => echo hello world
2 hello world
```

The 'help' command

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

```
1 => help
2 ?       - alias for 'help'
3 bdinfo  - print Board Info structure
4 bootm   - boot application image from memory
5 cmp     - memory compare
6 coninfo - print console devices and information
7 crc32   - checksum calculation
8 dhcp    - boot image via network using DHCP/TFTP protocol
9 echo    - echo args to console
10 go      - start application at address 'addr'
11 help    - print command description/usage
12 i2c     - I2C sub-system
13 load    - load binary file from a filesystem
14 usb     - USB sub-system
```

Getting further help

```
 1  => help usb
 2  usb - USB sub-system
 3
 4  Usage:
 5  usb start - start (scan) USB controller
 6  usb reset - reset (rescan) USB controller
 7  usb stop [f] - stop USB [f]=force stop
 8  usb tree - show USB device tree
 9  usb info [dev] - show available USB devices
10  usb test [dev] [port] [mode] - set USB 2.0 test mode
11      (specify port 0 to indicate the device's upstream port)
12      Available modes: J, K, S[E0_NAK], P[acket], F[orce_Enable]
```

- ▶ Source, documentation in doc/
  http://git.denx.de/?p=u-boot.git;a=tree;f=doc
- ▶ IRC: irc.freenode.net #u-boot
- ▶ ML: u-boot@lists.denx.de

The 'bdinfo' command

- Probing system info

```
=> bdinfo
arch_number = 0x00000E05
boot_params = 0x80000100
DRAM bank   = 0x00000000
-> start    = 0x80000000
-> size     = 0x20000000
eth0name    = usb_ether
ethaddr     = 60:64:05:f4:79:7f
current eth = usb_ether
ip_addr     = 192.168.1.2
baudrate    = 115200 bps
TLB addr    = 0x9FFF0000
relocaddr   = 0x9FF44000
reloc off   = 0x1F744000
irq_sp      = 0x9DF23EC0
sp start    = 0x9DF23EB0
Early malloc usage: 2a8 / 400
```

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 ..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|
- ▶ 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 f1400300 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: ffffffff ? fe1fffff
3 4804c138: f0002300 ?
4 4804c13c: 00000000 ? 00400000
5 4804c140: 00000000 ? q
6 =>
```

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 \
7     (0xea000003)
8 Total of 8 word(s) were the same
```

U-Boot shell

- There are two – HUSH and the old no-name
- Similar to bourne shell
- Persistent environment support
- Scripting support

U-Boot environment

- key-value storage
- Can contain values or even 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'

```
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 bar
4
5 => env ask    quux "Set quux to ?"
6 Set quux to ? 1234
7 => env print quux
8 quux=1234
9
10 => env edit   quux
11 edit: 24
12 => env print quux
13 quux=24
```

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 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 => setenv foo bar
 2 => setenv quux echo $foo
 3 => setenv foo baz
 4 => run quux
 5 bar
 6 => printenv quux
 7 quux=echo bar
 8
 9 => setenv quux echo \$foo
10 => printenv quux
11 => setenv quux 'echo $foo'
12 => printenv 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 cfr coninfo command
- loadaddr – Default load address
- filesize – Size of the loaded file
- bootargs – Boot arguments passed to Linux command line
- bootcmd – Default boot command (cfr 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 => setenv foo 1 ; setenv 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
```

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

```
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

```
1 => if true ; do echo "hello" ; else echo "bye" ; fi
2 hello
3 => false || echo "false!"
4 false!
5
6 => setenv foo 'true && "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

```
1 => for i in a b c d ; do echo "$i" ; done
2 a
3 b
4 c
5 d
```

The 'while' loop

- ▶ The while loop with a condition

```
1 => while true ; do echo hello ; done
2 hello
3 hello
4 hello
```

The 'gpio' command

- ▶ Useful for toggling/sampling GPIOs
- ▶ GPIO input sets return value

```
1 => gpio input 45
2 gpio: pin 45 (gpio 45) value is 1
3 => echo $?
4 1
5
6 => gpio set 53
7 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......
```

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 rescan
2  => mmc part
3
4  Partition Map for MMC device 0  --   Partition Type: DOS
5
6  Part   Start Sector   Num Sectors   UUID          Type
7    1    8192           6955008       1147c091-01   83 Boot
8
9  => ls mmc 0:1
10 <DIR>        4096 .
11 <DIR>        4096 ..
12               40 ID.txt
13 ...
14 => load mmc 0:1 $loadaddr ID.txt
15 => md.b $loadaddr $filesize
16 82000000: 42 65 61 67 6c 65 42 6f 61 72 ... BeagleBoard.org
17 82000010: 44 65 62 69 61 6e 20 49 6d 61 ... Debian Image 201
18 82000020: 38 2d 30 31 2d 32 38 0a         8-01-28.
```

Loading from network

- ▶ U-Boot network stack is UDP-only (no TCP)
- ▶ Support for TFTP, NFS (over UDP), DHCP/BOOTP, . . .

