TXUL
U-Boot

Table of contents

1 U-Boot....4
1.1 Getting started with U-Boot....4
1.2 Terminal Setup....4
1.3 Power-On and Reset Output....4
2 Update and Recovery....5
2.1 Requirements....5
2.1.1 Terminal Connection....5
2.1.2 TFTP Server Setup....5
2.1.3 USB Connection....5
2.2 Flashing U-Boot....6
2.2.1 U-Boot Images....6
2.2.2 Recovery Boot....7
2.2.2.1 eMMC Recovery Boot....9
2.2.3 Update U-Boot....10
2.2.3.1 TFTP....10
2.2.3.2 UART....12
2.2.4 Flashing Tools....13
2.2.4.1 sbloader....13
2.2.4.2 MFGTool....14
2.2.4.3 MFGTool Profiles....15
2.3 Erase NAND memory partition....16
2.3.1 Special Partitions....16
2.3.2 Firmware Configuration Block....16
2.4 WinCE....17
2.5 Configuring the network....18
2.6 Default Environments....19
2.6.1 NAND default environment....19
2.6.2 eMMC default environment....20
2.6.3 Reset of Variables....21
3 NVM sub-system....23
3.1 NAND sub-system....24
3.1.1 Partition system....25
3.1.2 Create and Delete Partitions....27
3.1.2.1 Linux partition numbering:....28
3.1.3 Erase NAND memory partition....29
3.1.4 Bad Block Handling....29
3.2 eMMC Sub-System....30
3.2.1 Handling eMMC....30
3.2.2 Pseudo SLC Mode....33
4 Environment....34
4.1 Customization variables....35
4.1.1 boot_mode....36
4.1.2 Display....37
4.2 Splash Screen....38
4.2.1 Splash Screen Alignment....40
4.3 Logo....41
4.4 DTB....42

4.4.1 DTB Introduction....42
4.4.2 Device Tree Boot....43
4.4.3 Managing the Device Tree....44
4.4.4 Customize the Device Tree....44
4.4.5 Creating a new DTB....44
4.4.6 FDT Documentation....45
5 Building U-Boot for TXUL....46
5.1 Unpack and compile the source....46
6 Building U-Boot....48
6.1 Requirements....48
6.2 Unpacking the source....49
6.3 Compiling U-Boot....49
6.4 Cleaning the Sources....50
6.5 Object Directory....50
7 U-Boot commands....51
7.1 Overview of commands....52
7.2 Commands and explanations....53
8 Document revision history....67

1 U-Boot

The TXUL is delivered with pre-installed U-Boot firmware. U-Boot supports several low-level-debugging options and file download via Ethernet or serial X/Y/ZModem. These files can additionally be stored into the permanent flash-memory to be started by command or power-on.

1.1 Getting started with U-Boot

Please refer for a more extensive description of and encompassing view on U-Boot the online manual, available here:

http://www.denx.de/wiki/view/DULG/Manual

1.2 Terminal Setup

To connect to the TXUL CoM a terminal program like Windows *'TeraTerm'* or Unix based *'minicom'*, must be running on the host PC. The communication parameters used are:

```
Baud rate:       115 200
Data bits:       8
Parity:          None
Stop bits:       1
Flow control:    None (or Xon/Xoff)
                 (disable hardware handshake (RTS/CTS))
```

Note:
It's highly recommended to always turn on logging in the terminal program

1.3 Power-On and Reset Output

After power up or reset, output of U-Boot will appear resembling this e.g.:

```
U-Boot 2015.10-rc2-04896-g7afc505-dirty (Mar 08 2016 - 11:28:33 +0100)

CPU:   Freescale i.MX6UL rev1.0 at 396 MHz
Temperature: Industrial grade (-40C to 105C) at 25C - calibration data 0x5bb52769
Reset cause: POR
I2C:   ready
DRAM:  256 MiB
Board: Ka-Ro TXUL-0011
VDDCORE set to 1300mV
MMC:   Card did not respond to voltage select!
FSL_SDHC: 0 (eMMC), FSL_SDHC: 1
Using virtual partition dtb(1) [680..6ff]
reading logo.bmp
** logo.bmp shorter than offset + len **
Read only 308278 of 1228800 bytes
Video: 640x480x24
CPU clock set to 528.000 MHz
Baseboard: ulmb-v1
Net:   MAC addr from fuse: 00:0c:c6:7e:b4:11
FEC0, FEC1
Warning: FEC1 MAC addresses don't match:
Address in SROM is         00:0c:c6:7e:00:00
Address in environment is  00:0c:c6:7e:fd:0c

Hit any key to stop autoboot:  0 ### 0
TX6UL U-Boot >
```

2 Update and Recovery

Following are basic instructions and settings which will allow the TXUL CoM to be either restored to factory installation or updated.

The TXUL CoM specific '*Recovery Boot*' option are described under Ch. **2.2.2** (**Recovery Boot**); instructions updating U-Boot are under Ch. **2.2.3** (**Update U-Boot**).

2.1 Requirements

Windows Host:	**Terminal Connection** **TFTP Server Setup** **USB Connection**
Linux Host:	**Terminal Connection** **USB Connection**

2.1.1 Terminal Connection

- Connect UART1 (Starterkit 5 ST9) to a COM port of your host PC

For the settings please refer to Chapter **1.2** (**Terminal Setup**).

2.1.2 TFTP Server Setup

- Connect STK5 Ethernet to your network:
  - StarterKit 5v3 (STK5v3): Connect ST23
  - StarterKit 5v5 (STK5v5): Connect ST19
  - COMpact TFT (COMTFT): Connect ST62
  - TX Mainboard 7 (MB7): Connect ST4

Software download via TFTP protocol is recommended, such as because download via X/Y/ZModem protocols are slow.

Any Linux or Windows based TFTP server software may be used. Linux distributions usually already offer such software in their respective package management. For Windows the following small and simple, free and opensource software is recommended:

- TFTPD (x32 & x64)
  (also includes a DHCP, DNS, SNTP and Syslog server)
  `http://tftpd32.jounin.net/`

Please contact the specific producer for support.

2.1.3 USB Connection

Connecting a host PC and the TX CoM/Ka-Ro baseboard assembly via USB use following connectors:

- StarterKit 5v3 (STK5v3): Connect USB Mini-A/B (ST21)
- StarterKit 5v5 (STK5v5): Connect USB Mini-A/B (ST12)
- COMpact TFT (COMTFT): Connect USB Mini-A/B (ST66)
- TX Mainboard 7 (MB7): Connect USB Mini-A/B (ST4)

2.2 Flashing U-Boot

U-Boot can be reset to factory installation or updated by either of these methods: TFTP, UART, USB.

The TXUL uses the **USB** (Ch. **2.1.3**) as main '***Recovery Boot***' method.

- U-Boot functional ('***Update U-Boot***' - Ch.**2.2.3**):
  - ➔ TFTP is the fastest and recommended method for updating a live system (s. Ch. **2.2.3.1**)
  - ➔ X/YModem over UART can also be used for updating U-Boot (Ch. **2.2.3.2**).
- U-Boot non-functional ('***Recovery Boot***' - Ch. **2.2.2**):

  The **USB** can be used in either of two ways:
  - ➔ *Restore and recover* - programming the U-Boot Image to the NAND
  - ➔ *Development/Debugging* - download and run an U-Boot image in RAM

The general procedure of '***Recovery Boot***' can also be used to load a custom U-Boot while in development to be run online without changing the original flash contents (also Ch. **2.2.4**).

The TXUL relies on either of the two tools **'sbloader'** and **'MFGTool'** for upload to non-functional, e.g. bricked, TX CoM. Both tools use **USB** as upload mechanism for restore and recover and are found under the folder 'Flashtools' folder on the Starterkit CD.

The hereafter described TXUL Series CoM '***Recovery Boot***' procedure use **'sbloader'** exclusively.

The **'MFGTool'** setup the TXUL Series CoM according to a specific profile, as illustrated in Ch. **2.2.4.2**.

Please see **Flashing Tools** (Ch. **2.2.4**) for more.

2.2.1 U-Boot Images

Following a listing of the different U-Boot binary images (a.k.a. either "U-Boot image" or "U-Boot binary"). Each of the binary is specific to their respective TX CoM and variants.

Further there are general purpose U-Boot binaries which are used in conjunction with **'sbloader'** in '***Recovery Boot***', and which are written to the TX CoM. Then there also is a version of each for the **'MFGTool'**, which is the TXUL CoM's Microsoft® Windows® based image uploader and setup util.

These files (examples below) are on the StarterKit CD found under: U-Boot/target

General Version
u-boot-txul-**0010**.bin
u-boot-txul-**0011**.bin

MFGTool Version
u-boot-txul-**0010_mfg**.bin
u-boot-txul-**0011_mfg**.bin

2.2.2 Recovery Boot

Example uses TFTP (Ch.**2.1.2**, **2.2.3.1**) for UART see Ch. **2.2.3.2**

1.) **Set** (close) BOOT_MODE jumper (ST3)

2.) Connect the STK5 debug UART (Ch. **2.1.1**)

3.) Launch the terminal application (Ch. **1.2**)

4.) Connect the STK5 per Ethernet to the network (Ch. **2.1.2**)

5.) Setup a DHCP server (needed for `'bootp'`, see Ch. **2.1.2** & **2.5**)

6.) Prepare the TFTP server:
- Locate the U-Boot binary (Ch. **2.2.1** - on StarterKit CD under: `U-Boot/target`)
- Copy the file(s) into the TFTP server's data directory (usually `"/tftpboot"`)

> ***Note:***
> The U-Boot binary files for the TXUL CoM have the file extension '.bin'.

> ***Note:***
> The actual filename depends on the TXUL CoM variant in use (Ch. **2.2.1**), e.g.:
> TXUL-0010 (..._txulul-0010.bin), TXUL-0011 (..._txul-0011.bin)

7.) Connect the STK5 per USB (Ch. **2.1.3**)
- Power-up of the module
- **NO** Power-up / Reset output will appear ~~(Ch. **1.3**)~~

8.) Start U-Boot on the TXUL with `'sbloader'` (Ch. **2.2.4.1 sbloader**)

```
./sbloader-x86_32 -m -s /cdrom/U-Boot/target/u-boot.bin
```

> ***Note:***
> The program `'sbloader'` comes in a 32-bit and 64-bit flavour. To run the 32-bit on a 64-bit OS an installed `"lib32"` is required.

9.) Abort autoboot (press any key)
- Power-up / Reset output will appear (Ch. **1.3**)

10.) Recover the TXUL using following commands:

(a) Full factory recovery
Procedure: Erase **entire** NAND → Set and load U-Boot image into memory → Write it to NAND

```
nand erase.chip
setenv bootfile u-boot.bin
setenv autoload y
bootp
romupdate
```

(b) U-Boot only recovery (skip Step **16**):
Procedure: Set and load U-Boot image into memory → Write it to NAND

```
setenv bootfile u-boot.bin
setenv autoload y
bootp
romupdate
```

11.) Power down the module

12.) **Remove** (open) BOOT_MODE jumper (ST3)

13.) Re-apply power to start from flash
- Power-up / Reset output will appear (Ch. **1.3**)

14.) Abort autoboot (press any key)

15.) **If** Step **10(b))** was chosen skip Step **16**. To remove the warning below go to Step **17**

> ***Note:***
> Erasing the entire NAND Flash will cause this warning at startup:
> `"*** Warning - bad CRC or NAND, using default environment"`

16.) Save the default environment, to remedy above warning

```
saveenv
```

17.) Done.

