AXEL

Solo / Dual / Quad ARM Cortex-A9 CPU Module

Axel Embedded Linux Kit (X*ELK*)

Quick Start Guide

<Page intentionally left blank>

Table of Contents

1 Preface

1.1 About this manual

This manual describes the AXEL Embedded Linux Kit (XELK) and serves as a quick guide for start working with the development kit.

1.2 Copyrights/Trademarks

Ethernet® is a registered trademark of XEROX Corporation.

All other products and trademarks mentioned in this manual are property of their respective owners.

All rights reserved. Specifications may change any time without notification.

1.3 Standards

DAVE Embedded Systems is certified to ISO 9001 standards.

1.4 Disclaimers

DAVE Embedded Systems does not assume any responsibility for availability, supply and support related to all products mentioned in this manual that are not strictly part of the AXEL CPU modules, the AXELEVB-Lite carrier board and the Dacu carrier board.

AXEL CPU Modules are not designed for use in life support appliances, devices, or systems where malfunctioning of these products can reasonably be expected to result in personal injury. **DAVE Embedded Systems** customers who are using or selling these products for use in such applications do so at their own risk and agree to fully indemnify **DAVE Embedded Systems** for any damage resulting from such improper use or sale.

1.5 Warranty

AXEL SOMs, AXELEVB-Lite and Dacu are guaranteed against defects in material and workmanship for the warranty period from the shipment date. During the warranty period, **DAVE Embedded Systems** will at its discretion decide to repair or replace defective products. Within the warranty period, the repair of products is free of charge provided that warranty conditions are observed.

The warranty does not apply to defects resulting from improper or inadequate maintenance or handling by the customer, unauthorized modification or misuse, operation outside of the product's specifications or improper installation or maintenance.

DAVE Embedded Systems will not be responsible for any defects or damages to other products not supplied by **DAVE Embedded Systems** that are caused by a faulty AXEL module, AXELEVB-Lite or Dacu.

1.6 Technical Support

We are committed to making our products easy to use and will help customers use our CPU modules in their systems.

Technical support is delivered through email for registered kits owners. Support requests can be sent to support-axel@dave.eu. Software upgrades are available for download in the restricted download area of **DAVE Embedded Systems** web site: http://www.dave.eu/reserved-area. An account is required to access this area.

Please refer to our Web site at http://www.dave.eu/dave-cpu-module-imx6-axel.html for the latest product documents, utilities, drivers, Product Change Notices, Board Support Packages, Application Notes, mechanical drawings and additional tools and software.

1.7 Related documents

Document	Location
DAVE Embedded Systems Developers Wiki	http://wiki.dave.eu/index.php/Main_Page
i.MX6 Application Processor Reference Manual	http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6DQRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation
Freescale I.MX community webiste	https://community.freescale.com/community/imx
Freescale L3.10.17-1.0.0 documentation bundle	https://www.freescale.com/webapp/Download?colCode=L3.10.17_1.0.0_IMX6QDLS_BUNDLE&appType=license&location=null&WT_TYPE=Board%20Support%20Packages&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=gz&WT_ASSET=Downloads&fileExt=.gz&Parent_nodeId=1337637154535695831062&Parent_pageType=product
AXEL main page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/Category:Axel
AXEL Hardware Manual	http://www.dave.eu/sites/default/files/files/axel-hm.pdf
AXEL Software Manual	http://wiki.dave.eu/index.php/Software_Manual_(Axel)
AXELEVB-Lite page on **DAVE Embedded**	http://wiki.dave.eu/index.php/AxelEVB-Lite

Document	Location
Systems Developers Wiki	
Dacu User's Guide	Provided with kit documentation
Building Embedded Linux Systems By Karim Yaghmour.	This book covers all matters involved in developing software for embedded systems. It is not a reference guide, but it provides a complete and exhaustive overview that helps the developer save a lot of time in searching for such information on the Internet
Training and Docs sections of Free Electrons website.	Brief but still exhaustive overview of the Linux and Embedded Linux world.

Tab. 1: Related documents

1.8 Conventions, Abbreviations, Acronyms

Abbreviation	Definition
BTN	Button
DVDK	Dave Virtual Development Kit
EMAC	Ethernet Media Access Controller
GPI	General purpose input
GPIO	General purpose input and output
GPO	General purpose output
LTIB	Linux Target Image Builder
OVA	Open Virtualization Archive
PCB	Printed circuit board
PMIC	Power Management Integrated Circuit
PSU	Power supply unit
RTC	Real time clock
SOC	System-on-chip

Abbreviation	Definition
SOM	System-on-module
WDT	Watchdog
XELK	AXEL Embedded Linux Kit

Tab. 2: Abbreviations and acronyms used in this manual

Revision History

Version	***Date***	***Notes***
1.0.0	November 2013	First official release
1.0.1	January 2014	Released with XELK 1.1.0 Minor fixes
1.0.2	May 2014	Added support for AXELLite SOM Minor fixes Released with XELK 1.2.0
1.0.3	November 2014	Released with XELK 2.0.0
1.0.4	April 2015	Released with XELK 2.1.0
1.0.5	February 2016	Minor fixes Released with XELK 2.2.0

2 Introduction

2.1 AXEL SOM

AXEL is the new top-class Solo/Dual/Quad core ARM Cortex-A9 CPU module by **DAVE Embedded Systems**, based on the recent Freescale i.MX6 application processor.

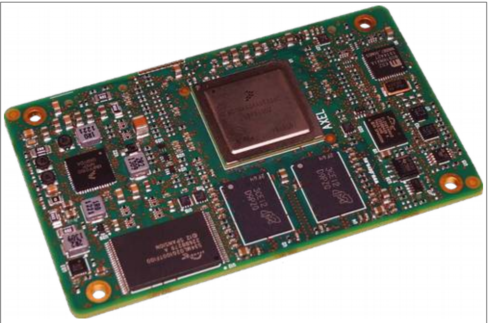


Fig. 1: Axel Ultra CPU module

Thanks to AXEL, customers have the chance to save time and resources by using a compact solution that permits to reach scalable performances that perfectly fits the application requirements avoiding complexities on the carrier board.

