Enclustra Build Environment - Documentation

Antmicro Ltd for Enclustra GmbH

2015-07-03

CHAPTER

ONE

INTRODUCTION

This is user documentation for the Enclustra Build Environment project.

1.1 Version Information

Date	Rev	Author	Changes
2015-05-08	0.1.0	Karol Gugala	Builsystem description
2015-05-11	0.1.1	Aleksandra Szawara	Language check

CHAPTER

TWO

BUILD ENVIRONMENT

This chapter describes the build environment usage. The whole build environment is written in `Python`. Its work is determined by *ini* files placed in a specific directory layout.

2.1 Prerequisites

To run the build script Python interpreter is required. The system is compatible with both Python 2 and Python 3.

The build environment requires an additional software installed as listed below.

Table 2.1: Required software

tool	minimal version	comments
dialog	1.1-20120215	Required only in the GUI mode
make	3.79.1	
git	1.7.8	
tar	1.15	
wget	1.0	
c++ compiler		Required to build a busybox rootfs

For the information how to install required packages in the supported systems please refer to corresponding subsection (*OpenSUSE 13.2 (Harlequin)*, *CentOS 7*, *Ubuntu 14.04 LTS*).

Additionally the following Python modules are required (this applies to every supported distribution):

- os2emxpath
- backports
- ntpath
- pkg_resources
- opcode
- posixpath
- sre_constants

- nturl2path
- sre_parse
- sre_compile
- pyexpat
- strop
- genericpath
- repr

Those can be obtained using a pip tool:

```
sudo pip install os2emxpath backports ntpath pkg_resources opcode posixpath sre_constants\
                 nturl2path sre_parse sre_compile pyexpat strop genericpath repr
```

Note: Either dialog Python module or external application are required only when using the build environment's GUI.

2.1.1 OpenSUSE 13.2 (Harlequin)

```
sudo pip install argparse
sudo yzpper install -y dialog git make
sudo yzpper install -y u-boot-tools gcc patch
sudo yzpper install -y gcc-c++
sudo yzpper install -y flex bison
sudo yzpper install -y linux32
```

2.1.2 CentOS 7

```
sudo yum install -y dialog make git tar wget
sudo yum -y groupinstall 'Development Tools'
sudo yum install -y glibc.i686 libgcc.i686 libstdc++.i686 glibc-devel.i686
```

2.1.3 Ubuntu 14.04 LTS

```
sudo apt-get install -y u-boot-tools
sudo apt-get install -y git
sudo apt-get install -y gcc-multilib
sudo apt-get install -y python-pip python-dev build-essential
sudo pip install --upgrade pip
sudo pip install --upgrade virtualenv
```

2.2 Directory Structure

The build environment is designed to work with a specific directory structure depicted below

```
.
|-- bin
|-- binaries
|-- sources
|   |-- target_submodule_1
|   |-- target_submodule_2
|   |-- target_submodule_3
|   |-- target_submodule_4
|-- targets
|   |-- Family_1
|       |-- Board_1
|       |-- Board_2
|   |-- Family_2
|       |-- Board_3
|-- target_output
```

Table 2.2: Folder description

Folder	function
bin	Remote toolchains installation folder.
binaries	Additional targets binaries download folder.
sources	master_git_repository clone folder. It contains submodules folders.
targets	Targets configuration. Family folders containing boards folders in which the device configuration files are placed.
target_output	A folder generated during the building process. After a succefull building of a target, stores the output files.

Important: Output folders are named according to scheme:

`out_<family>_<module>_<board>_<bootmode>`.

2.3 Repositories Structure

The `sources` directory is a master git repository with a number of submodules pointing to actual code repositories. During the fetch phase the build environment synchronizes only the submodules required for building selected targets.

```
.
|-- master_git_repository
|   |-- target_submodule_1
|   |-- target_submodule_2
|   |-- target_submodule_3
```

2.4 General Environment Configuration

Environment settings are stored in the `enclustra.ini` file in the main directory of the build environment. Before building one may need to adjust the general settings of the build environment. One of the most crucial setting is number of threads used in a parallel building. This parameter is set in the `[general]` section in the `ntreads` key. Additionally parameters in the `[debug]` section allows user to adjust a logging settings:

- If the `debug-calls` option if set to `true` an external tools calls (such as `make`, `tar` etc.) will be displayed.
- If the `quiet-mode` option is set to `true` a build log of the targets will not be printed in the terminal, only information about actual build state will be shown. This option does not affect the `build-logfile` option.
- If the `build-logfile` option is set to a file name the build environment will write the whole output to the file. If the option is not set the output will not be logged.
- If the `break-on-error` option is set to `true` the build environment will stop its work and exit on the first error. Otherwise the build environment will only print an error message and continue to work on a next available target.