How to setup TXUL to boot Linux from U-Boot is described in `Linux/README`

2.2.2.1 eMMC Recovery Boot

(TX6 eMMC, e.g.: TX6Q-1020)

Example uses TFTP (Ch.**2.1.2**, **2.2.3.1**) for UART see Ch. **2.2.3.2**

1.) **Set** (close) BOOT_MODE jumper (ST3)
2.) Connect the STK5 debug UART (Ch. **2.1.1**)
3.) Launch the terminal application (Ch. **1.2**)
4.) Connect the STK5 per Ethernet to the network (Ch. **2.1.2**)
5.) Setup a DHCP server (needed for **'bootp'**, see Ch. **2.1.2** & **2.5**)
6.) Prepare the TFTP server:
   - Locate the U-Boot binary (Ch. **2.2.1** - StarterKit CD under: `U-Boot/target`)
   - Copy the file(s) into the TFTP server's data directory (usually `"/tftpboot"`)

> ***Note:***
> The actual filename depends on the TXUL CoM variant in use, e.g.:
> TXUL-0010 (..._txulul-0010.bin), TXUL-0011 (..._txul-0011.bin)

7.) Connect the STK5 per USB (Ch. **2.1.3**)
   - Power-up of the module
   - **NO** Power-up / Reset output will appear ~~(Ch. **1.3**)~~

8.) Start U-Boot on the TXUL with `'sbloader'` (Ch. **2.2.4.1 sbloader**)

```
./sbloader-x86_32 -m -s /cdrom/U-Boot/target/u-boot.bin
```

> ***Note:***
> The program `'sbloader'` comes in a 32-bit and 64-bit flavour. To run the 32-bit on a 64-bit OS an installed `"lib32"` is required.

9.) Abort autoboot (press any key)
   - Power-up / Reset output will appear (Ch. **1.3**)

10.) Recover the TXUL using following commands:

Full factory recovery:

Procedure: Set and load U-Boot image into memory → Write it to eMMC

```
setenv autoload y
setenv autostart n
setenv uboot_file u-boot-txul.bin
bootp ${uboot_file}
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} ${emmc_boot_part}
mmc write ${fileaddr} 0x680 80
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} 0
```

11.) Power down the module
12.) **Remove** (open) BOOT_MODE jumper (ST3)
13.) Re-apply power to start from flash
   - Power-up / Reset output will appear (Ch. **1.3**)

14.) Abort autoboot (press any key)
15.) Done.

How to setup TXUL to boot Linux from U-Boot is described in `Linux/README`

2.2.3 Update U-Boot

2.2.3.1 TFTP

1.) Connect the STK5 debug UART (Ch. **2.1.1**)

2.) Launch the terminal application (Ch. **1.2**)

3.) Connect the STK5 per Ethernet to the network (Ch. **2.1.2**)

4.) Setup a DHCP server (needed for **'bootp'**, s. Ch. **2.1.2** & **2.5**)

5.) Prepare the TFTP server:
- Locate the U-Boot binary (Ch. **2.2.1** - on StarterKit CD under: `U-Boot/target`)
- Copy the file(s) into the TFTP server's data directory (usually `"/tftpboot"`)

> ***Note:***
> The U-Boot binary files for the TXUL CoM have the file extension '`.bin`'.

> ***Note:***
> The actual filename depends on the TXUL CoM variant in use, e.g.:
> TXUL-0010 (..._txulul-0010.bin), TXUL-0011 (..._txul-0011.bin)

6.) Power up the TXUL
- Power-up / Reset output will appear; s. Ch. **1.3**

7.) Abort autoboot (press any key)

8.) Load the U-Boot image(s) into memory and write to NAND (eMMC)

(a) manual load scenario:
```
setenv autoload no
bootp
tftp ${loadaddr} u-boot.bin
romupdate
```

(b) auto load scenario:
```
setenv autoload yes
setenv bootfile u-boot.bin
bootp
romupdate
```

(c) eMMC (TXUL w/ eMMC only)
```
setenv autoload y
setenv autostart n
setenv uboot_file u-boot.bin
bootp ${uboot_file}
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} ${emmc_boot_part}
mmc write ${fileaddr} 0x680 80
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} 0
```

9.) Power down the module

10.) Re-apply power to start from flash
- Power-up / Reset output will appear (Ch. **1.3**)

11.) Abort autoboot (press any key)

12.) Done.

13.) (Re-)Set-to and save the default environment (any custom settings will be lost) (Ch. **2.6.3**)

```
env default -a
saveenv
```

How to setup TXUL to boot Linux from U-Boot is described in `Linux/README`

2.2.3.2 UART

1.) Connect the STK5 debug UART (Ch. **2.1.1**)

2.) Launch the terminal application (Ch. **1.2**)

3.) Prepare the Y-Modem upload:
- Locate the U-Boot binary (Ch. 2.2.1) - on StarterKit CD under: `U-Boot/target`

> ***Note:***
> The U-Boot binary files for the TXUL CoM have the file extension '`.bin`'.

> ***Note:***
> The actual filename depends on the TXUL CoM variant in use, e.g.:
> TXUL-0010 (..._txulul-0010.bin), TXUL-0011 (..._txul-0011.bin)

4.) Power up the TXUL
- Power-up / Reset output will appear; s. Ch. **1.3**

5.) Abort autoboot (press any key)

6.) Load (Y-Modem) the U-Boot image and write it to NAND/eMMC:

(a) NAND:
Procedure:
Start Y-Modem loader → Load U-Boot image → Set '`romupdate`' memory address → Write image to NAND

```
loady
"CCCCCC"       → Initiate terminal application Y-Modem upload
setenv fileaddr ${loadaddr}
romupdate
```

(b) eMMC (TXUL w/ eMMC only):
Procedure:
Start Y-Modem loader → Load U-Boot image → Set memory address → Write image to eMMC

```
loady
"CCCCCC"       → Initiate terminal application Y-Modem upload
setenv fileaddr ${loadaddr}
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} ${emmc_boot_part}
mmc write ${fileaddr} 0x680 80
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} 0
```

7.) Power down the module

8.) Re-apply power to start from flash (eMMC)
- Power-up / Reset output will appear (Ch. **1.3**)

9.) Done.

10.) (Re-)Set to and save the default environment (<u>any</u> custom settings will be lost) (Ch. **2.6.3**)

```
env default -a
saveenv
```

How to setup TXUL to boot Linux from U-Boot is described in `Linux/README`

2.2.4 Flashing Tools

Short explanation of the tools used to flash the TX CoM.

2.2.4.1 sbloader

The **'sbloader'** tool is used to download and run an executable/binary image in RAM, creating the two general use cases:

- ➔ *Restore/Recover* - programming the U-Boot Image to the NAND
- ➔ *Development/Debugging* - download and run an U-Boot image in RAM

Restore and recover is described in Ch. **2.2.2**. While only the general procedure of '***Recovery Boot***' can also be applied in *Development/Debugging* to load a custom U-Boot to be run without changing the original flash contents.

There is only a binary/executable version for Linux® of **'sbloader'** included on the StarterKit CD; it's available for x86-32 (IA32) and x86-64 (AMD64/EM64T). The x86-32 version runs on both architectures (requires *'lib32'*).

It requires a USB connection. The host PC has to be connected to the various baseboards' USB port as follows (also s. Ch. **2.1.3**):

- StarterKit 5v3 (STK5v3): Connect USB Mini-A/B (ST21)
- StarterKit 5v5 (STK5v5): Connect USB Mini-A/B (ST12)
- COMpact TFT (COMTFT): Connect USB Mini-A/B (ST66)
- TX Mainboard 7 (MB7): Connect USB Mini-A/B (ST4)

After connection of the TXUL/StarterKit 5 assembly to the PC has been established issuing a command on the host PC as follows will run U-Boot on the Module, e.g. (example output - **TX6Q**):

```
$ cd /cdrom/Flashtools/Linux/sbloader
$ ./sbloader-x86_32 -v -m -s /cdrom/U-Boot/target/u-boot-tx6ul-0010.bin

Downloading 'u-boot-tx6ul-0010.bin' to 27800000
Starting code at 27800ac0
```

If the StarterKit 5 assembly is also connect via debug UART (ST9) and a terminal application is running the user will see the power-up/reset output appear (see. Ch. **1.3**).

> ***Note:***
> For Windows users to execute **'sbloader'** under a Linux® it's possible to use the ARMSDK-VM provided by Ka-Ro

> ***Note:***
> On a x86_64 Linux® system users can also use the 'sbloader-x86_64'

2.2.4.2 MFGTool

The MFGTool is a Windows solution which allows user to setup the TX-CoM based on packages as described in its config file. It requires, as `'sbloader'`, a USB connection to the TXUL/StarterKit assembly.

The MFGTool as provided by Ka-Ro is based upon the sources as published by the SoC provider NXP.

Ka-Ro provides prepared configurations and packages. Users can also create and upload custom solutions, provided editing the respective files.

The configuration of *MFGTool V2*; is done by the Windows batch file `'MfgToolSetup.bat'`, provided in the package itself. The batch file will offer a menu from which to choose the prepared configurations should be set.

MFGTool is used and setup to do a complete installation over all components:

- Bootloader
- Environment settings
- Operating system
  - Linux
    - DTB
    - Kernel
      - Kernel modules
    - rootfs
  - Windows
    - Image

By upload and exchanging the whole set of components updating can also be achieved; though it is recommended to target specific components for updating if possible.

Applying of a target operation:

- Start *MFGTool* application
- **Set** BOOT_MODE jumper (ST3)
- Connect the 5V cord power supply
- Connect USB (Ch. **2.1.3**) to a host PC USB port
- *MFGTool* shows:"*HID compliant device*"
  - Press '*Start*' button

2.2.4.3 MFGTool Profiles

The MFGTool allows to set up different profiles of software installation these are configured by the Windows batch file 'MfgToolSetup.bat', provided in the package itself

All Linux packages include a Ka-Ro Linux Kernel (4.3). The Linux Kernel provided is used in conjunction with either the Ka-Ro rootfs or with **Freescale Yocto BSP** (s. **(1)**)[sic] as rootfs. An example of this is already provided with the "TX6-LINUX-FSL-X11", which offers a hardware accelerated X11 on the TX6.

All image files for the profiles, exception being the "TX6-LINUX-FSL-X11" and "TX6-LINUX" profiles, have been left out of the MFGTool release. For information about obtaining these image please read the subsections following.

(1) Freescale Yocto BSP

The '*Freescale Yocto BSP*' [sic] can be used in combination with the Ka-Ro Linux kernel, as to provide a rootfs to the Ka-Ro Linux kernel. As most images in the MFGTool release have been left out, please acquire the respective image files at NXP:

NXP i.MX6UL Download Site[1][2]:

```
Run-time Software
     Operating System Software-Board Support Packages
          L4.1.15_1.0.0_iMX6UL
```

(2) Windows Embedded Compact

There are no Windows Images available in 2016-03 BSP

(3) Android

There are no Android Images available in 2016-03 BSP

1 registration required
2 http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx6qp/i.mx-6ultralite-processor-low-power-secure-arm-cortex-a7-core:i.MX6UL?fpsp=1&tab=Design_Tools_Tab

2.3 Erase NAND memory partition

Before (re-)writing a desired image file to the NAND memory, it is necessary to erase the corresponding flash area. Use the command **'mtdparts'** to get a list of the partitions, their names and respective memory areas.

It's output should look similar to this e.g.:

```
TX6Q U-Boot > mtdparts

device nand0 <gpmi-nand>, # parts = 7
 #: name                size            offset          mask_flags
 0: u-boot              0x00100000      0x00020000      0
 1: env                 0x00060000      0x00120000      0
 2: linux               0x00600000      0x00180000      0
 3: rootfs              0x02000000      0x00780000      0
 4: userfs              0x05780000      0x02780000      0
 5: dtb                 0x00080000      0x07f00000      0
 6: bbt                 0x00080000      0x07f80000      1

active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00020000

defaults:
mtdids  : nand0=gpmi-nand
mtdparts: mtdparts=gpmi-nand:1024k@0x20000(u-boot),384k(env),6m(linux),32m(rootfs),
 89600k(userfs),512k@0x7f00000(dtb),512k@0x7f80000(bbt)ro
```

After choosing the target partition using the command "`'nand erase.part'` *+ name*" will erase the target partition; e.g. (rootfs):

```
U-Boot > nand erase.part rootfs

NAND erase.part: device 0 offset 0x560000, size 0x1000000
Erasing at 0x1540000 -- 100% complete.
OK
```

2.3.1 Special Partitions

The TX CoM U-Boot has some special partitions that are needed for proper operation. If the user deletes or modifies these partitions - given in the list below - it will lead to failures and/or non-bootable system (Ch. **3.1** - **NAND sub-system**).

```
u-boot-spl          - size: 128 KiB         name: u-boot-spl (TX48 only)
u-boot              - size: 1 MiB           name: u-boot (U-Boot)
env                 - size: 384 KiB         name: env (The Environment)
dtb                 - size: 512 KiB         name: dtb (Device Tree Blob)
bbt                 - size: 512 KiB         name: bbt (Bad Block Table)
```

2.3.2 Firmware Configuration Block

The first flash block of the TXUL is occupied by the Firmware Configuration Block (FCB) that is required by the ROM boot code to be able to boot from flash. This block is not listed as a flash partition. For more please be refered to *Ch. 8.5.2.3* in the i.MX6 UL Applications Processor Reference Manual:

`Datasheets/i.MX6UltraLite/Data Sheet/IMX6ULIEC.pdf`

2.4 WinCE

The commands **bootce** and **nbootce** for WinCE users need to be issued differently depending if the environment variable `"autostart"` is set '*on*' or '*off*' a (Ch. **2.6**) as follows:

- set autostart **on**

  ```
  bootce -i
  nbootce
  ```

- set autostart **off**

  ```
  bootce -i;go
  nbootce;go
  ```

2.5 Configuring the network

To manually configure the network via DHCP or BOOTP use the command '**bootp**' as follows:

Prevent automatic loading of image files:

```
setenv autoload no
```

Configure network:

```
bootp
```

> ***Note:***
> Abort of the '**bootp**' procedure, called manually or by autoload, by user input or error like *'file not found'* will reverted the IP settings.