```
1 => setenv ethaddr 00:aa:bb:cc:dd:ee # optional!
2 => setenv ipaddr 192.168.1.300
3 => setenv netmask 255.255.255.0
4 => setenv 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 and kermit protocol

```
1 => loady
2 <send file over ymodem protocol>
```

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
- uImage
  - Legacy since forever
  - Wrapper around arbitrary binary
  - CRC32 checksum and small amount of metadata
  - Wraps single file only
  - Optional separate DT
- fitImage – multi-component image
  - Based on DT
  - Supports multiple files
  - Configurable checksum algorithm per entry
  - Supports digital signatures

Booting kernel image

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

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

Booting kernel image

```
 1 => setenv bootargs console=ttyO0,115200
 2 => load mmc 0:1 0x82000000 boot/vmlinuz-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/\
 5      am335x-pocketbeagle.dtb
 6 132769 bytes read in 180 ms (719.7 KiB/s)
 7 => bootz 0x82000000 - 0x88000000
 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
15 [    0.000000] Linux version 4.9.82-ti-r102 \
16    (root@b2-am57xx-beagle-x15-2gb) (gcc version 6.3.0 20170516
17    (Debian 6.3.0-18) ) #1 SMP PREEMPT Thu Feb 22 01:16:12 UTC 2
18 [    0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7
```

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/\
 2      am335x-pocketbeagle.dtb
 3  132769 bytes read in 180 ms (719.7 KiB/s)
 4  => fdt addr 0x88000000
 5  => fdt resize
 6  => fdt print /chosen
 7  chosen {
 8          stdout-path = "/ocp/serial@44e09000";
 9  };
10  => fdt set /chosen/ foo bar
11  => fdt print /chosen
12  chosen {
13          foo = "bar";
14          stdout-path = "/ocp/serial@44e09000";
15  };
16  => bootz 0x82000000 - 0x88000000
```

fitImage

```
 1 /dts-v1/;
 2
 3 / {
 4   description = "Linux kernel and FDT blob for sockit";
 5
 6   images {
 7     kernel@1 {
 8       description = "Linux kernel";
 9       data = /incbin/("./arch/arm/boot/zImage");
10       type = "kernel";
11       arch = "arm";
12       os = "linux";
13       compression = "none";
14       load = <0x00008000>;
15       entry = <0x00008000>;
16       hash@1 {
17         algo = "crc32";
18       };
19     };
```

fitImage

```
1    fdt@1 {
2      description = "Flattened Device Tree blob";
3      data = /incbin/("./arch/arm/boot/dts/socfpga....dtb");
4      type = "flat_dt";
5      arch = "arm";
6      compression = "none";
7      hash@1 {
8        algo = "crc32";
9      };
10   };
11  };
```

fitImage

```
1    configurations {
2      default = "conf@1";
3      conf@1 {
4        description = "Boot Linux kernel with FDT blob";
5        kernel = "kernel@1";
6        fdt = "fdt@1";
7        hash@1 {
8          algo = "crc32";
9        };
10     };
11   };
12 };
```

Compile with

```
1 mkimage -f fit-image.its fitImage
```

U-Boot sources

- Git master at:
  `http://git.denx.de/?p=u-boot.git;a=summary`
- 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 ARCH=plat                  # optional, set target architecture
4 $ export CROSS_COMPILE=plat-none-   # optional, set cross compiler
5 $ make board_defconfig              # ie. sandbox_defconfig
6 $ make
```