Fig. 2: AXEL-LITE – Solo / Dual / Quad core ARM Cortex A9

The use of this processor enables extensive system-level differentiation of new applications in many industry fields, where high-performance and extremely compact form factor (85mm x 50mm) are key factors. Smarter system designs are made possible, following the trends in functionalities and interfaces of the new, state-of-the-art embedded products. AXEL offers great computational power, thanks to the rich set of peripherals, the Scalable ARM Cortex-A9 together with a large set of high-speed I/Os (up to 5GHz).

AXEL enables designers to create smart products suitable for harsh mechanical and thermal environments, allowing the development of high computing and reliable solutions. Thanks to the tight integration between the ARM Core-based processing system, designers are able to share the application through the multi-core platform and/or to divide the task on different cores in order to match with specific application requirements (AMP makes possible the creation of applications where RTOS and Linux work together on different cores).Thanks to AXEL, customers are going to save time and resources by using a powerful and scalable compact solution, avoiding complexities on the carrier PCB.

AXEL is designed and manufactured according to **DAVE Embedded Systems** Ultra Line specifications, in order to guarantee premium quality and technical value for customers who require top performances and flexibility. AXEL is suitable for high-end applications such as medical instrumentation, advanced communication systems, critical real-time operations and safety applications.

For further information, please refer to AXEL Hardware Manual.

2.2 XELK

AXEL Embedded Linux Kit (XELK for short) provides all the necessary components required to set up the developing environment for:

- building the bootloader (U-Boot)
- building and running Linux operating system on AXEL-based systems
- building Linux applications that will run on the target

The heart of AXEL SOM is Freescale i.MX6 Solo/Dual/Quad core application processor. From a software point of view, Freescale supports this processor family through so-called Linux BSPs. The Linux BSP releases are published on a regular basis and the release packages have a reference code as L<Kernel_version>_<x.y.z> (eg: L3.10.17_1.0.3). For more details please refer to:

- http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX6Q&fpsp=1&tab=Design_Tools_Tab
- https://community.freescale.com/community/imx/content?filterID=contentstatus[published]~category[imx6all]&filterID=contentstatus[published]~objecttype~objecttype[document]

AXEL Embedded Linux Kit, in turn, is directly derived from L<Kernel_version>_<x.y.z> BSP versions. Hence XELK documentation often refers to L<Kernel_version>_<x.y.z> resources.

DAVE Embedded Systems adds to the latest BSP from Freescale the customization required to support the AXEL platform. For this reason most of the documentation provided by Freescale remains valid for the XELK development kit. However, some customization is required, in particular at bootloader and Linux kernel levels.

XELK 2.0.0 introduced support for the **Yocto** build system, an open source collaboration project that provides templates, tools and methods to help creating custom Linux-based systems for embedded products. It is derived from OpenEmbedded, but it provides a less steep learning curve, a graphical interface for Bitbake and very good documentation. **DAVE Embedded Systems** provides the so-called recipes/meta-repositories required to build all the XELK software components (bootloader, kernel and root file system) with the **Yocto** build system. For further details, please refer to https://wiki.yoctoproject.org/wiki/FAQ.

2.2.1 Kit Contents

The following table lists the XELK components:

Component	Description
	AXEL ULTRA **or** AXEL LITE SOM CPU: Freescale i.MX6
	AXELEVB-Lite Carrier board
	DACU Carrier board
	Ampire AM-800480STMQW 7” 800x480 LCD display LVDS interface
	AC/DC Single Output Wall Mount adapter Output: +12V – 2.0 A
	MicroSDHC card with SD adapter and USB adapter

2.2.2 XELK Release Notes

2.2.2.1 Version 1.0.0

- First official release

2.2.2.2 Version 1.1.0

- Minor update that adds support for more peripherals: NAND, RTC, I²C, SPI
- Touch screen works properly
- CAN works @ 1Mbps
- The system can boot from SD

2.2.2.3 Version 1.2.0

- Added support for AXEL LITE SOMs
- Bug fixes and minor changes

2.2.2.4 Version 2.0.0

- Updated u-boot and kernel versions
- Bug fixes and minor changes
- Added support for Yocto 1.5

2.2.2.5 Version 2.1.0

- Updated u-boot and kernel versions
- Added ConfigID support (http://wiki.dave.eu/index.php/ConfigID_and_UniqueID)
- Bug fixes and minor changes

2.2.2.6 Version 2.2.0

- Added splash screen support in U-Boot
- Updated u-boot and kernel versions
- Bug fixes and minor changes