Successful execution of the commands yields an output message similar to the following example:

```
TX48 U-Boot > bootp
BOOTP broadcast 1
link up on port 0, speed 100, full duplex
DHCP client bound to address 192.168.100.159
Using cpsw device
TFTP from server 192.168.200.1; our IP address is 192.168.100.159;
 sending through gateway 192.168.100.1
```

> ***Note:***
> Above serves as a representative example and is subject to change.
> All TX Series Modules with U-Boot will have an output similar to the above.

If 'autoload' is unset (or evaluates to '*true*') U-Boot will try to load the file specified in the variable "bootfile" (Ch. **2.6**) from the default TFTP server as determined from the variable "serverip" or via **bootp**/**dhcp** options.

The variable "serverip" is set automatically to the BOOTP/DHCP server by the '**bootp**'/'**dhcp**' command.

If the file cannot be found **or** the **bootp**/**dhcp** command is aborted with CTRL-C, the network is left unconfigured.

To configure the Ethernet manually, please see Chapter **5.10** of the U-Boot documentation:

Denx.de - 5.10. U-Boot Environment Variables[1]

1 http://www.denx.de/wiki/view/DULG/UBootEnvVariables

2.6 Default Environments

Hereafter are the listings of the default environments of the TXUL Series CoM

2.6.1 NAND default environment

For TXUL w/ NAND only:

(For eMMC TXUL environment please see Ch. **2.6.2**)

The following list shows the status of the environment variables after factory installation:

```
TX6S U-Boot > pr
autoload=no
autostart=no
baseboard=stk5-v3
baudrate=115200
boot_mode=nand
bootargs=init=/linuxrc console=ttymxc0,115200 ro debug panic=1
bootargs_jffs2=run default_bootargs;setenv bootargs ${bootargs} root=/dev/mtdblock3
 rootfstype=jffs2
bootargs_mmc=run default_bootargs;setenv bootargs ${bootargs} root=/dev/mmcblk0p2 rootwait
bootargs_nfs=run default_bootargs;setenv bootargs ${bootargs} root=/dev/nfs nfsroot=$
 {nfs_server}:${nfsroot},nolock ip=dhcp
bootargs_ubifs=run default_bootargs;setenv bootargs ${bootargs} ubi.mtd=rootfs root=ubi0:rootfs
 rootfstype=ubifs
bootcmd=run bootcmd_${boot_mode} bootm_cmd
bootcmd_jffs2=setenv autostart no;run bootargs_jffs2;nboot linux
bootcmd_mmc=setenv autostart no;run bootargs_mmc;fatload mmc 0 ${loadaddr} uImage
bootcmd_nand=setenv autostart no;run bootargs_ubifs;nboot linux
bootcmd_net=setenv autoload y;setenv autostart n;run bootargs_nfs;dhcp
bootdelay=1
bootfile=uImage
bootm_cmd=bootm ${loadaddr} - ${fdtaddr}
cpu_clk=792
default_bootargs=setenv bootargs init=/linuxrc console=ttymxc0,115200 ro debug panic=1 $
 {append_bootargs}
fdtaddr=11000000
fdtsave=fdt resize;nand erase.part dtb;nand write ${fdtaddr} dtb ${fdtsize}
loadaddr=18000000
mtdids=nand0=gpmi-nand
mtdparts=mtdparts=gpmi-nand:1024k@0x20000(u-
 boot),384k(env),6m(linux),32m(rootfs),89600k(userfs),512k@0x7f00000(dtb),512k@0x7f80000(bbt)ro
nfsroot=/tftpboot/rootfs
otg_mode=device
touchpanel=tsc2007
video_mode=VGA

Environment size: 1485/131068 bytes
TX6S U-Boot >
```

2.6.2 eMMC default environment

For TXUL w/ eMMC only:

The following list shows the status of the environment variables after factory installation:

```
TX6UL U-Boot > pr
autoload=no
autostart=yes
baseboard=ulmb-v1
baudrate=115200
boot_mode=mmc
bootargs_jffs2=run default_bootargs;setenv bootargs ${bootargs} root=/dev/mtdblock3
 rootfstype=jffs2
bootargs_mmc=run default_bootargs;setenv bootargs ${bootargs} root=PARTUUID=${rootpart_uuid}
 rootwait
bootargs_nfs=run default_bootargs;setenv bootargs ${bootargs} root=/dev/nfs nfsroot=$
 {nfs_server}:${nfsroot},nolock ip=dhcp
bootargs_ubifs=run default_bootargs;setenv bootargs ${bootargs} ubi.mtd=rootfs root=ubi0:rootfs
 rootfstype=ubifs
bootcmd=run bootcmd_${boot_mode} bootm_cmd
bootcmd_jffs2=setenv autostart no;run bootargs_jffs2;nboot linux
bootcmd_mmc=setenv autostart no;run bootargs_mmc;fatload mmc 0 ${loadaddr} uImage
bootcmd_net=setenv autoload y;setenv autostart n;run bootargs_nfs;dhcp
bootdelay=0
bootfile=uImage
bootm_cmd=bootm ${loadaddr} - ${fdtaddr}
cpu_clk=528
default_bootargs=setenv bootargs init=/linuxrc console=ttymxc0,115200 ro debug panic=1 $
 {append_bootargs}
emmc_boot_ack=1
emmc_boot_part=1
eth1addr=00:0C:C6:7E:FD:0C
ethact=FEC0
ethaddr=00:0c:c6:7e:b4:11
fdtaddr=81000000
fdtsave=mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} ${emmc_boot_part};mmc write ${fdtaddr}
 0x680 80;mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} 0
fdtsize=86f7
loadaddr=82000000
nfsroot=/tftpboot/rootfs
otg_mode=device
rootpart_uuid=0cc66cc0-02
splashimage=82000000
stderr=serial
stdin=serial
stdout=serial
touchpanel=edt-ft5x06
ver=U-Boot 2015.10-rc2-04896-g7afc505-dirty (Mar 08 2016 - 11:28:33 +0100)
video_mode=VGA

Environment size: 1582/131068 bytes
TX6UL U-Boot >
```

2.6.3 Reset of Variables

Important:

It's is highly recommended for users to use this feature with each update to assure popper function of U-Boot and Linux releases, as well as for adoption of new feature-sets, e.g. support of UBIFS or DTB.

Users can reset the environment as a whole, as describes in Ch. **2.2.2** (Step **13**), or can reset just specific variables. Selectively resetting variables can be used to access variables or environment settings that have been updated or added between releases.

To unset an environment variable the same common procedure as setting is used; but instead setting it to an empty string.

As long as the command `'save'` (abbr.: `'saveenv'` - all U-Boot commands can be abbreviated, s. Ch. **7**) is not executed these changes will be lost by the next reset. In development this should be the preferred method.

Examples:

- Unset variable

  e.g. "*unset* `'apppend_bootargs'`":

  ```
  setenv apppend_bootargs
  save
  ```

- Whole environment (reset)

  ```
  env default -a
  saveenv
  ```

- Specific variable (reset)

  ```
  env default <variable>
  saveenv
  ```

  e.g. "*resetting* `'fdtsave'`":

  ```
  env default fdtsave
  save
  ```

<u>Important:</u>

Hereafter displayed are <u>exemplary</u> outputs and settings
these serve as <u>reference</u> and might not reflect actual data.

3 NVM sub-system

U-Boot is capable and supports booting from multiple non-volatile memory (NVM) sub-systems. While the Ka-Ro TX Series CoM are generally equipped with "on module" NAND this is therefore the primary boot medium from which U-Boot will be loaded and booted from after power-on reset.

With the advent of the TX6 and the integration of eMMC onto the TX6 CoM series modules U-Boot on these specific modules is loaded and booted from the eMMC on power-on reset.

Any further booting on either NAND or eMMC TX CoM will require scripting in the U-Boot environment, e.g. via the given pre-defined **boot_mode** variable [recommended] (see **Customization variables**, Ch. **4.1**).

For a more detailed view upon each non-volatile memory sub-system please read each respective chapter concerning the technology:

- **NAND sub-system**, Ch. **3.1**
- **eMMC Sub-System**, Ch. **3.2**

3.1 NAND sub-system

Following sub-chapters describe points of interest for the user when working with the U-Boot boot-loader concerning writing of data images (rootfs, etc.) or customization of NAND settings and/or the operating system.

Important:

U-Boot depends on following partitions to be available in NAND, these partitions may not be removed:

```
u-boot-spl  -     Function: U-Boot SPL (TX48 special)
u-boot      -     Function: U-Boot
env         -     Function: Environment
bbt         -     Function: Bad Block Table
```

Linux and Windows users will need the `'dtb'` partition for Logo/Splash screen functionality, recommendation is to keep it's default values:

```
dtb         -     Function: Device-Tree-Blob
```

Linux users also will need the `'dtb'` partition for general graphics output and kernel booting.

3.1.1 Partition system

For the ease of use for the user U-Boot supports a partitioning system, which allows the user to program and write data to the NAND while using short and memorable names for parts of the NAND. U-Boot itself internally **does not** work with **partitions**. These partitions just represent easy memorable and usable aliases to the user for the NAND addresses tagged by these. Therefore this system behaves unlike any encountered on a PC and/or HDD system. Which can lead, if applied fallacious, to undesired effects, such as describe in **Bad Block Handling** (Ch. **3.1.4**).

Data concerning the partitions, like size, offset in the NAND, name, etc., are saved and set in the environment variable called "`mtdparts`", for notation please see example below. User interaction with the partition system therefore can be by either of two ways:

`mtdparts` - command (see Ch. **7.2** and **3.1.2**)
`setenv mtdparts` - (setting) the variable (use '`printenv` *`[mtdparts]`*' to view)

> ***Note:***
> Changing the offset or the size of the partitions '`u-boot`', '`env`' and '`bbt`' will have undesired effects, since these values are hardcoded in the U-Boot source.

By using the command '**`mtdparts`**' the user can modify the partitions as defined in environment variable, i.e. changing the order, name and/or size, etc., using the commands respective options. The command also will do sanity checks on the users input and as such it's use is highly recommended. Advanced users can also edit the "`mtdparts`" variable itself.

Changing the partition table will not alter the flash contents, so that data contained in affected partitions may (partially) show up in different partitions after the change. Refer to **Create and Delete Partitions** (Ch. **3.1.2**), **Erase NAND memory partition** (Ch. **3.1.3**) and **Splash Screen** (Ch. **4.2**) for more.