2.5 Supported Devices

Table 2.3: Supported devices

Family	Module	Base board	Available targets
Xilinx	Mars ZX3	Mars Starter	Linux, U-Boot, Busybox
Xilinx	Mars ZX3	Mars PM3	Linux, U-Boot, Busybox
Xilinx	Mercury ZX1	Mercury PE1	Linux, U-Boot, Busybox
Xilinx	Mercury ZX5	Mercury PE1	Linux, U-Boot, Busybox
Altera	Mercury SA1	Mercury PE1	Linux, U-Boot, Busybox

CHAPTER

THREE

USAGE

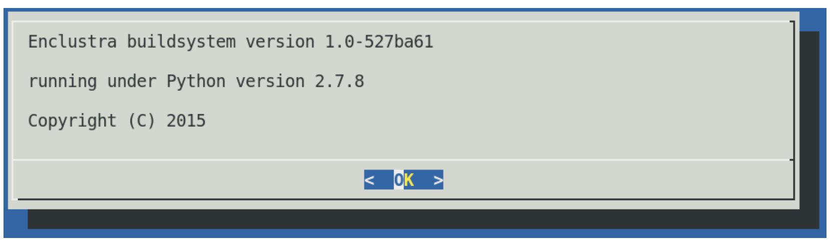
3.1 GUI

In order to build the software for a chosen board using the GUI, follow these steps:

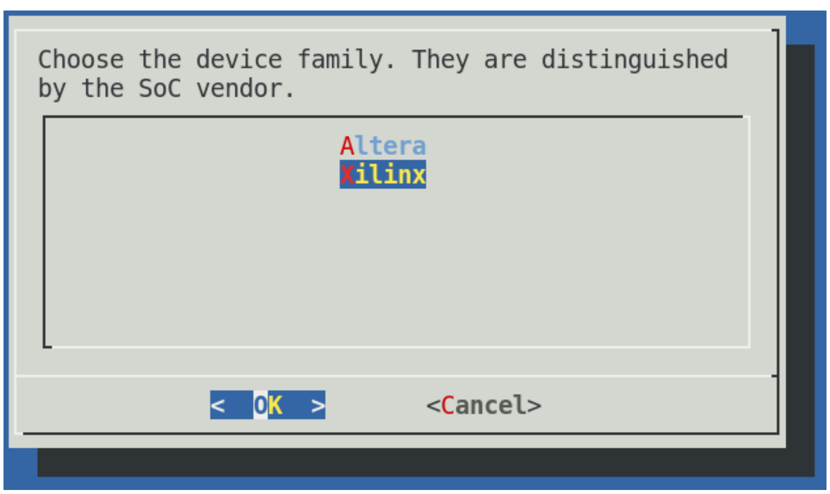
1. Clone build environment repository with:

```
git clone https://github.com/enclustra-bsp/enclustra-buildscripts.git
```

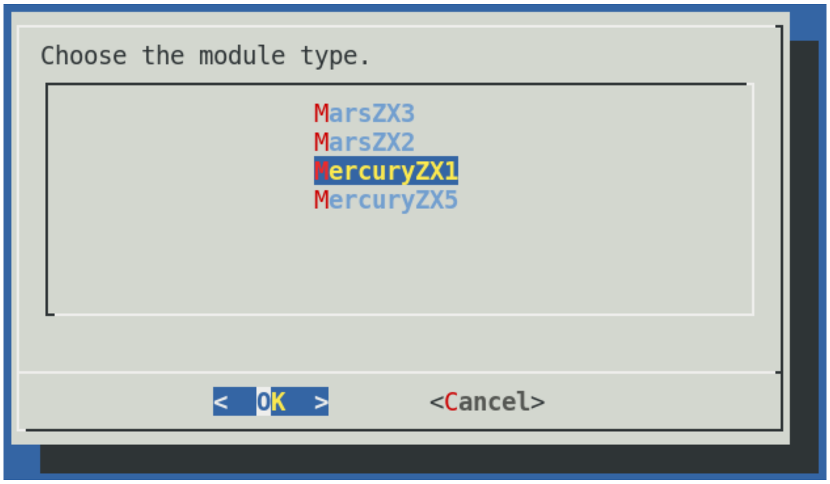
1. Run `./build.py` script.
2. The welcome screen provides basic information about the version and environment.