2.2.2.7 Releases history

	XELK Version					
Release number	1.0.0	1.1.0	1.2.0	2.0.0	2.1.0	2.2.0
Status	Released	Released	Released	Released	Released	Released
Release date	November 2013	January 2014	May 2014	November 2014	April 2015	January 2016
Release type	Major (see 2.2.2.8)	Maintenance (see 2.2.2.8)	Maintenance (see 2.2.2.8)	Major (see 2.2.2.8)	Maintenance (see 2.2.2.8)	Maintenance (see 2.2.2.8)
Release notes	Version 1.0.0	Version 1.1.0	Version 1.2.0	Version 2.0.0	Version 2.1.0	Version 2.2.0
SOM PCB version	AXEL Ultra: CS030713	AXEL Ultra: CS030713A	AXEL Ultra: CS030713A AXEL Lite: CS335013A	AXEL Ultra: CS030713A AXEL Lite: CS335013A	AXEL Ultra: CS030713B AXEL Lite: CS335013B	AXEL Ultra: CS030713B AXEL Lite: CS335013B
Supported carrier boards	AXELEVB-Lite Dacu	AXELEVB-Lite Dacu	AXELEVB-Lite Dacu	AXELEVB-Lite Dacu	AXELEVB-Lite Dacu	AXELEVB-Lite Dacu
U-Boot version	2013.10-xelk-1.0.0	2013.10-xelk-1.1.0	2013.10-xelk-1.2.0	2013.04-xelk-2.0.0	2013.04-xelk-2.1.0	2013.04-xelk-2.2.0
Linux version	3.0.35-xelk-1.0.0	3.0.35-xelk-1.1.0	3.0.35-xelk-1.2.0	3.10.17-xelk-2.0.0	3.10.17-xelk-2.1.0	3.10.17-xelk-2.2.0
Drivers	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1 NAND RTC I²C SPI	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1 NAND RTC I²C SPI	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1 NAND RTC I²C SPI Video Input (MIPI)	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1 NAND RTC I²C SPI Video Input (MIPI)	SPI NOR Flash (boot) UART debug (2-wire) USB Host USB OTG SD/MMC1 CAN Touch screen controller EMAC SATA HMDI LVDS1 NAND RTC I²C SPI Video Input (MIPI)

	XELK Version					
					PCIe ConfigID	PCIe ConfigID Splash screen
Freescale BSP version	L3.0.35-4.1.0	L3.0.35-4.1.0	L3.0.35-4.1.0	L3.10.17-1.0.0	L3.10.17-1.0.0	L3.10.17-1.0.3
Graphic Libraries	Qt 4.8	Qt 4.8	Qt 4.8	Qt 5.3.2	Qt 5.3.2	Qt 5.3.2
Build tool	LTIB	LTIB	LTIB	Yocto 1.5	Yocto 1.5	Yocto 1.5

2.2.2.8 Release type

XELK release type can be:

- **Major**, when substantial changes are applied to the BSP (eg: major kernel version upgrades) or to the development kit (eg: new features, build system updates, ..). This usually means that a new DVDK is created for the XELK release.
- **Maintenance**, when minor updates and bug fixes are introduced. This usually means that the DVDK remains the same provided with the previous major version, and only an update of the source tree repositories (and the tftp binaries) is required.

As an example, XELK 2.2.0 is a maintenance release, so it provides the DVDK released with the 2.0.0 major release; customers can easily upgrade to the 2.2.0 release by updating the software components as described in Section 3.5.6.

2.2.2.9 Known limitations

The following table reports the known limitations of the latest XELK version, which will be solved for the next releases of the development kit:

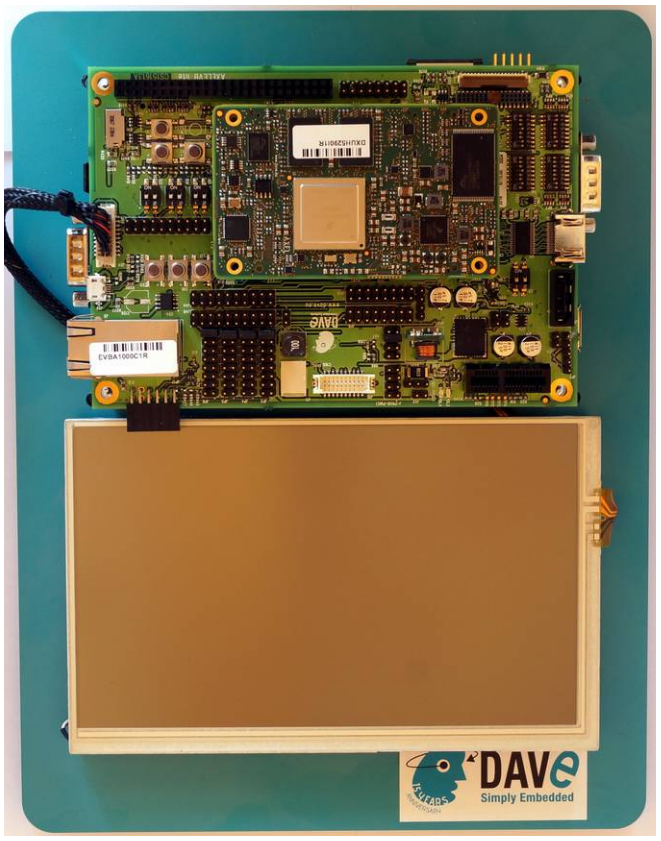
Issue	Description
USB OTG	Verified in Host and Device modes
Reboot from software	Rebooting the system from software (eg: launching the reboot command from Linux user space) can lead to a system lock. To solve it, reset the board with the dedicated button (S10)
Ethernet	10 Mbps connections have not been tested

3 XELK Quick Start

This chapter describes how to quickly start working with the XELK kit. The following paragraphs will guide you through the setup and installation procedures.

3.1 Unboxing

Once you've received the kit, please open the box and check the kit contents with the packing list included in the box, using the table on chapter 2.2.1 as a reference. The hardware components (SOM, carrier boards and display) are pre-assembled, as shown in the picture below:



3.2 Hardware setup

This section describes how to quick start an AXEL system composed of a AXEL SOM plugged into the AXELEVB-Lite and then mounted on the DACU carrier board, provided that it is programmed according to the XELK configuration.

The MicroSD provided with the XELK can be used to boot the system, since it contains a bootable partition (mmcblk0p1) and a root file system partition (mmcblk0p2).

1. insert the MicroSD card provided with the development kit into the MicroSD slot
2. connect the 12Vcc power supply to JP2 on the DACU board
3. (optional) connect a serial cable between the J25[1] connector on the DACU board and PC COM port through a NULL-modem (https://en.wikipedia.org/wiki/Null_modem) cable (not provided)
4. (optional) start your favorite terminal software on PC; communication parameters are:

Parameter	Value
Baud rate	115200 bps
Data bits	8
Stop bits	1
Parity	None

5. (optional) to connect the system to Ethernet LAN, please plug cable on connector J6 connector of the AXELEVB-Lite

The system is configured to boot automatically from the SD card when powered up.