Following an exemplary output **[defaults]** of both *parts* of `'mtdparts'`:

- Notation:

```
nand.0                 - NAND short name used with command "mtdparts"
omap2-nand.0           - NAND driver (dependent on used TX CoM)
128k(u-boot-spl)       - size: 128 KiB      name: u-boot-spl (TX48 special)
1m(u-boot)             - size: 1 MiB        name: u-boot
0x60000(env)           - size: 0x60000 Byte name: env (the environment)
4m(linux)              - size: 4 MiB        name: linux
16m(rootfs)            - size: 16 MiB       name: rootfs
108416k(userfs)        - size: 108416k      name: userfs
256k(dtb)              - size: 256KiB       name: dtb (Device-Tree-Blob)
                                            (needed for 'video_mode' settings)
bbt                    - size: 512KiB       name: bbt (Bad Block Table)
                                            (DO NOT delete or modify)
```

- Environment variable:

```
TX6DL U-Boot > print mtdparts
mtdparts=mtdparts=gpmi-nand:1024k@0x20000(u-boot),384k(env),4m(linux),32m(rootf
 s),91648k(userfs),512k@0x7f00000(dtb),512k@0x7f80000(bbt)ro
```

- Command:

```
TX6DL U-Boot > mtdparts

device nand0 <gpmi-nand>, # parts = 7
 #: name                size            offset          mask_flags
 0: u-boot              0x00100000      0x00020000      0
 1: env                 0x00060000      0x00120000      0
 2: linux               0x00400000      0x00180000      0
 3: rootfs              0x02000000      0x00580000      0
 4: userfs              0x05980000      0x02580000      0
 5: dtb                 0x00080000      0x07f00000      0
 6: bbt                 0x00080000      0x07f80000      1

active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00020000

defaults:
mtdids  : nand0=gpmi-nand
mtdparts: mtdparts=gpmi-nand:1024k@0x20000(u-boot),384k(env),4m(linux),32m(root
 fs),91648k(userfs),512k@0x7f00000(dtb),512k@0x7f80000(bbt)ro
```

Note:
The output of `"mtdparts:"` under `"defaults:"` are the compiled in defaults.

3.1.2 Create and Delete Partitions

As described above in Chapter **3.1.1** (**Partition system**), U-Boot does only work with the "`mtdparts`" variable. Adhering to the notation of "`mtdparts`" (see previous chapter) subsequently an illustration, by a partition added "in between", on how to create and delete partitions. Example:

- Current '`mtdparts`' values:

```
TX53 U-Boot > print mtdparts
mtdparts=mtdparts=mxc_nand:1m(u-boot),0x60000(env),4m(linux),16m(rootfs),108416
 k(userfs),256k(dtb),512k@0x7f80000(bbt)ro

TX53 U-Boot > mtdparts

device nand0 <mxc_nand>, # parts = 7
 #: name                size        offset          mask_flags
 0: u-boot              0x00100000    0x00000000 0
 1: env                 0x00060000    0x00100000 0
 2: linux               0x00400000    0x00160000 0
 3: rootfs              0x01000000    0x00560000 0
 4: userfs              0x069e0000    0x01560000 0
 5: dtb                 0x00040000    0x07f40000 0
 6: bbt                 0x00080000    0x07f80000 1

active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00000000

defaults:
mtdids  : nand0=mxc_nand
mtdparts: mtdparts=mxc_nand:1m(u-boot),0x60000(env),4m(linux),16m(rootfs),10841
 6k(userfs),256k(dtb),512k@0x7f80000(bbt)ro
```

- Edit partitioning
  (Procedure: *del* '`userfs`', *add* *resized* '`userfs`', *add* '`logo.bmp`', size: 3 MiB; [between '`userfs`' and '`dtb`']; *save*)

```
TX53 U-Boot > mtdparts del userfs
TX53 U-Boot > mtdparts add nand0 105344k@0x01560000 userfs
TX53 U-Boot > mtdparts add nand0 3072k@0x7C40000 logo.bmp
TX53 U-Boot > save
```

  The creation of the partitions needs to be done with an offset (see above '`mtdparts`') to be successful. Otherwise there will be errors, e.g.:

```
TX53 U-Boot > mtdparts add nand0,4 3m test
mxc_nand: partitioning exceeds flash size
```

Until the user uses the **'`saveenv`'** command the changes to the "`mtdparts`" variable are only in memory and volatile and will be lost at reset – also see Ch. **2.6.3** (**Reset of Variables**).

Data written in the meantime can only be accessed via specifying the offset (as defined before reset).

Note: The smallest size a partition can be is the size of an erase block, which depends on the NAND chip used (usually 128KiB)
Note: As the '**`mtdparts`**' command can only calculate and set the remaining size using the "`-`" (dash) with no tailing partition(s), the '`bbt`' prevents this.

- Confirm new partitioning

```
TX53 U-Boot > mtdparts

device nand0 <mxc_nand>, # parts = 8
 #: name    size  offset     mask_flags
 0: u-boot               0x00100000     0x00000000 0
 1: env                  0x00060000     0x00100000 0
 2: linux                0x00400000     0x00160000 0
 3: rootfs               0x01000000     0x00560000 0
 4: userfs               0x066e0000     0x01560000 0
 5: logo.bmp             0x00300000     0x07c40000 0
 6: dtb                  0x00040000     0x07f40000 0
 7: bbt                  0x00080000     0x07f80000 1

active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00000000

defaults:
mtdids  : nand0=mxc_nand
mtdparts: mtdparts=mxc_nand:1m(u-boot),0x60000(env),4m(linux),16m(rootfs),10841
 6k(userfs),256k(dtb),512k@0x7f80000(bbt)ro

TX53 U-Boot > print mtdparts
mtdparts=mtdparts=mxc_nand:1m(u-boot),0x60000(env),4m(linux),16m(rootfs),105344
 k(userfs),3m(logo.bmp),256k(dtb),512k@0x7f80000(bbt)ro
```

Important:

With the exceptions of `'u-boot'`, `'env'` and `'bbt'` (and TX48's: `'u-boot-spl'`) all partitions are moveable. Mentioned exceptions are U-Boot relevant and their offsets and sizes are defined at compile time; any manual changes in `"mtdparts"` will lead to defects or a non-bootable system.

Important:

The `'dtb'` partition is moveable but recommendation is to keep the default values given. As a DTB is needed to provide a FDT structure from which U-Boot determines the graphics settings for itself and its logo/splash screen functionality and Linux graphical output.

3.1.2.1 Linux partition numbering:

Adding or deleting of one or more partitions will lead to a changed partition table for the Linux BSP. In case of above given example the previous assignment `"/dev/mtdblock5"` for `'userfs'` would then be the `"/dev/mtdblock6"`. This has to be accounted for in the U-Boot environment variables, as well as in the Linux system.

Note:
As the `'mtdparts'` command can only calculate and set the remaining size using the `"-"` (dash) with no tailing partition(s), the `'bbt'` prevents this.

Note:
The smallest size a partition can be is the size of an erase block, which depends on the NAND chip used (usually 128KiB)

3.1.3 Erase NAND memory partition

Before (re-)writing a desired image file to the NAND memory, it is necessary to erase the corresponding area. Using the command '**mtdparts**' allows to get a list of the partitions, their names and their respective memory areas (see Ch. **3.1.1**).

After choosing the target partition the using the command "'**nand** erase.part' + *name*" will erase the target partition; e.g. (rootfs):

```
U-Boot > nand erase.part rootfs
NAND erase: device 0 offset 0x540000, size 0x1000000
Erasing at 0x1520000 -- 100% complete.
OK
```

3.1.4 Bad Block Handling

As of U-Boot version v2013.07 (source base), U-Boot now recognizes partition boundaries when writing or reading data to/from a flash partition. The amount of data actually written or read is automatically adjusted by the size of the bad blocks within the partition. This solves that trailing data will **not** spill over into succeeding partitions anymore.

This is **not** valid in case writes are done to specific addresses to NAND rather than a partition name as this is not checking whether data read or written actually fits a potential partition it might affect, as in e.g.:

```
nand write.trimffs ${fileaddr} rootfs ${filesize}
```

vs.

```
nand write.trimffs ${fileaddr} 0x00580000 ${filesize}
```

Since NAND flash may contain bad (i.e. unusable) blocks, the actual usable space of a partition may be smaller than the size given in the "mtdparts" (see Ch. **3.1.1**) variable. The NAND driver will skip those bad blocks when reading or writing data, but, depending the command used, can still read/write the predetermined amount of data.

Therefore users always should take precautions when writing data to NAND flash, like:

- While creating a partition, the size should be chosen one or more erase blocks (128 KiB) larger than the designated data size.
- When writing data to a partition the actual size of the data should be explicitly specified; e.g. by passing '${filesize}' as argument to the '**nand** write' command.
- The output of '**nand** write' and/or '**nand** erase' should be checked for messages identifying bad blocks: "Skipping bad block ..." and verified, that the partition size excluding those blocks is still sufficient to keep all the data.

Note:
In case of address based writes trailing data still
spill over into the succeeding partition without notice!

3.2 eMMC Sub-System

With the introduction eMMC to the TXUL CoM series, instead of NAND as non-volatile memory, the procedures for recovery, update and partitioning have seen some changes and adaptations to the NVM in question.

In contrast to NAND is eMMC not a unified device. It is split into two distinct parts, being the "*protected boot*" and "*general purpose*" MMC device.

The "*protected boot*" partition is where U-Boot resides as well as the DTB and the environment, which in NAND are distinct partitions. Further there is no BBT partition as **Bad Block Handling** is controlled by the eMMC chip itself.

3.2.1 Handling eMMC

Important:

To install a Linux rootfs or WinCE onto the eMMC the user will have to partition and format the eMMC as would be done with a general plug-in SD-Card.

It is recommended to use MFGTool (Ch. **2.2.4.2**) for this procedure.

Changes were introduced to the `'mmc'` command of das U-Boot. These new `'mmc'` subcommands are the following, e.g. (usage):

```
mmc partconf 0 1 1 0
mmc bootbus 0 1 0 2
```

These commands change the way how users access the boot behaviour of the eMMC chip and thus TX CoM.

Changes executed by these commands will remain valid independent of and over power cycling, e.g. POR, on/off, etc.).

For all (sub-)commands apples that the first (1st) parameter always represents the mmc-device number – which is `'0'` for the soldered eMMC chip. Therefore hereafter applies:

- The first (1st) parameter represents the mmc-device number – which is `'0'` for the soldered eMMC chip

1. **`mmc partconf`**

   - The second (2nd) parameter specifies if at boot a BOOT_ACK is either send or not, this has no effect on i.MX6 and thus can always be on/true.
   - The third (3rd) parameter specifies the boot partition setting:

     ```
     0 - eMMC-Boot disabled
     1 - boot from eMMC bootpart 1
     2 - boot from eMMC bootpart 2
     7 - boot from eMMC user partition
     ```

   - The 4th parameter selects which partition is going to be used for read/write:

     ```
     0 - No access to boot partition
     1 - access boot partition 1
     2 - access boot partition 2
     3 - access rpmb partition
     4..7 access general purpose partition 1..4
     ```
     (not available on TX CoM eMMC chip)

To allow a deterministic usage of this feature set and access to the boot partition their respective parameters `'BOOT_ACK'` and `'BOOT_PART'` have been introduced to the U-Boot environment as variables that are set via:

`'emmc_boot_ack'`

`'emmc_boot_part'`

and used in e.g. `'fdtsave'` command, e.g.:

```
mmc partconf 0 ${emmc_boot_ack} ${emmc_boot_part} 1;mmc write
  ${fdtaddr} 0x1e80 80;mmc partconf 0 ${emmc_boot_ack}
  ${emmc_boot_part} 0
```

Via `'mmc partconf'` it is also possible to very easily switch between two (2) boot images, e.g.:

`mmc partconf 0 1 1 0` -> select boot partition 1
`mmc partconf 0 1 2 0` -> select boot partition 2

this generally also allows to choose a user partition to boot from, e.g.:

```
mmc partconf 0 1 7 0
```

2. **mmc bootbus**

- The second (2nd) parameter sets the boot time bus width

```
0 - x1 (sdr) or x4 (ddr) bus width in boot operation mode
1 - x4 (sdr/ddr) bus width in boot operation mode
2 - x8 (sdr/ddr) bus width in boot operation mode
```

- The third (3rd) parameter indicates if the boot settings should retain after the boot operation:

```
0 - Reset bus width to x1, single data rate and backward compatible
    timings after boot operation
1 - Retain boot bus width and boot mode after boot operation
```

- The fourth (4th) parameter sets `'BOOT_MODE'`:

```
0 - Use single data rate + backward compatible timings in boot
    operation
1 - Use single data rate + high speed timings in boot operation
    mode
2 - Use dual data rate in boot operation
```

3.2.2 Pseudo SLC Mode

Important:

Pseudo SLC (pSLC) activation will lead to data loss on the respective eMMC partition.