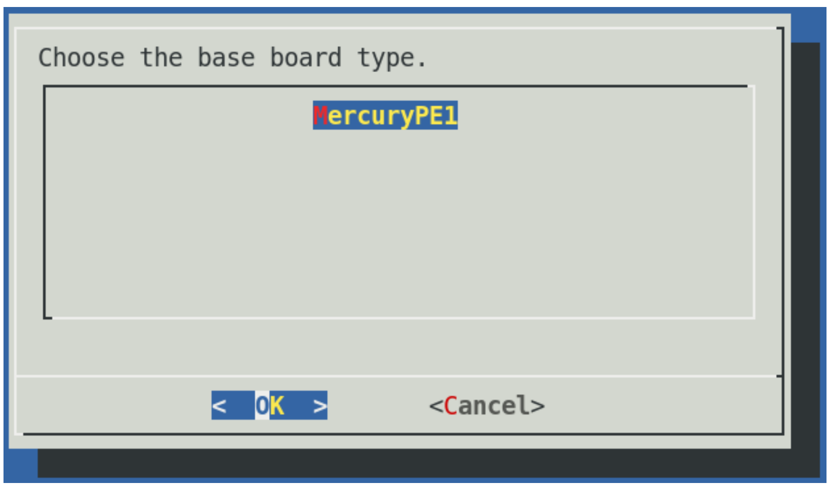
3. Choose the device family.



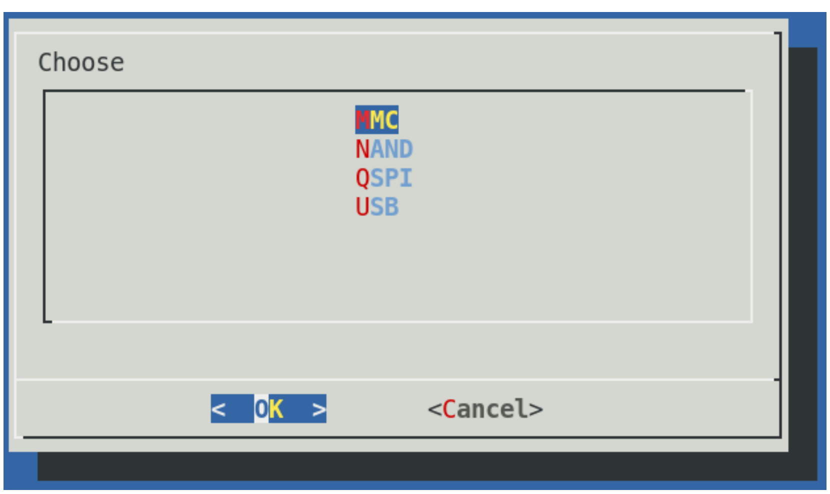
4. Choose the module.



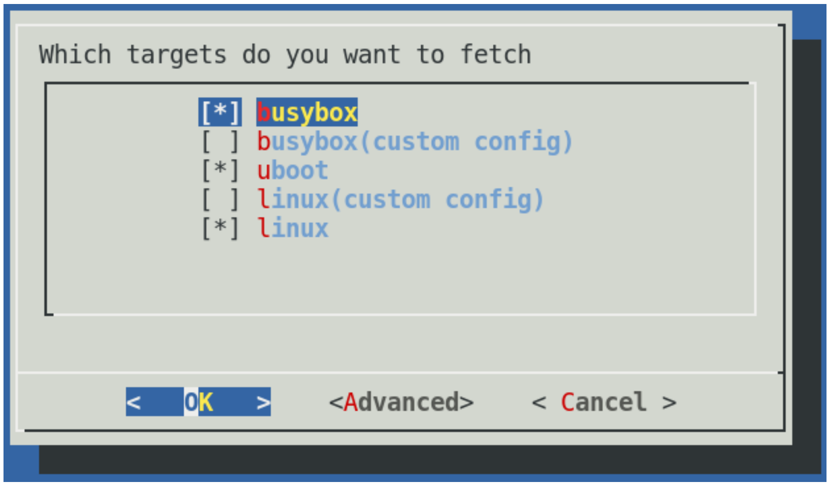
5. Choose the board.



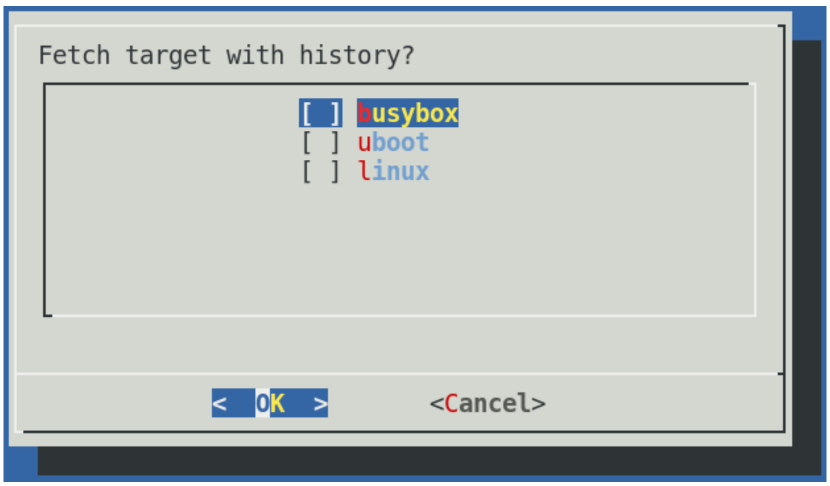
6. Choose the boot mode.