3.3 First boot

Once power has been applied, U-Boot bootloader will be

1 For the previous versions of the AXELEVB LITE hardware (PCB rel. CS151613), the serial console is available on the J28 DB9 connector.

executed from the SPI NOR flash, and the debug messages will be printed on the serial console. U-Boot automatically runs the autoboot macro, that loads the kernel and launches it with the options for mounting the root file system from the mmcblk0p2 partition. At the end of the boot process, a demo application is launched and you can interact with the system using the touchscreen.

Moreover, the Linux shell is available on the serial console. Lastly, both telnet and ssh services are available to connect to the system through the network.

Please refer to Appendix 6.3 for an example of the boot messages.

3.4 Selecting boot device

The boot device is the one used to load U-Boot and is selected through S5-S9 dip switches configuration.

3.4.1 Boot from SD

Dip switch	1	2	3	4	5	6	7	8
S5	ON	OFF	ON	ON	ON	ON	OFF	OFF
S6	OFF	OFF	ON	ON	OFF	ON	ON	ON
S7	OFF	OFF	ON	ON	ON	ON	ON	ON
S8	ON	OFF	ON	ON	OFF	ON	ON	ON
S9	OFF	OFF						

3.4.2 Boot from SPI NOR flash (XELK default)

Dip switch	1	2	3	4	5	6	7	8
S5	ON	ON	OFF	OFF	ON	ON	OFF	OFF
S6	OFF	OFF	ON	ON	OFF	ON	ON	ON
S7	OFF	OFF	ON	OFF	OFF	OFF	ON	OFF
S8	ON	OFF	ON	ON	OFF	ON	ON	ON
S9	OFF	OFF						

3.5 DVDK installation

DAVE Embedded Systems Virtual Development Kit is a virtual machine, based on Oracle VirtualBox that allows developers to start using **DAVE Embedded Systems** platform without wasting time in installing the development environment. The Virtual Machine comes with all the development tools and source code (pre-configured), and requires only a minimal setup by the end user (usually only to adapt network interface to the user environment).

DVDK can also be converted, easily, into a physical environment, for example to increase speed on slower machines. Please note that DVDK can be used also with VMWare.

Please refer to DVDK page (http://wiki.dave.eu/index.php/Category:DVDK) on **DAVE Embedded Systems** Developer's Wiki for further information.

3.5.1 DVDK features

- VirtualBox virtual machine (.OVA archive)
- Based on Lubuntu 12.04 LTS (64-bit version)
- Pre-installed VirtualBox Guest Additions
- LXDE desktop environment available
- Boot disk with pre-installed Lubuntu Linux 12.04.2 LTS and pre-configured basic Linux services (TFTP, NFS, ...)
- Secondary disk[2] containing source code and tools:
  - Bootloader (u-boot) source tree cloned from **DAVE Embedded Systems** public git repository
  - Linux kernel source tree cloned from **DAVE Embedded Systems** public git repository
  - External pre-built toolchain

2 Please note that the secondary disk is not automatically mounted at DVDK boot. It must be mounted manually using a dedicate script, as described in Section 3.5.5 at point 4.

  - Yocto bsp for AXEL
  - Pre-installed Yocto-based root file systems with setup scripts, makefiles, example applications, ...
- Administrator account (dvdk) with autologin. Please note that the user account credentials are provided with the development kit (you can find them into the `README` file contained in the `sw/dvdk` folder of the kit distribution)

3.5.2 MicroSD contents

The microSD provided with XELK provides:

- A bootable partition (**mmcblk0p1**, **vfat**) containing:
  - binary images (U-Boot, Linux and device tree images), up to date with the latest XELK release
  - XELK documentation
  - DVDK virtual machine image (in a compressed .OVA archive, see the `sw/dvdk/README` file)
- XELK root file system partition (**mmcblk0p2**, **ext3**)

XELK contains all the required software and documentation to start developing Linux application on the AXEL platform.

3.5.3 Extracting the .OVA file

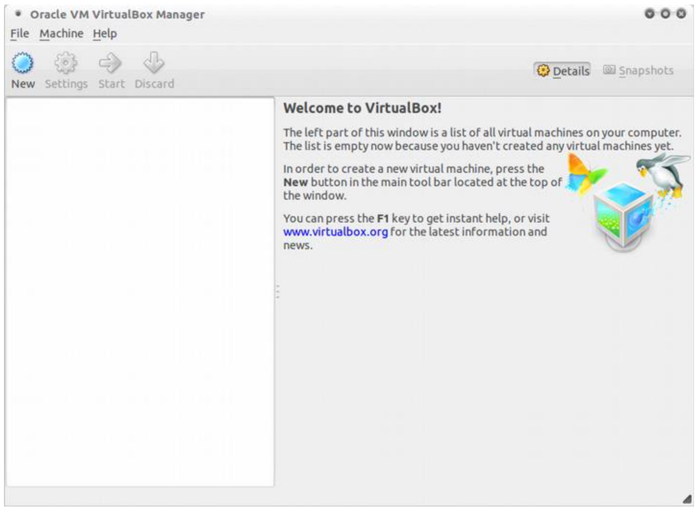
The DVDK image is archived in the `sw/dvdk` directory of the SD card using the open source 7z format. Please use an archive manager (eg: http://www.7-zip.org/) with support for .7z archives to extract the virtual machine image file. To extract it, open the first part of the compressed archive (XELK 2.0.0.ova.7z.001) and launch the extract command.

Please note that the 7z program for Windows is included into the dvdk directory. For Linux users: 7z is available for the most common distros. Please refer to your distribution software manager.