The eMMC can be used in the so called pseudo SLC (pSLC) mode, which allows, with e.g. slower cell degradation, for longer program/erase endurance.

To activate and use this mode the Ka-Ro Linux rootfs has the so called `'mmc-utils'` tool integrated.

For more about pSLC please be referred to[1]:

`Electronicsdesign.com – Toshiba September FAQ`

1 http://electronicdesign.com/site-files/electronicdesign.com/files/uploads/2013/09/FAQs-Toshiba-September.pdf

4 Environment

U-Boot has multiple extensions to it's general environment as a boot-loader. Ka-Ro offers in the standard configuration of it's U-Boot following features:

- **Customization variables**, Ch. **4.1**
- **Splash Screen**, Ch. **4.2**
- **Logo**, Ch. **4.3**
- **DTB**, Ch. **4.4**

Description of these features is also available under (StarterKit CD) `'U-Boot/README'` for each TX-CoM as well, general information in the extracted U-Boot sources under:

`[`*`U-Boot_src/`*`]doc/README.KARO-???`

Important:

A customization may render a board unbootable or unreactive, i.e. hang on boot or watchdog reset during start-up.

Therefore variables, like `"cpu_clk"` or `"splashimage"`, will not be evaluated at start-up in case a *watchdog reset* occurred.

Pressing *CTRL-C* on the UART console also allows the users to manually recover from this situation and correct the faulty setting(s).

Pressing any key (a character is available) on UART console will abort the autostart.

Pressing *CTRL-C* or any key while in the bootdelay phase will only abort the autostart.

Important:

Pressing any key (sending any character) on UART console will abort the autostart (automatic start of the operating system after the '`bootdelay`').

4.1 Customization variables

Following variables let the user set up specific customizations built-in to U-Boot, the release DTB and Linux Kernel (see also **Ch. 4.4.2 - Device Tree Boot**, `Linux/TXUL-Driver.pdf` and `U-Boot/FDT-Quickreference.pdf`):

`boot_mode`
`[jffs2|mmc|nand|net]`
`boot_mode` is an enhancement of the before used `bootcmd_`*`xyz`*
`(see '[U-Boot_src/]doc/README.KARO-TXUL')`
Details see below at Ch. **4.1.1** - **boot_mode**

`cpu_clk`
`<CPU freq [MHz]>`

`touchpanel`
`[tsc2007|edt-ft5x06|egalax_ts]`
`(Linux/TXUL-Driver.pdf)`

`otg_mode`
`[host|device|none] (Linux/TXUL-Driver.pdf)`

`video_mode`
`<name of one of the supported display configurations from the DT>`
e.g.: ET0570
for more see:
Ch. **4.1.2** - **Display**
`Linux/TXUL-Driver.pdf`
`U-Boot/FDT-Quickreference.pdf`

`baseboard`
`[stk5-v3|stk5-v5|<none|non-stk5-value>]`
selects type of baseboard `'stk5-v5'` setting enables CAN transceiver switch on `GPIO4_21` and disables USB Host mode on `USBOTG` port. Strings not starting in `'stk5'` prevent the STK5 specific pad initialization to be done.
`'none'` (N/A) or any other `'non-stk5-value'` entries will skip the boardsetup procedure.
Beside of booting this behaviour can be manually forced using the command: `'fdt boardsetup'`

`splashimage`
Ch. **4.2** - **Splash Screen**
Either:
Memory address (e.g. `${loadaddr}`) of a BMP file to be displayed instead of the built-in logo. Since NAND flash is not accessible in a memory mapped fashion, U-Boot will try to load the contents of the flash partition `'logo.bmp'` to the address given with `'splashimage'`.
Or:
The name of an MTD partition, that contains a raw dump of the frame buffer contents which will be loaded to the framebuffer.

`splashpos`
(when `'splashimage'` contains a memory address) the position (`'x,y'`) on the screen at which the BMP image will be displayed.
Setting splashpos to `'m,m'` will center (median) the image on the screen. (Ch. **4.2.1** - **Splash Screen Alignment**)

4.1.1 boot_mode

This selects which of the (default) boot scripts will be used by `'bootcmd'` to boot the application (Linux). The supported values are:

`nand`	[***NAND: default***] load kernel from NAND partition `'linux'` and mount rootfs (fstype UBIFS) from partition `'rootfs'`.
`mmc`	[***eMMC: default***] load kernel from file `'uImage'` on first partition (FAT) on (first) SD/MMC card device and mount rootfs (fstype autodetected) from second partition. TX6Q-1020: The first (default) SD/MMC card is the eMMC.
`net`	load kernel image via tftp (file uImage) and mount rootfs via NFS. This requires the additional variables `'nfsroot'` (path to rootfs on NFS server) and `'nfs_server'` (hostname or IP address of NFS server) to be set.
`jffs2`	[***legacy***] load kernel from NAND partition `'linux'` and mount rootfs (fstype JFFS2) from partition `'rootfs'`.

Important:

The JFFS2 file system is no longer supported and the boot mode option only remains as legacy support.

Users of JFFS2 are advised to upgrade to UBIFS.

4.1.2 Display

The environment variable **'video_mode'** is evaluated for (pre-)defined values in the **FDT** structure and has the double-acting function of managing:

- U-Boot attached display for logo or splash screen output
- Linux Kernel command line for graphical output

The DTB as provided by Ka-Ro has already defined values that the user can see using the command below:

```
fdt print display
```

Setting **'video_mode'** to one of the there given values will show the compiled-in logo and enable users to show their own splash image as described here:

U-Boot/TX##_U-Boot.pdf
Ch. **4.2** - **Splash Screen**

Setting the environment variable (permanently) is done by the commands (e.g. ET0700):

```
setenv video_mode ET0700
save
```

Following a short list of available (pre-defined) modes:
(The list follows displays as given under "/Datasheets/Display/*.pdf")

video_mode **value**	**Display**	**Resolution**	**Initials**
LCD			
ET0350	ET0350G0DH6 / ET0350G2DH6	480x272	HVGA
ET0430	ET0430G0DH6 / ET0430G2DH6	480x272	HVGA
ETQ570	ETQ570G0DH6 / ETQ570G2DH6	320x240	QVGA
ETV570	ETV570G0DH6 / ETV570G2DH6	640x480	VGA
ET0500	ET0500G0DH6 / ET0500G2DH6	800x480	WVGA
ET0700	ET0700G0DH6 / ET0700G2DH6	800x480	WVGA
VGA	standard VGA configuration	640x480	VGA
LVDS[1)]			
hsd100pxn1	HSD100PXN1	1024x768	XGA
CoMTFT[2)3)]			
COMTFT	ET070001DM6 (CoMTFT)	800x480	WVGA

Users who want to add a configuration for a custom display please refer to:

U-Boot/FDT-Quickreference.pdf
Ch. 3.1 ff.

1) Only LVDS version of: TX53, TX6
2) Only TX6
3) CoMTFT platform is compatible with non-TX6 CoM, unsupport

4.2 Splash Screen

In the factory configuration U-Boot displays a boot logo which is compiled-in (see Ch.**4.3** - **Logo**). This can be overridden by a boot splash screen/splash image being defined which is loaded from a NAND or eMMC partition. In the standard Ka-Ro release installation the boot splash screen/splash image feature is built-in, but not active.

Requirements for an active splash screen:

➔ Image:

- File Format: BMP
- Color Format[1]: 32 bpp XRGB
  8 bpp indexed colors
  24 bpp & 16 bpp
- Resolution scaled to display
  - no scaling supported[2][3]
- Resolution maximum depends on the capabilities of TX CoM.

Note:
Any unused space around the picture will be set to a white background.

➔ TX CoM (NAND or eMMC):

- Setup of DTB (Ch. **4.4**) and U-Boot **`'video_mode'`**
  `Linux/TXUL-Driver.pdf` - Ch. 6
  `U-Boot/FDT-Quickreference.pdf'` – Ch. 3 ff.
- Logo partition creation on the TX CoM (`"logo.bmp"`)

Note:
Only TX CoM using NAND, e.g. TXULQ-10.

- Setup of U-Boot environment (activation)
- Upload of BMP into the NAND/eMMC of the TX CoM

Between a TX CoM using NAND and one using eMMC users have to follow slightly different procedures:

➔ NAND
As for the splash screen to be loaded at boot time from the NAND the user first has to create a partition for the splash screen image, named `"logo.bmp"`(Ch. **3.1.2** - **Create and Delete Partitions**).

➔ eMMC
Using eMMC the splash screen will have to be uploaded into the first FAT partition for U-Boot to load the uploaded image file.

After setting up the U-Boot environment for display output (see Ch. **4.1** - **video_mode**), the splash screen can be loaded into the NAND or eMMC like any other file (see e.g.: Ch. **2.2.3.1,** Step **8(a)**).

1 Supported by e.g.: GIMP (GNU Image Manipulation Program)
2 Oversized images will be clipped
3 Undersized images will have a white border

Following shows examples how to partition (NAND), activate, position (optional, Ch. **4.2.1**) and upload via TFTP the splash screen on a TX CoM.

> ***Note:***
> Example setup assume a symmetric positioning of the shown image, see Ch.**4.2.1** - **Splash Screen Alignment**

Examples:

a) Setup NAND:

Assumption: 1024x768x32 + 2 erase blocks spare = 3328 KiB (see Ch. **3.1.4**)

```
mtdparts del userfs
mtdparts add nand0 105472k@0x01560000 userfs
mtdparts add nand0 3328k@0x7C40000 logo.bmp
setenv splashimage ${loadaddr}
setenv splashpos m,m
save
```

```
setenv autoload no
bootp
tftp ${loadaddr} [ip:]path/to/logofile.bmp
nand erase.part logo.bmp
nand write ${fileaddr} logo.bmp ${filesize}
```

> ***Note:***
> The U-Boot setup is done before loading the image, as to not save information from "bootp" into the environment.

b) Setup eMMC:

```
setenv splashimage ${loadaddr}
setenv splashpos m,m
save
```

```
setenv autoload no
bootp
tftp ${loadaddr} [ip:]path/to/logofile.bmp
fatwrite mmc 0 1 ${fileaddr} logofile.bmp ${filesize}
```

> ***Note:***
> The U-Boot setup is done before loading the image, as to not save information from "bootp" into the environment.

4.2.1 Splash Screen Alignment

It is possible to adjust the position of the splash screen, loaded by U-Boot either from the NAND `'logo.bmp'` or eMMC partition, on the screen. Setting following environment variable adjusts shifts the position as defined by the values given for width and height (e.g. to upper left corner - zero shift [*default setting*]):

```
setenv splashpos 0,0
save
```

If the variable is not set, i.e. missing, a default value of `"0,0"` is applied to the splash screen image. Setting this variable to a value `'!=0'`, i.e. non-zero value, it will move the splash screen image in respect to the default position, the upper left corner; the compiled-in logo is not affected by this.

The variable has the special value `"m"`, allowing U-Boot to automatically centre (median) the splash screen image.

Examples (also see `README`):

`setenv splashpos x,y`	- Set logo position to: 'x pixels, y lines' (width, height)
`setenv splashpos 10,m`	- Set logo position to: '10 pixels, centred height (median)'
`setenv splashpos m,10`	- Set logo position to: 'centred width, 10 lines'
`setenv splashpos m,m`	- Set logo position to: 'centred on screen (median)'

Note:
Negative values will set the logo position in respect to the lower right corner.
Example:`'setenv splashpos -1,-1'` *or* `'setenv splashpos -120,m'`*, etc.*

4.3 Logo

The customization of the logo involves a U-Boot recompile. To compile in a logo the requirements for building U-Boot have to be satisfied, as stated in Ch. **6** (**Building U-Boot**).