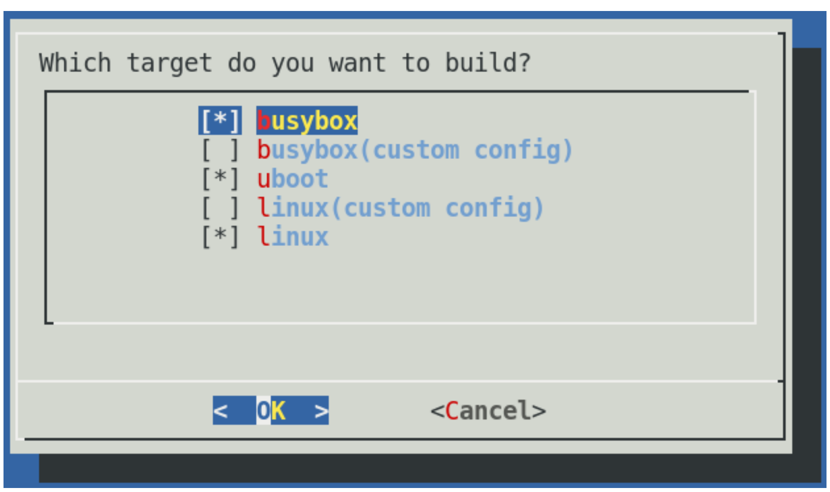
7. Choose which targets available for the chosen family will be fetched. On the bottom of the screen a short description of the highlighted target is displayed. Choosing certain targets may disable fetching others - information about that is displayed on the bottom of the screen.



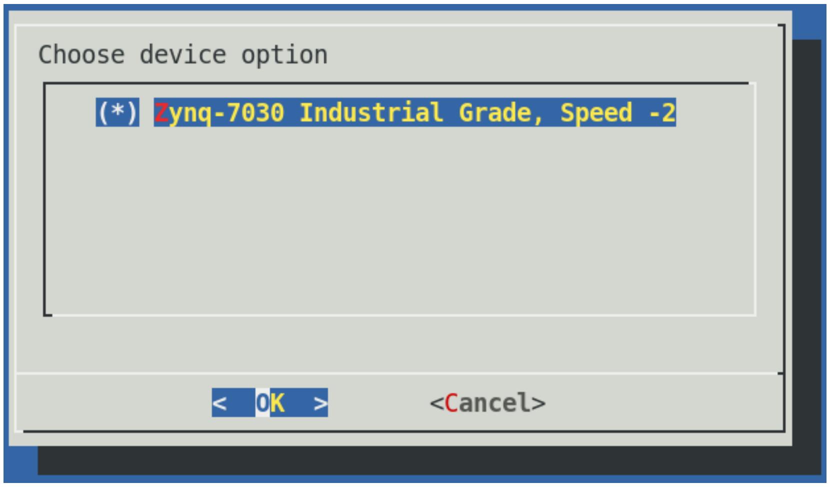
- Under `Advanced` a user may choose whether repositories will be fetched with their history.



8. Choose which targets will be built. On the bottom of the screen a short description of a highlighted target is displayed. Choosing certain targets may disable building others - information about that is displayed on the bottom of the screen.



9. Choose the exact version of the device (chip type, industrial/commercial grade, speed grade).



10. The build environment will fetch and build the chosen targets.

3.2 Command Line

The building process can be invoked from the command line. A list of the available command line options can be obtained with:

```
./build.py --help

usage: build.py [-h] [-d device] [--disable-fetch target]
                [--fetch-history target] [--disable-build target] [-t target]
                [-l] [-L] [--list-dev-options] [-o option_number]

Enclustra Build Environment

optional arguments:
  -h, --help            show this help message and exit
  -d device, --device device
                        Device ini file location
  --disable-fetch target
                        Exclude specific target from fetching
  --fetch-history target
                        Fetch specific target with history
  --disable-build target
                        Exclude specific target from building
  -t target, --target target
                        Fetch and build on the chosen target
  -l, --list            List default targets for chosen device
  -L, --list-all        List all available targets for chosen device
  --list-dev-options    List all available device options for chosen device
  -o option_number, --dev-option option_number
                        Set device option. If unset default will be used
```