3.5.4 Importing the virtual machine

XELK provides a virtual machine image as a .OVA file, which is a virtual application exported in Open Virtualization Format (OVF). Please find below the instructions for importing the virtual machine into Virtualbox:

1. Start the Oracle VM VirtualBox Manager



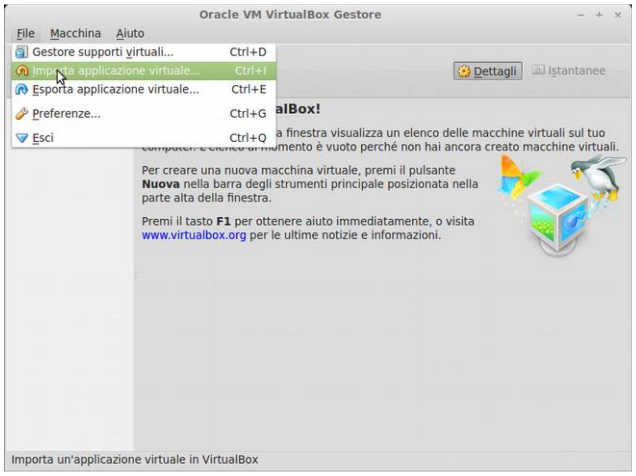
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

2. Click on File and select "Import Virtual Application", then click on "Open Virtual Application"


Oracle VM VirtualBox Gestore
File
Macchina
Aiuto
Gestore supporti virtuali... Ctrl+D
Importa applicazione virtuale... Ctrl+I
Esporta applicazione virtuale... Ctrl+E
Preferenze... Ctrl+G
Esci Ctrl+Q
Dettagli
Istantanee

alBox!

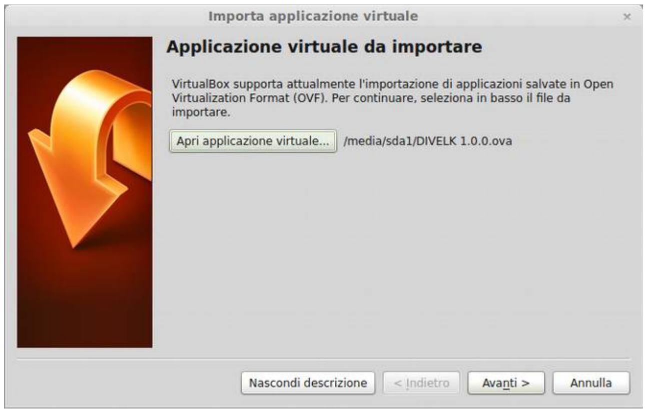
a finestra visualizza un elenco delle macchine virtuali sul tuo computer. L'elenco al momento è vuoto perché non hai ancora creato macchine virtuali.

Per creare una nuova macchina virtuale, premi il pulsante **Nuova** nella barra degli strumenti principale posizionata nella parte alta della finestra.

Premi il tasto **F1** per ottenere aiuto immediatamente, o visita www.virtualbox.org per le ultime notizie e informazioni.

Importa un'applicazione virtuale in VirtualBox

3. Navigate your file system and select the .ova file provided with the XELK

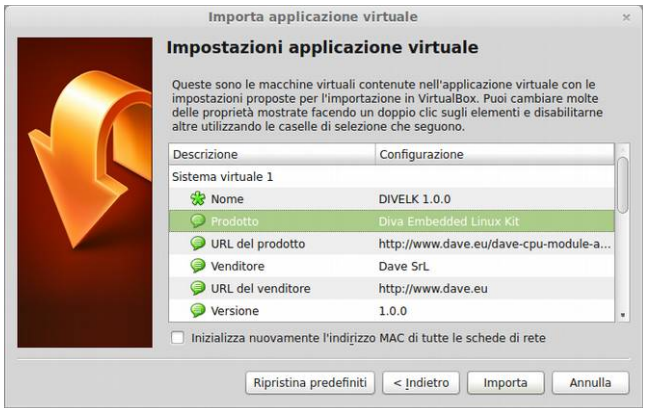


Applicazione virtuale da importare

VirtualBox supporta attualmente l'importazione di applicazioni salvate in Open Virtualization Format (OVF). Per continuare, seleziona in basso il file da importare.

/media/sda1/DIVELK 1.0.0.ova

4. Click "Next" and on the next window click on "Import"



Impostazioni applicazione virtuale

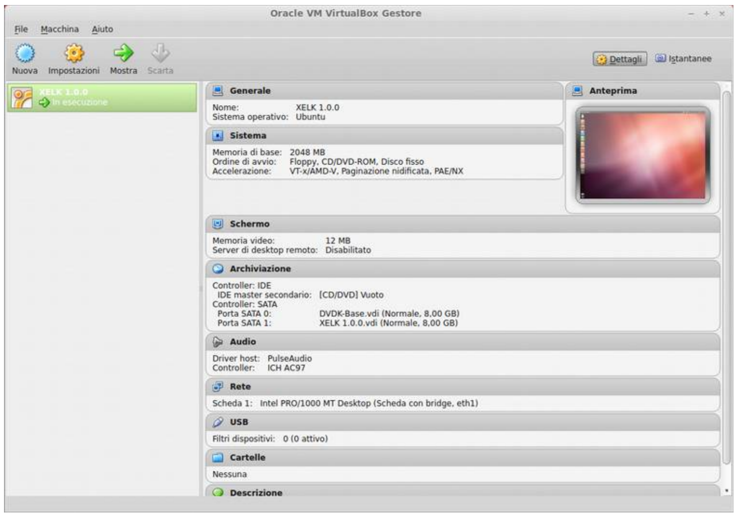
Queste sono le macchine virtuali contenute nell'applicazione virtuale con le impostazioni proposte per l'importazione in VirtualBox. Puoi cambiare molte delle proprietà mostrate facendo un doppio clic sugli elementi e disabilitarne altre utilizzando le caselle di selezione che seguono.