The intended logo file has the following requirements:

- ➔ Image
  - File Format: BMP
  - Color Format[1]: 8 bpp indexed colors
  - Size[2]: ≤ 64KiB
- ➔ TX CoM (NAND or eMMC):
  - Setup of **DTB** (Ch. **4.4**) and U-Boot `'video_mode'`
    `Linux/TXUL-Driver.pdf` - Ch. 6
    `U-Boot/FDT-Quickreference.pdf'` – Ch. 3 ff.

The size of the file should not exceed 64KiB as it will be directly attached to the U-Boot binary. For images of greater dimensions or file size use the splash screen/image feature.

After extracting the U-Boot source the logo file then has to be copied into the source directory and included in the make process. The valid files and directories for this are as follows:

[*U-Boot_src/*]tools/logos	-	directory for all the logo BMPs
[*U-Boot_src/*]board/karo/tx*##*	-	directory for each TX CoM
[*U-Boot_src/*]board/karo/tx*##*/config.mk	-	make config, replace the Ka-Ro defined with the filename of the customized logo in here.

After concluding preparations a customized U-Boot can be compiled as described in Ch. **6** (**Building U-Boot**).

1 Supported by e.g.: GIMP (GNU Image Manipulation Program)
2 Recommended

4.4 DTB

> ***Note:***
> This feature concerns only the Linux Kernel and/or U-Boot itself.
> On systems based upon Windows CE/EC it has no influence.

- **DTB Introduction**, Ch. **4.4.1**
- **Device Tree Boot**, Ch. **4.4.2**
- **Managing the Device Tree**, Ch. **4.4.3**
- **Customize the Device Tree, Ch. 4.4.4**
- **Creating a new DTB, Ch. 4.4.5**
- **FDT Documentation**, Ch. **4.4.6**

4.4.1 DTB Introduction

Accompanying the Linux Kernel's change (*since Linux 3.4*) of hardware initialization, configuration and addition from an effectively hard-coded built-in structure moved to the more generalized and interchangeable by using the *Flattened Device Tree* (**FDT**) (or short: Device Tree (**DT**)) structure. The FDT is provided in form of a *Device Tree Blob* (**DTB**)

The FDT is part of and developed in unison with the Linux Kernel. The Ka-Ro provided and released DTB are created from the Linux Kernel source. Therefore any changes to the FDT are Linux Kernel changes and vice versa. Thus it has to be made sure to use the respective Linux Kernel appendant DTB sources when creating a customized DTB.

The DTB is also used by U-Boot for partially self-configuration (e.g. **`'video_mode'`**). While it is optional (e.g. headless configuration), this feature is always active.

To start a Linux kernel and for configuring the display in U-Boot, a valid DTB file has to be stored in the `'dtb'` partition.

Once a valid DTB has been read by U-Boot, it will then pass it to the Linux Kernel. This also includes the Kernel command line.

U-Boot has the capability and allows users to exchange the TX CoM and Ka-Ro Starterkit specific default **FDT DTB** by a customized, user specific **DTB**. By either of the following ways:

- manual editing the **DTB** (Ch. **4.4.3**)
- upload of a user created **DTB** (Ch. **4.4.5**)

4.4.2 Device Tree Boot

The DTB is primarily used for the Linux Kernel for it's hardware initialization and configuration but U-Boot also uses it to partially self-configuration (e.g. **'video_mode'**). Therefore a valid DTB file has to be stored in the 'dtb' (NAND) partition. The release supplied DTB can be found here (actual filename depends on TX CoM):

```
Linux/target/<soc>-<tx-com-series>-<series[-sub-series]>.dtb
Linux/target/imx??[dl|q]-tx??[dl|q]-<series[-sub-series]>.dtb
```

e.g.:

```
Linux/target/imx6q-tx6q-1010.dtb
Linux/target/imx6dl-tx6u-8010-comtft.dtb
Linux/target/imx53-tx53-x03x.dtb
```

Beside having a valid DTB available there are following major attendant circumstances:

1. U-Boot

   While using the DTB is optional (e.g. headless configuration) for U-Boot, the feature is always active. Therefore at boot-time U-Boot will try to load the DTB from the partition 'dtb', and if empty or invalid, output error messages similar to these, e.g.:

   ```
   No FDT found
   karo_fdt_set_display: Failed to create 'aliases' node: FDT_ERR_BADMAGIC
   'display' node not found in FDT
   Failed to create new display-timing node from 'dev=ldb,1024x768MR-32@60,if=RGB66
    6': -9
   ```

2. Linux

   Once a valid DTB has been read by U-Boot, it will then pass it to the Linux Kernel. This also includes the Kernel command line.

   U-Boot will automatically alter the DTB, respecting the settings given via the customization variables (Ch. **4.1** - **Customization variables**) this can be manually forced using the command:

   ```
   fdt boardsetup
   ```

   Start-up using the variable "bootm_cmd" (Ch. **2.6**) will setup the baseboard and ensure passing the FDT to the Linux Kernel, e.g.:

   ```
   bootm_cmd=bootm ${loadaddr} - ${fdtaddr}
   ```

4.4.3 Managing the Device Tree

U-Boot allows to edit the FDT using the command series '**fdt**', which also allows actions like print, adding and changing (i.e. *set*) and removal of nodes. More information on notation and structure can be found in Ch. **4.4.4** (**Customize the Device Tree**), Ch. **4.4.6** (**FDT Documentation**) and in:

```
U-Boot/FDT-Quickreference.pdf
```

This then edited FDT can be rewritten (saved) onto the TX CoM NAND by saving it into the partition "dtb", by passing memory address and size to the "**nand** write" command, e.g. (***Note:*** Ch. **3.1.3**):

```
nand erase.part dtb
nand write ${fdtaddr} dtb ${fdtsize}
```

The pre-defined variable 'fdtsave' contains these commands as a macro to facilitate the update of the "dtb" partition with the currently active FDT. Executing the following will achieve the same as the commands above:

```
run fdtsave
```

4.4.4 Customize the Device Tree

As the FDT is part of and developed in unison with the Linux Kernel, U-Boot uses a DTB created from the Linux Kernel source. Therefore any changes to the FDT are Linux Kernel changes and vice versa. It has to be made sure to use the respective Linux Kernel appendant DTB sources. The sources of the DTB are to be found in the Linux Kernel source directory under (U-Boot/FDT-Quickreference.pdf):

```
[Linux-src/]arch/arm/boot/dts/*.dts[i]
```

The necessary information on notation and structure for editing the FDT and the ensuing DTB can be found here in the Linux Kernel source:

```
[Linux-src/]Documentation/devicetree/bindings/*
```

4.4.5 Creating a new DTB

To create a new DTB, either after the user has edited the source of the Linux BSP for the respective TX CoM and compile the resulting changes into a new DTB (see also Ch. **5** and U-Boot/FDT-Quickreference.pdf) or to just compile it from new Kernel sources:

```
export ARCH=arm
export CROSS_COMPILE=arm-cortexa7-linux-gnueabi-
make dtbs
```

The resulting DTB is to be uploaded and written to the TX CoM, like any other image, into the "dtb" partition or eMMC. U-Boot will then at boot time load and hand it over to the Linux Kernel.

Compile is done with the "device-tree-compiler" (v1.3.0, a.k.a. "dtc") which is supplied with Ka-Ro ARMSDK-VM (Ch. **5** & **6**).

4.4.6 FDT Documentation

For full and comprehensive documentation and further information about the FDT notation, structure, bindings and handling as well as the "dtc" (Device Tree Compiler) sources, please refer to:

- Kernel (primary):

  [*Linux-src/*]Documentation/devicetree/*
  [*Linux-src/*]Documentation/devicetree/bindings/*

- URL:

  http://devicetree.org/Main_Page
  http://devicetree.org/Device_Tree_Usage
  https://wiki.ubuntu.com/KernelTeam/ARMDeviceTrees
  http://wiki.freebsd.org/FlattenedDeviceTree
  http://elinux.org/Device_Trees

- PDF:

  U-Boot/FDT-Quickreference.pdf
  http://mvista.com/download/fetchdoc.php?docid=351
  http://ozlabs.org/~dgibson/papers/dtc-paper.pdf

- Tools:

  git://www.jdl.com/software/dtc.git
  http://wiki.xilinx.com/device-tree-generator

5 Building U-Boot for TXUL

With the following commands U-Boot can be build successfully with the same settings used for the respective release. For further information on building U-Boot please also see '[*U-Boot_src/*]doc/README.KARO', '[*U-Boot_src/*]doc/README.KARO-TX##' in the 'u-boot-src.tar.bz2' or the general README file in '*U-Boot*' directory on the StarterKit CD and below in Ch. **6** (**Building U-Boot**).

5.1 Unpack and compile the source

```
mkdir u-boot
mkdir build_u-boot_txul
cd u-boot
tar -xjf u-boot-src.tar.bz2
```

The commands following allow the user to compile U-Boot (incl. optional **object directory**):

```
export ARCH=arm
export CROSS_COMPILE=arm-cortexa7-linux-gnueabi-
```

- For TXUL-*0010*:
  `make txul-0010_config O=../build_u-boot_txul`
- For TXUL-*0011*
  `make txul-0011_config O=../build_u-boot_txul`

```
make O=../build_u-boot_txul
```

Important:

The above given specific commands are also supplied by the U-Boot 'README' in the source directory.

The commands above, invoked correctly, will then create the U-Boot binaries in the given **object directory**, defined by the option `"O=..."` (see Ch. **6.5**), an exemplary listing of the files created for the TXUL CoM include following:

```
u-boot
u-boot.bin          (the actual U-Boot binary)
u-boot.lds
u-boot.map
u-boot.srec
```

The TXUL specific file `'u-boot.bin'` can then be started via **`'sbloader'`** or installed in flash for NAND-boot as described in Chapters **2.2** ff.

Note:
For the Ka-Ro TX28 CoM the `'u-boot.bin'` *image file is named* `'u-boot.sb'`

Note:
The Ka-Ro provided image files differ only by name, if compiled from unchanged source, i.e. `u-boot.bin => u-boot-txulul-10.bin`

6 Building U-Boot

With the following commands U-Boot can be build successfully with the same settings used for the respective release. For further information on building U-Boot please also see `'[U-Boot_src/]doc/README.KARO'`, `'[U-Boot_src/]doc/README.KARO-TX##'` in the `'u-boot-src.tar.bz2'` or the general `README` file in `'U-Boot'` directory on the CD and above given Chapter **5 - Building U-Boot for TXUL**.

It is recommend to use a separate **object directory** (Ch. **6.5**) when building, allowing for differentiation in either for the purpose of testing or by TX module targets. This is done with the parameter for '***object directory***' `"O=/path/to/directory"` to the `'make'` command (Example: Ch. **6.3**). The value can either be a relative (as in examples) or full path.

6.1 Requirements

To compile U-Boot the user will need to have a Linux capable of cross compiling. For this purpose Ka-Ro offers a Virtual Appliance, called ARMSDK VM. This Virtual Appliance offers the comprehensive capabilities to use the tool-chain supplied by Ka-Ro either in it's pre-compiled version or from the source. Allowing the user to compile all packages offered by Ka-Ro, including but not limited to U-Boot.

Although it is possible to use all Linux distributions, because of diversity and versatility of Linux distributions the offered *ARMSDK VM Virtual Appliance* is the only means supported by Ka-Ro for cross compilation. For further information concerning the ARMSDK VM please refer to the there enclosed documentation.

The ARMSDK VM Virtual Appliance is available on the Ka-Ro electronics' website public download area:

ARMSDK Download Site

`http://www.karo-electronics.de/arm-sdk`

Note:

Because of legacy support there are two versions of the ARMSDK VM. Please only use the version called 'Squeeze'.

6.2 Unpacking the source

After setting up and preparing the Virtual Appliance, the user has extract the U-Boot sources in appropriate directories, as explained in detail by the package's respective README.

without object directory (default):

```
mkdir u-boot
cd u-boot
tar -xjf u-boot-src.tar.bz2
```

with **object directory** (Ch. **6.5**) :

```
mkdir u-boot
mkdir build_u-boot_tx##
cd u-boot
tar -xjf u-boot-src.tar.bz2
```

6.3 Compiling U-Boot

The commands following show generalized example to compile U-Boot (optional **object directory**):

```
export ARCH=arm
export CROSS_COMPILE=arm-[cpu_family]-linux-gnueabi-
make tx##_config [O=../build_u-boot_tx##]
make [O=../build_u-boot_tx##]
```

For the specific commands please refer to the U-Boot 'README' as supplied in the source, specific to the various TX CoMs. The ARMSDK VM offers some simplifications, for more please consult the documentation there.