If the build.py script is invoked with the -d option, the build environment switches to console mode. This mode requires providing the location of the device configuration within the `targets` directory for the required device, e.g. for the *MarsZX3* module on the *MarsPM3* base board in *QSPI* boot mode the command is:

```
./build.py -d Xilinx/MarsZX3/MarsPM3/QSPI
```

Such a command will fetch and build all the default targets for a selected device. To list all the default targets for a selected device, the user needs to add the `-l` switch to the command, e.g.:

```
./build.py -d Xilinx/MarsZX3/MarsPM3/QSPI -l
```

The `-L` option will list all the available targets.

The `--disable-fetch` and the `--disable-build` options will exclude the selected target from the fetching or building, e.g.:

```
./build.py -d Xilinx/MarsZX3/MarsPM3/QSPI --disable-fetch linux --disable-build linux
```

That will fetch and build all the default targets except `linux`, for a selected device.

The `-t` option will fetch and build only the selected target, e.g.:

```
./build.py -d Xilinx/MarsZX3/MarsPM3/QSPI -t linux
```

That will fetch and build only the `linux` target for the selected device. This option can be used with the `--disable-fetch` and the `--disable-build` options.

The `--list-dev-options` option will list all the available options for the chosen device.

The `-o` option allows to choose a device option for the selected device. The user should provide a number of the device option. The number of a specific option can be obtained by listing all the available device options. If no device option is provided, the default one will be used.

CHAPTER

FOUR

DEPLOYMENT

Note: Target output folders are named according to scheme:

`out_<family>_<module>_<board>_<bootmode>`.

All the guides in this section are using variables from the U-Boot default environment. Moreover the boot scripts used by U-Boot also rely on those variables. If the environment was changed and saved earlier, U-Boot will always use the new ones, even after updating the U-Boot itself. To restore the default environment run the following command in the U-Boot command line:

```
env default - a
```

This will not overwrite the stored environment but will only restore the default one in the current run. To permanently restore the default environment, the `saveenv` command has to be invoked.

4.1 Xilinx Family

In order to create the boot image for the Xilinx family of devices, one has to use the Xilinx SDK software. Please refer to the Xilinx Wiki for further information. All the files required to build the boot image (including `boot.bif`) are available in the output directory of a selected device.

Note: In the following instructions the boot image is referred to as a `boot.bin` file.

4.1.1 SD Card (MMC)

In order to deploy images to a SD Card and boot from it, do the following steps:

1. Create a FAT formatted partition as the first one on a SD Card. The size of the partition should be at least 16 MB.
2. Copy `boot.bin`, `uimage`, `devicetree.dtb`, `uboot.scr` and `uramdisk` from the build environment output directory to that partition.
3. Insert the card into the SD Card slot on the board.
4. Set the board to boot from the SD Card (refer to the board User Manual).

5. The board should boot the Linux system.

If one wants to manually trigger booting from a SD Card, the following command has to be invoked from the U-Boot command line:

```
run sdboot
```

4.1.2 QSPI Flash

Table 4.1: Xilinx Family QSPI Flash Layout

Partition	Offset	Size
Boot image	0x0	0x600000
Linux kernel	0x600000	0x500000
Linux Device Tree	0xB00000	0x80000
U-Boot environment	0xB08000	0x80000
Bootscript	0xC00000	0x80000
Rootfs	0xC40000	0x3C0000

In order to deploy images to QSPI Flash and boot from it, do the following steps:

1. Setup a TFTP server on a host computer.
2. Power up the board and boot to U-Boot (e.g. from a SD Card (MMC)).
3. Connect an Ethernet cable to the device.
4. Connect a serial console to the device (e.g. using PuTTY or picocom).
5. Setup the U-Boot connection parameters (in the U-Boot console):

   ```
   setenv ipaddr 'xxx.xxx.xxx.xxx'
   # where xxx.xxx.xxx.xxx is the board address
   setenv serverip 'yyy.yyy.yyy.yyy'
   # where yyy.yyy.yyy.yyy is the server (host computer) address
   ```

6. Copy `boot.bin`, `uimage`, `devicetree.dtb`, `uboot.scr` from the build environment output directory to the TFTP server directory
7. Set memory pinumuxe to QSPI Flash:

   ```
   zx_set_storage QSPI
   ```

8. Update the boot image:

   ```
   mw.b ${bootimage_loadaddr} 0xFF ${bootimage_size}
   tftpboot ${bootimage_loadaddr} ${bootimage_image}
   sf probe
   sf erase ${qspi_bootimage_offset} ${bootimage_size}
   sf write ${bootimage_loadaddr} ${qspi_bootimage_offset} ${filesize}
   ```

9. Update the boot script image:

```
mw.b ${bootscript_loadaddr} 0xFF ${bootscript_size}
tftpboot ${bootscript_loadaddr} ${bootscript_image}
sf probe
sf erase ${qspi_bootscript_offset} ${bootscript_size}
sf write ${bootscript_loadaddr} ${qspi_bootscript_offset} ${filesize}
```

10. Update the Linux kernel:

```
mw.b ${kernel_loadaddr} 0xFF ${kernel_size}
tftpboot ${kernel_loadaddr} ${kernel_image}
sf probe
sf erase ${qspi_kernel_offset} ${kernel_size}
sf write ${kernel_loadaddr} ${qspi_kernel_offset} ${filesize}
```

11. Update the devicetree image:

```
mw.b ${devicetree_loadaddr} 0xFF ${devicetree_size}
tftpboot ${devicetree_loadaddr} ${devicetree_image}
sf probe
sf erase ${qspi_devicetree_offset} ${devicetree_size}
sf write ${devicetree_loadaddr} ${qspi_devicetree_offset} ${filesize}
```

12. Update the rootfs image:

```
mw.b ${ramdisk_loadaddr} 0xFF ${ramdisk_size}
tftpboot ${ramdisk_loadaddr} ${ramdisk_image}
sf probe
sf erase ${qspi_ramdisk_offset} ${ramdisk_size}
sf write ${ramdisk_loadaddr} ${qspi_ramdisk_offset} ${filesize}
```

13. Set the board to boot from the QSPI Flash (refer to the board User Manual).
14. The board should boot the Linux system.

If one wants to manually trigger booting from the QSPI Flash, the following command has to be invoked from the U-Boot command line:

```
run qspiboot
```

Note: Note that step 8 to 12 can be invoked independently.

4.1.3 NAND Flash

The Xilinx family devices cannot boot directly from a NAND Flash memory. The FSBL and the U-Boot have to be started from SD Card (MMC) or QSPI Flash. Please refer to *SD Card (MMC)* or *QSPI Flash* in order to boot U-Boot from SD Card or QSPI Flash. When U-Boot is booted it can load and boot the Linux system stored on the NAND Flash memory.

Table 4.2: Xilinx Family NAND Flash Layout

Partition	Offset	Size
Linux kernel	0x0	0x500000
Linux Device Tree	0x500000	0x100000
Bootscript	0x600000	0x100000
Rootfs	0x700000	Rest of the NAND Storage space

In order to deploy images and boot the Linux system from NAND Flash, do the following steps:

1. Setup an TFTP server on a host computer.
2. Power up the board and boot to U-Boot (e.g. from a SD Card (MMC)).
3. Connect an Ethernet cable to the device.
4. Connect a serial console to the device (e.g. using PuTTY or picocom).
5. Copy `uimage`, `devicetree.dtb`, `uboot.scr` and `uramdisk` files from the build environment output directory to the TFTP server directory.
6. Setup the U-Boot connection parameters (in the U-Boot console):

```
setenv ipaddr 'xxx.xxx.xxx.xxx'
# where xxx.xxx.xxx.xxx is the board address
setenv serverip 'yyy.yyy.yyy.yyy'
# where yyy.yyy.yyy.yyy is the server (host computer) address
```

7. Set the memory pinumux to NAND Flash:

```
zx_set_storage NAND
```

8. Update the boot script image:

```
mw.b ${bootscript_loadaddr} 0xFF ${bootscript_size}
tftpboot ${bootscript_loadaddr} ${bootscript_image}
nand device 0
nand erase.part nand-bootscript
nand write ${bootscript_loadaddr} nand-bootscript ${filesize}
```

9. Update the Linux kernel:

```
mw.b ${kernel_loadaddr} 0xFF ${kernel_size}
tftpboot ${kernel_loadaddr} ${kernel_image}
nand device 0
nand erase.part nand-linux
nand write ${kernel_loadaddr} nand-linux ${filesize}
```

10. Update the devicetree image:

```
mw.b ${devicetree_loadaddr} 0xFF ${devicetree_size}
tftpboot ${devicetree_loadaddr} ${devicetree_image}
nand device 0
nand erase.part nand-device-tree
nand write ${devicetree_loadaddr} nand-device-tree ${filesize}
```

11. Update the rootfs image:

```
mw.b ${ramdisk_loadaddr} 0xFF ${ramdisk_size}
tftpboot ${ramdisk_loadaddr} ${ramdisk_image}
nand device 0
nand erase.part nand-rootfs
nand write ${ramdisk_loadaddr} nand-rootfs ${filesize}
```

12. Stop the U-Boot autoboot.
13. Trigger NAND Flash boot with:

```
run nandboot
```

Note: Note that step 8 to 11 can be invoked independently.