Descrizione	Configurazione
Sistema virtuale 1	
Nome	DIVELK 1.0.0
Prodotto	Diva Embedded Linux Kit
URL del prodotto	http://www.dave.eu/dave-cpu-module-a...
Venditore	Dave SrL
URL del venditore	http://www.dave.eu
Versione	1.0.0

☐ Inizializza nuovamente l'indirizzo MAC di tutte le schede di rete

3.5.5 Launching the virtual machine

1. Select the XELK virtual machine from the left panel and click the “Start” button (green arrow icon).



Generale
Nome: XELK 1.0.0
Sistema operativo: Ubuntu

Sistema
Memoria di base: 2048 MB
Ordine di avvio: Floppy, CD/DVD-ROM, Disco fisso
Accelerazione: VT-x/AMD-V, Paginazione nidificata, PAE/NX

Schermo
Memoria video: 12 MB
Server di desktop remoto: Disabilitato

Archiviazione
Controller: IDE
IDE master secondario: [CD/DVD] Vuoto
Controller: SATA
Porta SATA 0: DVDK-Base.vdi (Normale, 8,00 GB)
Porta SATA 1: XELK 1.0.0.vdi (Normale, 8,00 GB)

Audio
Driver host: PulseAudio
Controller: ICH AC97

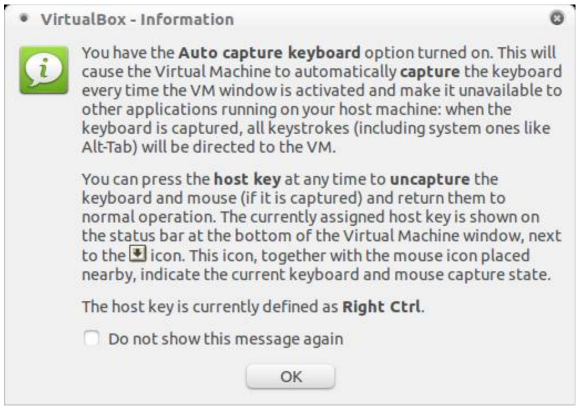
Rete
Scheda 1: Intel PRO/1000 MT Desktop (Scheda con bridge, eth1)

USB
Filtri dispositivi: 0 (0 attivo)

Cartelle
Nessuna

Descrizione

2. VirtualBox will open some message windows like the following, you can click "Ok" to close them



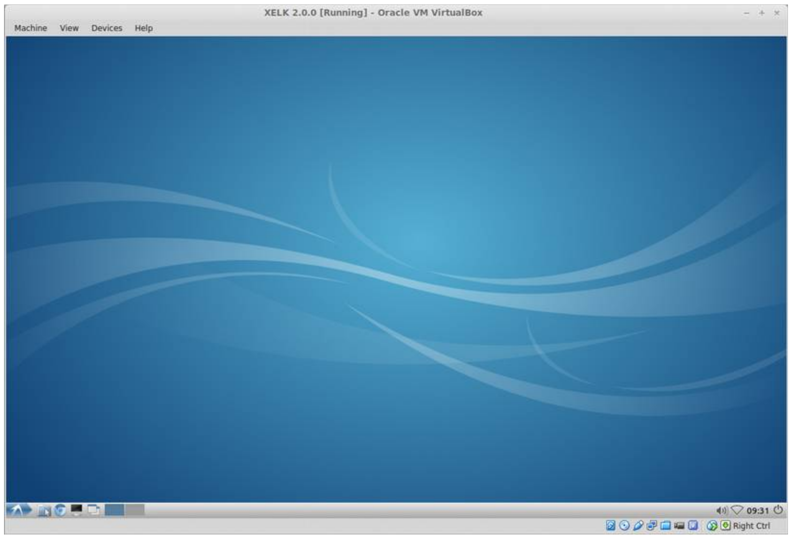
You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard every time the VM window is activated and make it unavailable to other applications running on your host machine: when the keyboard is captured, all keystrokes (including system ones like Alt-Tab) will be directed to the VM.

You can press the **host key** at any time to **uncapture** the keyboard and mouse (if it is captured) and return them to normal operation. The currently assigned host key is shown on the status bar at the bottom of the Virtual Machine window, next to the icon. This icon, together with the mouse icon placed nearby, indicate the current keyboard and mouse capture state.

The host key is currently defined as **Right Ctrl**.

☐ Do not show this message again

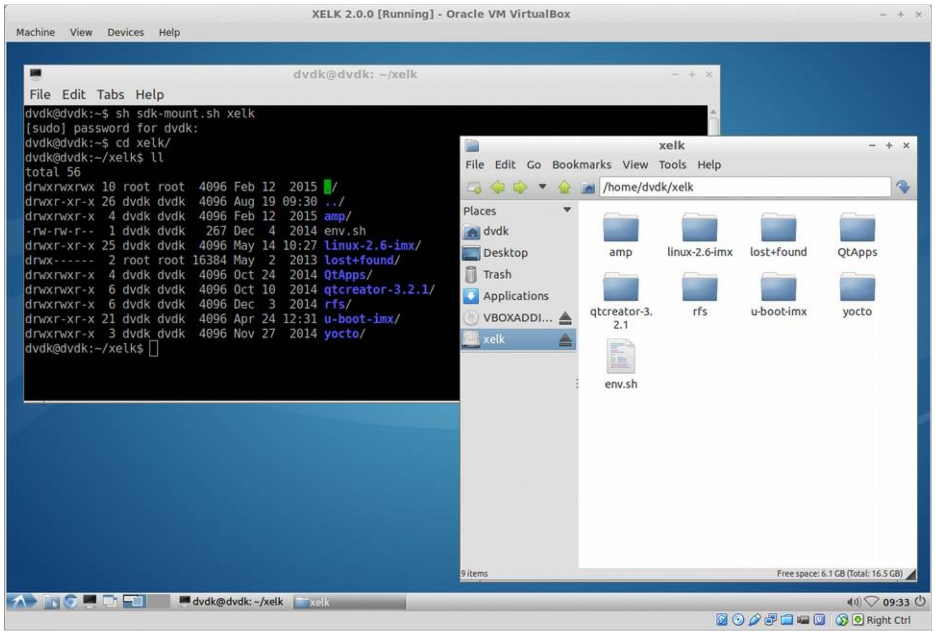
3. At the end of the boot process, the Ubunu desktop will be available. Please note that the user account credentials are provided with the development kit (you can find them into the '''README''' file contained in the '''dvdk''' folder of the kit distribution)