These commands above, invoked correctly, will create, either in the U-Boot source directory or in the given object directory, if defined by the option "O=...", the relevant files; dependent on the TX CoM set as target. A listing of the significant files created for the TX48 CoM include following:

```
MLO
u-boot.img
u-boot.bin
[...]
u-boot-spl.bin (see "spl/" sub directory)
```

The resulting files can be started or installed into NAND as described in Ch. **2** (**Update and Recovery**).

Note:
The Ka-Ro provided image files differ only by name, if compiled from unchanged source, i.e. `u-boot.bin => u-boot_tx53.bin`

6.4 Cleaning the Sources

It might be necessary, in case of e.g. a mishap compile, to clean up the source. This can be done by one of the following commands:

```
make distclean [O=../build_u-boot_tx##]
```

or:

```
make mrproper [O=../build_u-boot_tx##]
```

6.5 Object Directory

An object directory, a.k.a. "build dir", allows the user to compile from one source multiple targets without contamination of the source itself. The object directory can be inside the source directory, but it's recommended to keep it outside.

Further it can be specified either by parameter "O=..." or exporting the environment variable "BUILD_DIR" beforehand, similar to "ARCH". Note that the command line "O=..." setting overrides the "BUILD_DIR" environment variable.

Example:

Environment:

```
export BUILD_DIR=../build_u-boot_tx##
export ARCH=arm
export CROSS_COMPILE=arm-[family]-linux-gnueabi-
make tx##_config
make
```

Command option:

```
export ARCH=arm
export CROSS_COMPILE=arm-[family]-linux-gnueabi-
make tx##_config O=../build_u-boot_tx##
make O=../build_u-boot_tx##
```

7 U-Boot commands

Subsequently are some of the most important commands of U-Boot being described. Using the '**help**' (or '**?**') command will print a list of all available commands for the system, also '**help** <command>' (or '**?** <command>') will show the specifics for each single command.

All commands can be abbreviated to the shortest string that uniquely identifies each command or sub-command, e.g. "sa" for '**saveenv**', "pr" for '**printenv**' or "re" for '**reset**'.

U-Boot will auto-complete the commands by pressing the "TAB" key. Respectively it will complete to the least common denominator and show a list of commands, while allowing to auto-complete step-by-step.

The features of U-Boot are configurable and can depend on the TX Series CoM basis, such that some TX Series CoM might present the user different availability of commands. Ka-Ro strives to configure U-Boot to the highest possible commonalities. Please also use the '**help**' (or '**?**') command to find out more about your configuration's commands.

Also note that above described feature-set is subject to change and defined at compile time. For a general introduction to possible changes and customization please refer to Chapter **5** (**Building U-Boot for TXUL**).

All U-Boot commands expect numbers to be entered in hexadecimal input format (exceptions where noted). A leading "0x" prefix is not required.

Important:

Do not use 'set' as abbreviation of the 'setenv' command anymore!

Changes in the available commands allow no longer for 'set' being a unique abbreviation.

Users upgrading from older U-Boot releases will need to reset the environment to apply the changes.

Ch. 2.6.3 - Reset of Variables

7.1 Overview of commands

(Click command to jump)

```
?             - alias for 'help'
base          - print or set address offset
bdinfo        - print Board Info structure
bmp           - manipulate BMP image data
boot          - boot default, i.e., run 'bootcmd'
bootce        - Boot a Windows CE image from RAM
bootd         - boot default, i.e., run 'bootcmd'
bootm         - boot application image from memory
bootp         - boot image via network using BOOTP/TFTP protocol
ceconnect     - Set up a connection to the CE host PC over TCP/IP and download the run-time image
chpart        - change active partition
clocks        - display/set clocks
cls           - clear screen
cmp           - memory compare
coninfo       - print console devices and information
cp            - memory copy
crc32         - checksum calculation
dcache        - enable or disable data cache
dhcp          - boot image via network using DHCP/TFTP protocol
echo          - echo args to console
editenv       - edit environment variable
env           - environment handling commands
ext2load      - load binary file from a Ext2 filesystem
ext2ls        - list files in a directory (default /)
fatinfo       - print information about filesystem
fatload       - load binary file from a dos filesystem
fatls         - list files in a directory (default /)
fatwrite      - write file into a dos filesystem
fdt           - flattened device tree utility commands
go            - start application at address 'addr'
help          - print command description/usage
i2c*          - I2C sub-system
icache        - enable or disable instruction cache
iminfo        - print header information for application image
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
mdio          - MDIO utility commands
mii           - MII utility commands
mm            - memory modify (auto-incrementing address)
mmc           - MMC sub system
mmcinfo       - display MMC info
nand          - define flash/nand partitions
nboot         - simple RAM read/write test
mw            - memory write (fill)
nand          - NAND sub-system
nboot         - boot from NAND device
nbootce       - Boot a Windows CE image from NAND
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
reset         - Perform RESET of the CPU
romupdate*    - Creates an FCB data structure and writes an U-Boot image to flash
run           - run commands in an environment variable
source        - save environment varia bles to persistent storage
setenv        - set environment variables
sleep         - delay execution for some time
source        - run script from memory
tftpboot      - boot image via network using TFTP protocol
time          - run commands and summarize execution time
version       - print monitor, compiler and linker version
```

* Commands are not available on all TX Series CoM

7.2 Commands and explanations

?

⇨ alias for 'help'

base

⇨ print or set address offset which is used for all memory commands

`base` - print address offset for memory commands
`base off` - set address offset for memory commands to 'off'

bdinfo

⇨ print Board Info structure

`bdinfo` - prints the information that U-Boot passes about the board such as memory addresses and sizes, clock frequencies, MAC address, etc.

bmp

⇨ manipulate BMP image data

`bmp info <imageAddr>`
- display image info bmp display `<imageAddr> [x y]`
- display image at x,y

boot

⇨ the same as bootd; boot default, i.e., run 'bootcmd'

bootce

⇨ bootce - Boot a Windows CE image from memory

`bootce [-i]|[addr]`
`addr` - boot image from address 'addr' (default ${fileaddr})
or
`-i` - initialize the WinCE globals data structure (before loading a .nb0 image)

bootd

⇨ bootd - boot default, i.e., run 'bootcmd'

bootm

⇨ boot application image from memory

⇨ boot default, i.e., run 'bootcmd'

`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 bootp

⇨ boot image via network using BOOTP/TFTP protocol

```
bootp [loadAddress] [[hostIPaddr:]bootfilename]
```

ceconnect

⇨ Set up a connection to the CE host PC over TCP/IP and download the run-time image

```
ceconnect [-v] [-t <timeout>]
```

`-v` verbose operation

`-t <timeout>`

- max wait time (#sec) for the connection

chpart

⇨ change active partition

clocks* (only: TX51, TX53)

⇨ display/set clocks

cls

⇨ clear screen

cmp

⇨ memory compare

```
cmp [.b, .w, .l] addr1 addr2 count
```

compare memory

`.b`	- access memory in size	byte	(8 bit)
`.w`	- access memory in size	word	(16 bit)
`.l`	- access memory in size	long	(32 bit)
`addr1`	- address of memory area 1		
`addr2`	- address of memory area 2		
`count`	- number of elements (byte, word, long) to compare		

coninfo

⇨ print console devices and information

cp

⇨ memory copy

`cp [.b, .w, .l] source target count`

copy memory

`.b`	- access memory in size byte	(8 bit)
`.w`	- access memory in size word	(16 bit)
`.l`	- access memory in size long	(32 bit)
`source`	- source address of the data	
`target`	- target address of the data	
`count`	- number of elements (byte, word, long) to copy	

crc32

⇨ checksum calculation

`crc32 addr1 count [addr2]`

calculate checksum of data starting at address `addr1` for length `count` and (if given) store the result at address `addr2`

`addr1`	- start address of data
`count`	- number of memory addresses of the data
`addr2`	- storage address for the result of the calculation

dcache

⇨ enable or disable data cache

`dcache [on, off, flush]`

enable, disable, or flush data (writethrough) cache

dhcp

⇨ invoke DHCP client to obtain IP/boot params and `load ${bootfile}` via TFTP if the environment variable `'autoload'` is set to `'yes'`

echo

⇨ echo args to console

`echo [args]` - echo args to console; `\c` suppresses newline

editenv

⇨ edit environment variable

`editenv name` - edit environment variable `'name'` env

⇨ environment handling commands

```
env default [-f] -a              - [forcibly] reset default environment
env default [-f] var [...]       - [forcibly] reset variable(s) to their default values
env delete [-f] var [...]        - [forcibly] delete variable(s)
env edit name                    - edit environment variable
env export [-t | -b | -c] [-s size] addr [var ...]
                                 - export environment
env import [-d] [-t | -b | -c] addr [size]
                                 - import environment
env print [-a | name ...]        - print environment
env run var […]                  - run commands in an environment variable
env save                         - save environment
env set [-f] name [arg ...]      - [forcibly] set environment variable
```

ext2load

⇨ load binary file from a Ext2 filesystem

`ext2load <interface> <dev[:part]>  [addr] [filename] [bytes]`

load binary file `'filename'` from `'dev'` on `'interface'` to address `'addr'` from ext2 filesystem

ext2ls

⇨ list files in a directory (default /)

`ext2ls <interface> <dev[:part]> [directory]`

list files from `'dev'` on `'interface'` in a `'directory'`

fatinfo

⇨ print information about filesystem

`fatinfo <interface> <dev[:part]>`

print information about filesystem from `'dev'` on `'interface'`

fatload

⇨ load binary file from a dos filesystem

`fatload <interface> <dev[:part]> <addr> <filename> [bytes]`

load binary file `'filename'` from `'dev'` on `'interface'` to address `'addr'` from DOS filesystem

fatls

⇨ list files in a directory (default /)

`fatls <interface> <dev[:part]> [directory]`

list files from `'dev'` on `'interface'` in a `'directory'` fatwrite

⇨ write file into a dos filesystem

```
fatwrite <interface> <dev[:part]> <addr> <filename> <bytes>
```

write file 'filename' from the address 'addr' in RAM to 'dev' on 'interface'

fbdump

⇨ dump framebuffer contents to flash

```
fbdump [partition name]
```

default partition name: 'logo'

fdt

⇨ flattened device tree utility commands

```
fdt addr   <addr> [<length>]   - Set the fdt location to <addr>
fdt boardsetup                 - Do board-specific set up
fdt move   <fdt> <newaddr> <length>
                               - Copy the fdt to <addr> and make it active
fdt resize                     - Resize fdt to size + padding to 4k addr
fdt print  <path> [<prop>]     - Recursive print starting at <path>
fdt list   <path> [<prop>]           - Print one level starting at <path>
fdt set    <path> <prop> [<val>]
                               - Set <property> [to <val>]
fdt mknode <path> <node>       - Create a new node after <path>
fdt rm     <path> [<prop>]     - Delete the node or <property>
fdt header                     - Display header info
fdt bootcpu <id>               - Set boot cpuid
fdt memory <addr> <size>       - Add/Update memory node
fdt rsvmem print               - Show current mem reserves
fdt rsvmem add <addr> <size>   - Add a mem reserve
fdt rsvmem delete <index>      - Delete a mem reserves
fdt chosen [<start> <end>]     - Add/update the /chosen branch in the tree
                               <start>/<end> - initrd start/end addr
```

Note:
Dereference aliases by omitting the leading '/', e.g. `fdt print ethernet0`.

go

⇨ start application at address 'addr'

```
go addr [arg ...]         - start application at address 'addr' passing 'arg' as arguments
```

help (alias: ?)