4.1.4 USB Drive

The Xilinx family devices cannot boot directly from a USB Drive. The FSBL and the U-Boot have to be started from SD Card (MMC) or QSPI Flash. Please refer to *SD Card (MMC)* or *QSPI Flash* in order to boot U-Boot from SD Card or QSPI Flash. When U-Boot is booted it can load and boot the Linux system stored on the USB Drive.

In order to deploy images and boot the Linux system from a USB Drive, do the following steps:

1. Create a FAT formatted partition as the first partition on the drive. This partition should have at least 16 MiB.
2. Copy `uimage`, `devicetree.dtb`, `uramdisk` and `uboot.scr` from the build environment output directory to the FAT formatted partition.
3. Insert the USB drive to USB port of the board.
4. Set the board to boot from the SD Card (MMC) or QSPI Flash (refer to the board User Manual).
5. Stop the U-Boot autoboot.
6. Trigger USB boot with:

```
run usbboot
```

4.2 Altera Family

4.2.1 SD Card (MMC)

In order to deploy images to an SD Card and boot from it, do the following steps:

1. Prepare the SD Card (refer to the *MMC Partitioning Guide*).
2. Record the preloader image to the unformatted partition of a SD Card (type a2):

- In Linux:

```
sudo dd if=preloader-mkpimage.bin of=/dev/sdX2
sync
# where X is the letter of the device
```

And Mount the BOOT and rootfs partitions.

- In Windows:

(a) Run the `Win32 Disk Imager` as administrator.

(b) Select the drive letter representing the SD Card in the device drop-down list.

(c) Open the `preloader-mkpimage.bin` file.

(d) Hit `Write`.

(e) Wait for completion and exit the application.

3. Copy `uimage`, `devicetree.dtb`, `fpga.rbf`, `u-boot.img` and `uboot.scr` from the build environment output directory to the BOOT partition (FAT formatted).
4. Unpack root file system onto the ext2 partition of the SD Card (rootfs partition). This has to be done as root.

```
sudo tar -xpf rootfs.tar -C /path/to/mmc/mountpoint

#. Unmount all partitions mounted from the SD Card.
```

5. Insert the card into the SD Card slot on the board.
6. Set the board to boot from the SD Card (refer to the board User Manual).
7. The board should boot the Linux system.

If one wants to manually trigger booting from a SD Card, the following command has to be invoked from the U-Boot command line:

```
run mmcboot
```

4.2.2 QSPI Flash

Table 4.3: Altera Family QSPI Flash Layout

Partition	Offset	Size
Preloader	0x0	0x40000
U-Boot image	0x60000	0x40000
FPGA Bitstream	0x100000	0x700000
U-Boot environment	0x800000	0x40000
Linux Device Tree	0x840000	0x40000
Bootscript	0x880000	0x40000
Linux kernel	0x8C0000	0x740000
JFFS2 Rootfs	0x1000000	0x3000000

In order to deploy images and boot from QSPI Flash do the following steps:

1. Setup an TFTP server on a host computer.
2. Connect the board and boot to the U-Boot (e.g. from a MMC card).
3. Connect an Ethernet cable to the device.
4. Connect a serial console to the device (e.g. using PuTTY or picocom).
5. Setup the U-Boot connection parameters:

```
setenv ipaddr 'xxx.xxx.xxx.xxx'
# where xxx.xxx.xxx.xxx is the board address
setenv serverip 'yyy.yyy.yyy.yyy'
# where yyy.yyy.yyy.yyy is the server (host computer) address
```

6. Copy uimage, devicetree.dtb, uboot.scr, rootfs.jffs2, u-boot.img, fpga.rbf.img and preloader-mkpimage.bin from the build environment output directory to the TFTP server directory
7. Update the preloader image:

```
mw.b ${preloader_loadaddr} 0xFF ${preloader_size}
tftpboot ${preloader_loadaddr} ${preloader_image}
sf probe
sf erase ${qspi_preloader_offset} ${preloader_size}
sf write ${preloader_loadaddr} ${qspi_preloader_offset} ${filesize}
```

8. Update the U-Boot image:

```
mw.b ${uboot_loadaddr} 0xFF ${qspi_uboot_erase_size}
tftpboot ${uboot_loadaddr} ${uboot_image}
sf probe
sf erase ${qspi_uboot_erase_offset} ${qspi_uboot_erase_size}
sf write ${uboot_loadaddr} ${qspi_uboot_offset} ${filesize}
```