Partical part

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

Conveniently load custom environment

- ▶ HINT: loady and env import commands

Task 1

```
 1 linux$ cat << EOF > /tmp/env.txt
 2 > hello=world
 3 > foo=bar
 4 > EOF
 5
 6 => loady
 7 ## Ready for binary (ymodem) download to 0x82000000 at 115200 bps...
 8 C## Total Size      = 0x00000014 = 20 Bytes
 9 => md.b $loadaddr $filesize
10 82000000: 68 65 6c 6c 6f 3d 77 6f 72 6c 64 0a 66 6f 6f 3d    hello=world.foo=
11 82000010: 62 61 72 0a                                        bar.
12 => env import $loadaddr $filesize
13 ## Warning: defaulting to text format
14 => env print hello
15 hello=world
```

Task 2

Implement moving light using USR LEDs 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 2

```
1 => mw 0x4804c134 0xfe1fffff
2 => while true ; do
3         for i in 02 04 08 10 ; do
4                 mw 0x4804c13c 0x0£{i}00000 ;
5                 sleep 1 ;
6             done ;
7 done
```

Task 3

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 3

```
1 => while true ; do
2        for i in 53 54 55 56 ; do
3                gpio set $i ;
4                sleep 1 ;
5                gpio clear $i ;
6            done ;
7 done
```

Task 4

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 4

```
1  => if gpio input 45 ; then
2         echo "Button pressed" ;
3     else
4         echo "Button not pressed" ;
5     fi
```

Task 5

Barcode reader

- ▶ U-Boot queries ethernet MAC address from barcode reader, which does not necessarily use ASCII. Filter the MAC out and ignore the separators (ie. 00xaaxbbxccxddxee becomes 00:aa:bb:cc:dd:ee). Assume the list of separators is known and fixed (ie. xyz).
- ▶ HINT: env ask and setexpr

Task 5

```
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 6

Recovery system

- ► Check if USB stick is plugged in and contains kernel image and DT. If so, boot those, otherwise boot the images on SD card.
- ► HINT: usb reset, load, bootz commands

Task 6

```
1 => setenv bootargs console=ttyO0,115200
2 => usb reset && \
3     load usb 0:1 0x82000000 vmlinuz && \
4     load usb 0:1 0x88000000 am335x-pocketbeagle.dtb && \
5     bootz 0x82000000 - 0x88000000
6 resetting USB...
7 USB0:   scanning bus 0 for devices... 1 USB Device(s) found
8         scanning usb for storage devices... 1 Storage Device(s) found
9 9970640 bytes read in 6594 ms (1.4 MiB/s)
10 132769 bytes read in 123 ms (1 MiB/s)
11 ## Flattened Device Tree blob at 88000000
12    Booting using the fdt blob at 0x88000000
13    Loading Device Tree to 8ffdc000, end 8ffff6a0 ... OK
14
15 Starting kernel ...
16
17 [    0.000000] Booting Linux on physical CPU 0x0
```

Task 7

Compiling U-Boot

- ► Clone U-Boot sources, configure them, adjust bootdelay to 30 seconds, compile U-Boot and install on the board.
- ► HINT: U-Boot sources are provided on the USB stick
- ► HINT:

```
1 export ARCH=arm
2 export CROSS_COMPILE=arm-linux-gnueabi-
3 make am335x_pocketbeagle_defconfig
4 make menuconfig # locate CONFIG_BOOTDELAY
5 make
6 dd if=MLO of=/dev/sdg bs=128k count=1 seek=1
7 dd if=u-boot.img of=/dev/sdg bs=384k seek=1 count=2
```

Task 7

```
 1 $ make am335x_pocketbeagle_defconfig
 2 #
 3 # configuration written to .config
 4 #
 5
 6 $ make
 7 scripts/kconfig/conf  --silentoldconfig Kconfig
 8   CHK     include/config.h
 9   CFG     u-boot.cfg
10   GEN     include/autoconf.mk
11 ...
```

Task 8

Accelerometer

- ▶ Read out the MMA8452Q accelerometer data
- ▶ HINT: i2c commands
- ▶ HINT: Accelerometer is on bus 2
  Use i2c dev 2 to select bus 2
- ▶ HINT: Accelerometer has I2C address 0x1c
  Try: i2c md 0x1c 0 0x10
- ▶ HINT: Accelerometer is in standby, wake it up with
  i2c mw 0x1c 0x2a 0x1
  then try reading samples at offset 0x1..0x6 again

Task 9

DT

- Load DT from SD/MMC, add property to it, boot Linux and check it's presence
- HINT: fdt commands , /proc/devicetree

Thank you for your attention!

Contact: Marek Vasut <marek.vasut@gmail.com>