⇨ print online help

`help [command ...]` - show help information (for `'command'`)
`'help'` prints online help for the monitor commands.
Without arguments, it prints a short usage message for all commands.
To get detailed help information for specific commands you can type `'help'` with one or more command names as arguments.

i2c* (not: TX28, TX48, TX51)

⇨ I2C sub-system

`i2c crc32 chip address[.0, .1, .2] count`
- compute CRC32 checksum

`i2c loop chip address[.0, .1, .2] [# of objects]`
- looping read of device

`i2c md chip address[.0, .1, .2] [# of objects]`
- read from I2C device

`i2c mm chip address[.0, .1, .2]`
- write to I2C device (auto-incrementing)

`i2c mw chip address[.0, .1, .2] value [count]`
- write to I2C device (fill)

`i2c nm chip address[.0, .1, .2]`
- write to I2C device (constant address)

`i2c probe [address]` - test for and show device(s) on the I2C bus

`i2c read chip address[.0, .1, .2] length memaddress`
- read to memory

`i2c write memaddress chip address[.0, .1, .2] length`
- write memory to i2c

`i2c reset` - re-init the I2C Controller

`i2c speed [speed]` - show or set I2C bus speed

icache

⇨ enable or disable instruction cache

`icache [on, off, flush]`
enable, disable, or flush instruction cache

iminfo

⇨ print header information for application image

`iminfo addr [addr …]` - print header information for application image starting at address 'addr' in memory; this includes verification of the image contents (magic number, header and payload checksums) imxtract

⇨ extract a part of a multi-image

`imxtract addr part [dest]`

extract <`part`> from legacy image at <`addr`> and copy to <`dest`>

itest

⇨ return true/false on integer compare

`itest [.b, .w, .l, .s] [*]value1 <op> [*]value2`

`.b`	- access memory in size	byte	(8 bit)
`.w`	- access memory in size	word	(16 bit)
`.l`	- access memory in size	long	(32 bit)

loadb

⇨ load binary file over serial line (kermit mode)

`loadb [ off ] [ baud ]` - load binary file over serial line with offset `'off'` and baudrate `'baud'`

loads

⇨ load S-Record file over serial line

`loads [ off ]` - load S-Record file over serial line with offset `'off'`

loady

⇨ load binary file over serial line (ymodem mode)

`loady [ off ] [ baud ]` - load binary file over serial line with offset `'off'` and baudrate `'baud'`

loop

⇨ infinite loop on address range

`loop [.b, .w, .l] address count`

loop on a set of addresses

`.b`	- access memory in size	byte	(8 bit)
`.w`	- access memory in size	word	(16 bit)
`.l`	- access memory in size	long	(32 bit)

`address` - start address of the loop
`count` - number of objects to read

This command can only be terminated by resetting the board!

md

⇨ memory display, used to display memory contents both as hexadecimal and ASCII data.

```
md [.b, .w, .l] address [count]
```

memory display

`.b` - access memory in size byte (8 bit)
`.w` - access memory in size word (16 bit)
`.l` - access memory in size long (32 bit)
`address` - start address
`count` - number of objects to be displayed

mdio

⇨ MDIO utility commands

`mdio list` - List MDIO buses
`mdio read <phydev> [<devad>.]<reg>`
- read PHY's register at `<devad>.<reg>`
`mdio write <phydev> [<devad>.]<reg> <data>`
- write PHY's register at `<devad>.<reg>`

`<phydev>` may be:
`<busname> <addr>`
`<addr>`
`<eth name>`

`<addr> <devad>`, and `<reg>` may be ranges, e.g. `1-5.4-0x1f`.

mii

⇨ MII utility commands

`mii device` - list available devices
`mii device <devname>` - set current device
`mii info <addr>` - display MII PHY info
`mii read <addr> <reg>` - read MII PHY `<addr>` register `<reg>`
`mii write <addr> <reg> <data>`
- write MII PHY `<addr>` register `<reg>`
`mii dump <addr> <reg>` - pretty-print `<addr> <reg>` (0-5 only)
`<addr>` and/or `<reg>` may be ranges, e.g. 2-7.

mm

⇨ memory modify (auto-incrementing); displays the memory address and current content and prompts for hexadecimal user input as desired new content for this address; the address is automatically incremented each time

```
mm[.b, .w, .l] address
```

`.b` - access memory in size byte (8 bit)
`.w` - access memory in size word (16 bit)
`.l` - access memory in size long (32 bit)
`address` - start address for modification

mmc

⇨ MMC sub system

```
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
```

`mmc part` - lists available partition on current mmc device

`mmc dev [dev] [part]` - show or set current mmc device [partition]

`mmc list` - lists available devices

`mmc hwpartition [args...]` - does hardware partitioning

arguments (sizes in 512-byte blocks):

`[user [enh start cnt] [wrrel {on|off}]]` - sets user data area attributes

`[gp1|gp2|gp3|gp4 cnt [enh] [wrrel {on|off}]]` - general purpose partition

`[check|set|complete]` - mode, complete set partitioning completed

WARNING: Partitioning is a write-once setting once it is set to complete.
Power cycling is required to initialize partitions after set to complete.

`mmc bootbus dev boot_bus_width reset_boot_bus_width boot_mode`
- Set the BOOT_BUS_WIDTH field of the specified device

`mmc bootpart-resize <dev> <boot part size MB> <RPMB part size MB>`
- Change sizes of boot and RPMB partitions of specified device

`mmc partconf dev boot_ack boot_partition partition_access`
- Change the bits of the PARTITION_CONFIG field of the specified device

`mmc rst-function dev value`
- Change the RST_n_FUNCTION field of the specified device
WARNING: This is a write-once field and 0 / 1 / 2 are the only valid values.

`mmc setdsr <value>` - set DSR register value

mmcinfo

⇨ display MMC info

`mmcinfo` - device number of the device to dislay info of mtdparts

⇨ define flash/nand partitions

`mtdparts` - list partition table

`mtdparts delall` - delete all partitions

`mtdparts del part-id` - delete partition (e.g. `part-id = nand0,1`)

`mtdparts default` - reset partition table to defaults

`mtdparts add <mtd-dev> <size>[@<offset>] [<name>] [ro]`
- add partition

<u>the command uses three environment variables:</u>

`partition` - keeps current partition identifier

```
partition    := <part-id>
<part-id>    := <dev-id>,part_num
```

`mtdids` - linux kernel mtd device id <-> u-boot device id mapping

```
mtdids        = <idmap>[,<idmap>,...]
<idmap>      := <dev-id>=<mtd-id>
<dev-id>     := [nand | nor]<dev-num>
<dev-num>    := mtd device number, 0
<mtd-id>     := unique device tag used by linux kernel to find
                mtd device (mtd->name)
```

`mtdparts` - partition list

```
mtdparts      = mtdparts=<mtd-def>[;<mtd-def>...]
<mtd-def>    := <mtd-id>:<part-def>[,<part-def>...]
<mtd-id>     := unique device tag used by linux kernel to find
                mtd device (mtd->name)
<part-def>   := <size>[@<offset>][<name>][<ro-flag>]
<size>       := standard linux memsize OR '-' to denote all
                remaining space
<offset>     := partition start offset within the device
<name>       := '(' NAME ')'
<ro-flag>    := when set to 'ro' makes partition read-only (not
                used, passed to kernel)
```

mtest

⇨ simple RAM test

```
mtest [start [end [pattern [iterations]]]]
```

simple RAM read/write test

`start`	- start address of the RAM test area
`end`	- end address of the RAM test area
`pattern`	- pattern, that is applied to the RAM for testing

Note:
This test changes the contents of the RAM and may therefore cause crashing of the system if the memory area, where this test is applied to, is needed for the system operation.

mw

⇨ memory write (fill)

```
mw [.b, .w, .l] address value [count]
```

write memory

`.b`	- access memory in size byte (8 bit)
`.w`	- access memory in size word (16 bit)
`.l`	- access memory in size long (32 bit)
`address`	- start address to write to
`value`	- value to write to the address(es)
`count`	- number of addresses to which "`value`" is written

nand

⇨ NAND sub-system

```
nand info                 - show available NAND devices
nand device [dev]         - show or set current device
nand read                 - addr off|partition size
nand write                - addr off|partition size
                          read/write 'size' bytes starting at offset 'off'
                          to/from memory address 'addr', skipping bad blocks.
nand read.raw             - addr off|partition
nand write.raw            - addr off|partition
                          Use read.raw/write.raw to avoid ECC and access the page as-is.
nand write.trimffs        - addr off|partition size
                          write 'size' bytes starting at offset 'off' from memory address
                          'addr', skipping bad blocks and dropping any pages at the end
                          of eraseblocks that contain only 0xFF
nand erase[.spread] [clean] off size
                          - erase 'size' bytes from offset 'off'
                          With '.spread', erase enough for given file size, otherwise, 'size'
                          includes skipped bad blocks.
nand erase.part [clean] partition
                          - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad                  - show bad blocks
nand dump[.oob] off       - dump page
nand scrub [-y] off size | scrub.part partition | scrub.chip
                          - really clean NAND erasing bad blocks (UNSAFE)
nand markbad off […]      - mark bad block(s) at offset (UNSAFE)
nand biterr off           - make a bit error at offset (UNSAFE)
```

Note:

*Please see Chapter **3.1***

nboot

⇨ boot from NAND device

```
nboot [partition] | [[[loadAddr] dev] offset]
```

nbootce

⇨ Boot a Windows CE image from NAND

```
nbootce [off|partitition]
                          off        - flash offset (hex)
                          partition  - partition name
```

nfs

⇨ boot image via network using NFS protocol

```
nfs [loadAddress] [host ip addr:bootfilename]
```

nm

⇨ memory modify (constant address → non-incrementing); displays the memory address and current content and prompts for hexadecimal user input as desired new content for this address

`nm [.b, .w, .l] address`

memory modify, read and keep address

`.b`	- access memory in size	byte	(8 bit)
`.w`	- access memory in size	word	(16 bit)
`.l`	- access memory in size	long	(32 bit)
`address`	- address for modification		

ping

⇨ send ICMP ECHO_REQUEST to network host

`ping pingAddress`

printenv

⇨ print environment variables

`printenv`	- print values of all environment variables
`printenv name`	- print value of environment variable 'name'

reset

⇨ Perform RESET of the CPU

romupdate* (not: TX48, TX51)

⇨ Creates an FCB data structure and writes an U-Boot image to flash

`romupdate [-f {<part>|block#}] [-r [{<part>|block#}]] [-e #] [<address>] [<length>]`

`-f <part>`	write bootloader image to partition `<part>`
`-f #`	write bootloader image at block # (decimal)
`-r`	write redundant bootloader image at next free block after first image
`-r <part>`	write redundant bootloader image to partition `<part>`
`-r #`	write redundant bootloader image at block # (decimal)
`-e #`	specify number of redundant blocks per boot loader image only valid if `-f` or `-r` specify a flash address rather than a partition name

`<address>`

RAM address of bootloader image (default: *`${fileaddr}`*)

`<length>`

length of bootloader image in RAM (default: *`${filesize}`*) run

⇨ run commands in an environment variable; environment variables can also store sequences of commands; run can be called with several variables as arguments

`run var [...]`	- run the commands in the environment variable(s) `'var'` If a variable contains several commands and the execution of one command fails, the remaining commands are executed anyway! If a call of run contains several variables and the execution of one command fails, the execution of run is terminated and the remaining variables are NOT executed!

saveenv

⇨ save environment variables to persistent storage
(all unsaved changes to the environment variables will be lost when the system is rebooted next time)

setenv

⇨ set environment variables

`setenv name value ...`	- set environment variable `'name'` to `'value ...'`
`setenv name`	- delete environment variable `'name'` Remember that name and value have to be separated by space and/or tab characters!

sleep

⇨ delay execution for some time

`sleep N`	- delay execution for N seconds (N is decimal!)

source

⇨ run script from memory

`source [addr]`	- run script starting at `addr` - A valid image header must be present

tftpboot

⇨ boot image via network using TFTP protocol

`tftpboot [loadAddress] [bootfilename]`

time

⇨ run commands and summarize execution time

`time command [args...]`

version

⇨ print monitor version (prints version and build date of currently running U-Boot)

8 Document revision history

Revision	Changes
2016-05-11	Minor Changes: Corrections
2016-03-18	Minor Changes: Cleanup, corrections, changed monospaced font to: Liberation Mono
2015-11-13	Initial Release