9. Update the bitstream image:

```
mw.b ${bitstream_loadaddr} 0xFF ${bitstream_size}
tftpboot ${bitstream_loadaddr} ${bitstream_image}
sf probe
sf erase ${qspi_bitstream_offset} ${bitstream_size}
sf write ${bitstream_loadaddr} ${qspi_bitstream_offset} ${filesize}
```

10. Update the boot script image:

```
mw.b ${bootscript_loadaddr} 0xFF ${bootscript_size}
tftpboot ${bootscript_loadaddr} ${bootscript_image}
sf probe
sf erase ${qspi_bootscript_offset} ${bootscript_size}
sf write ${bootscript_loadaddr} ${qspi_bootscript_offset} ${filesize}
```

11. Update the Linux kernel:

```
mw.b ${kernel_loadaddr} 0xFF ${kernel_size}
tftpboot ${kernel_loadaddr} ${kernel_image}
sf probe
sf erase ${qspi_kernel_offset} ${kernel_size}
sf write ${kernel_loadaddr} ${qspi_kernel_offset} ${filesize}
```

12. Update the devicetree image:

```
mw.b ${devicetree_loadaddr} 0xFF ${devicetree_size}
tftpboot ${devicetree_loadaddr} ${devicetree_image}
sf probe
sf erase ${qspi_devicetree_offset} ${devicetree_size}
sf write ${devicetree_loadaddr} ${qspi_devicetree_offset} ${filesize}
```

13. Update the rootfs image:

```
mw.b ${rootfs_loadaddr} 0xFF ${rootfs_size}
tftpboot ${rootfs_loadaddr} ${rootfs_image}
sf probe
sf erase ${qspi_rootfs_offset} ${rootfs_size}
sf write ${rootfs_loadaddr} ${qspi_rootfs_offset} ${filesize}
```

14. Set the board to boot from the QSPI flash (refer to the board User Manual).
15. Reset the board.
16. The board should boot to the Linux system.

If one wants to manually trigger booting from the QSPI flash - following command have to invoked from the U-Boot command line:

```
run qspiboot
```

Note: Note that the steps from 7 to 13 can be invoked independently.

4.2.3 USB Drive

The Altera family devices cannot boot directly from a USB device. The preloader and the U-Boot have to be started from a MMC card or a QSPI flash. Please refer to *SD Card (MMC)* or *QSPI Flash* to boot the U-Boot from MMC or QSPI. When the U-Boot is booted it can load and boot the Linux system stored on the USB drive.

In order to deploy images and boot the Linux system from a USB drive do the following steps:

1. Create a FAT formatted partition as the first partition on the drive. This partition should have at least 16 MiB.
2. Create a ext2 formatted partition as the second partition on the drive. This partition should have at least 16 MiB.
3. Copy `uimage`, `devicetree.dtb` and `uboot.scr` from the build environment output directory to the FAT formatted partition.

4. Unpack root file system onto the ext2 partition of the USB drive. This has to be done as root.

```
sudo tar -xpf rootfs.tar -C /path/to/mmc/mountpoint
```

5. Insert the USB drive to the board.
6. Set the board to boot from the MMC card or QSPI flash (refer to the board User Manual).
7. Stop the U-Boot autoboot.
8. Trigger USB boot with

```
run usbboot
```

4.2.4 MMC Partitioning Guide

1. Insert card into the reader.
2. If the partitions were mounted - unmount them:

```
sudo umount /path/to/mountpoint
```

3. Run `fdisk` tool:

```
sudo fdisk /dev/sdX
# where X is the letter of the SD card
```

4. Within `fdisk` run the following commands:

```
# delete any existing partition repeat until all are deleted
d
# create a new primary partition
n
# choose primary
p
# set number to '2'
2
# leave default start sector, and set the size to 2 MiB
+2M
# change the partition type
t
# choose the second partition (may be chosen automatically if only one exists)
2
# set type to Altera Boot Partition
a2
# create a new primary partition
n
# set as primary
p
# set number to '1'
1
# leave default start sector, and set the size to 16MiB
+16M
# change the partition type
```

```
t
# choose the first partition
1
# set type to fat16
4
# create the third partition
n
# set as primary
p
# set number to '3'
3
# leave default start and end sector
# write changes to the disk
w
# leave 'fdisk' (might be happening automatically)
q
```

5. Format newly created partitions:

```
sudo mkfs.fat -n BOOT /dev/sdX1
sudo mkfs.ext2 -L rootfs /dev/sdX3
# where X is the letter of the SD card
```