XELK 2.0.0 [Running] - Oracle VM VirtualBox
Machine View Devices Help
09:31
Right Ctrl

4. Mount the sdk disk launching the following commands from a shell terminal:

```
cd /home/dvdk

sh sdk-mount.sh xelk
```



```
dvdk@dvdk:~$ sh sdk-mount.sh xelk
[sudo] password for dvdk:
dvdk@dvdk:~$ cd xelk/
dvdk@dvdk:~/xelk$ ll
total 56
drwxrwxrwx 10 root root  4096 Feb 12  2015 ./
drwxr-xr-x 26 dvdk dvdk  4096 Aug 19 09:30 ../
drwxrwxr-x  4 dvdk dvdk  4096 Feb 12  2015 amp/
-rw-rw-r--  1 dvdk dvdk   267 Dec  4  2014 env.sh
drwxr-xr-x 25 dvdk dvdk  4096 May 14 10:27 linux-2.6-imx/
drwx------  2 root root 16384 May  2  2013 lost+found/
drwxrwxr-x  4 dvdk dvdk  4096 Oct 24  2014 QtApps/
drwxrwxr-x  6 dvdk dvdk  4096 Oct 10  2014 qtcreator-3.2.1/
drwxrwxr-x  6 dvdk dvdk  4096 Dec  3  2014 rfs/
drwxr-xr-x 21 dvdk dvdk  4096 Apr 24 12:31 u-boot-imx/
drwxrwxr-x  3 dvdk dvdk  4096 Nov 27  2014 yocto/
dvdk@dvdk:~/xelk$
```

5. Once logged in, the system could suggest to update the Virtualbox Guest Additions package. You can follow the on-screen instructions to easily

install the updated package.

6. Check if your keyboard layout matches the Ubuntu keyboard settings. You can change the keyboard layout selecting System->Preferences->Keyboard from the top panel menù.
7. Configure the Virtual Machine network interface, as described in this page: http://wiki.dave.eu/index.php/VirtualBox_Network_Configuration

3.5.6 Updating the XELK distributions

It's recommended to use the latest available XELK version. Once the DVDK is up and running, please do the following tasks, in particular if you are installing the DVDK of a major release and want to update to the latest maintenance release (for additional information, please refer to Section 2.2.2.8):

1. update the source code repositories, as described in 4.2.8.
2. update the kernel and device tree binary images in the /srv/tftp/xelk directory for booting through tftp. To do this, simply copy the uImage and device tree binaries from the microSD card provided with the kit to the /srv/tftp/xelk directory of the DVDK

For additonal information, please refer to section 4.2.7.

4 Develoment tools

4.1 Embedded Linux

When we talk in general about Embedded Linux[3], we refer to an embedded system running Linux operating system. As the reader probably knows, Linux was first developed on the PC platform, based on the famous x86 architecture. Typical embedded systems using an operating system (O.S. for short), are equipped with much lighter software. Recent hardware advances made these systems so powerful that now they can run a complex O.S. such as Linux. This choice has several benefits:

- The developer can count on a reliable and efficient software, developed and maintained by a large community all over the world
- The software is open-source, so developers have access to the whole source code
- Since Linux runs on many different platforms (x86, PowerPC, ARM, SuperH, MIPS etc.), applications are portable by definition
- There are a lot of open-source applications running on top of Linux that can easily be integrated in the embedded system
- Last but not least, there are no license fees.
- The typical Embedded Linux system is composed of:
- the bootloader – this software is run by the processor after exiting the reset state. It performs basic hardware initialization, retrieves the Linux kernel image (for example from a remote server via the TFTP protocol) and launches it by passing the proper arguments (command line and tags)
- the Linux kernel
- the root file system – this file system is mounted (which means "made available", "attached") by the kernel during the boot process on the root directory

3 An exhaustive description of this topic is beyond the scope of this document. We recommend reading specific documents, eg Building Embedded Linux Systems By Karim Yaghmour.

("/").

The typical developing environment for an Embedded Linux system is composed of a host machine and a target machine. The host is used by the developer to compile the code that will run on the target. In our case the target is obviously the AXEL module, while the host is assumed to be a PC running the Linux operating system. The Linux kernel running on the target can mount the root file system from different physical media. For example, during the software development, we strongly recommend using a directory exported via NFS by the host for this purpose (see the example configuration called net_nfs); however, for system deployed to the field, the root file system is usually stored into a flash device.

4.2 Software components

4.2.1 Toolchain

With the term "toolchain" we refer to the set of programs that allow the building of a generic application. For applications built to run on the same platform as the tool chain, we use a native toolchain. On the contrary, for applications built to run on a target architecture different from the host architecture, we use a cross-toolchain. In this case all the tools involved in this process are lead by the "cross-" prefix. So we talk about cross-compiler, cross-toolchain and so on. The cross-toolchain used to build U-Boot and the Linux kernel is the GNU toolchain for the ARM architecture built for x86 hosts. In other words, the toolchain runs on x86 machines but generates binaries for ARM processors. As for all the software compliant to the GPL license, it is released in source code. Thus the first thing to do to set up the developing environment should be building the cross-toolchain. This is not a trivial task, it takes a lot of time and hard disk space. To avoid this tedious task, we suggest use of a pre-built toolchain as explained in the following sections.

4.2.2 Bootloader

U-Boot is a very powerful boot loader and it became the "de facto" standard on non-x86 embedded platforms. The main tasks performed by U-Boot are:

- hardware initialization (external bus, internal PLL, SDRAM controller etc.)
- starting a shell on the serial port allowing the user to interact with the system through the provided commands
- automatic execution of the boot script (if any)

After system power-up, U-Boot prints some information about itself and about the system it is running on. Once the bootstrap sequence is completed, the prompt is printed and U-Boot is ready to accept user's commands. U-Boot manages an environment space where several variables can be stored. These variables are extremely useful to permanently save system settings (such as ethernet MAC address) and to automate boot procedures. This environment is redundantly stored in two physical sectors of boot flash memory; the default variables set is hard-coded in the source code itself. User can modify these variables and add new ones in order to create his/her own custom set of configurations. The commands used to do that are `setenv` and `saveenv`. This process allows the user to easily set up the required configuration. Once U-Boot prompt is available, it is possible to print the whole environment by issuing the command `printenv`.

For further information on use of U-Boot, please refer to http://www.denx.de/wiki/view/DULG/UBoot

4.2.3 Kernel

Linux kernel for i.MX processors is maintained primarily by Freescale. Periodically Freescale releases the so-called Linux BSP, which provides updated kernel sources.

Kernels released within XELK derive directly from Freescale Linux BSP kernels.

4.2.3.1 Linux Device Tree

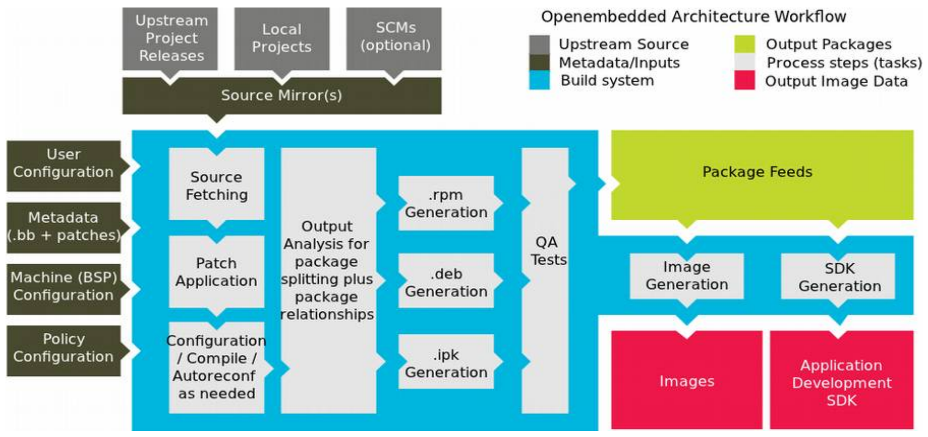
The Flattened Device Tree (FDT) is a data structure for describing the hardware in a system (for further information, please refer to http://elinux.org/Device_Tree). Device tree source code is stored into the `arch/arm/boot/dts/` directory.

4.2.4 Target root file system

The Linux kernel running on the target needs to mount a root file system. Building a root file system from scratch is definitively a complex task because several well known directories must be created and populated with a lot of files that must follow some standard rules. Again we will use pre-packaged root file systems that make this task much easier. However, using a build tool as Yocto, developers can build their own version of the root file system.

4.2.5 Yocto

The Yocto Project, hosted by the Linux Foundation, provides open source, high-quality infrastructure and tools to help developers create their own custom Linux distributions for any hardware architecture and across multiple market segments.



The Yocto Project is intended to simplify the work of the developers, providing a set of tools and components, including a highly configurable build system, that enables users to construct their own custom distributions, targeted for specific embedded devices. It is not, itself, a Linux distribution. Rather, it is capable of producing an image for a particular embedded device without dictating the composition of the Linux distribution actually built or the hardware architecture used.

The software components for the AXEL platform can be built using Yocto and the source trees released with the XELK 2.0.0 and above.

4.2.6 ConfigID

ConfigID is a new feature of **DAVE Embedded Systems** products. It's main purpose is providing an automatic mechanism for the identification of the product model and configuration.

With ConfigID, we aim at:

- completing the hardware configuration information that the software can't normally auto-detect (i.e. RAM chip version,...), implementing a dedicated reliable detect procedure
- when required, overriding the auto-detected hardware configuration information

When implemented, this mechanism allows for:

- initializing in the proper way the hardware platform, based on the specific features and parameters of the product, using a common software base (eg: a typical case is the SDRAM controller parameters, which must be configured by U-Boot depending on the particular memory chip, which can be different for the various SOM models)
- getting the complete hardware configuration (combining ConfigID with the information collectable at runtime) of a product deployed on the field

In simple words, model identification means the capability of reading a numerical code, stored in an available device (SOC's OTP , I2C EEPROM, 1-wire memories, protected NOR flash, etc.)

There are two ConfigIDs:

- SOM ConfigID: which reflects the characteristics of the SOM (stored on the SOM itself)

- Carrier Board (CB) ConfigID: which reflects the characteristics of the carrier board that hosts the SOM (stored on the carrier board itself and read by the SOM at boot time)

An additional attribute is UniqueID, which is a read-only code which univocally identifies a single product and is used for traceability.

DAVE Embedded Systems recommends to be up-to-date with Official SOM's BSPs for taking advantages of ConfigID/UniqueId features: this is the only required action.

4.2.6.1 ConfigID advantage

It allows U-Boot bootloader to be executed only with the correct configuration (if the U-Boot loaded is not the proper one, it may stop execution avoiding incorrect behaviour)

4.2.6.2 UniqueID advantage

It allows to trace univocally each individual SOMs and, in turn, all the on-the-field equipments.

4.2.7 Software components git repositories

XELK source trees for U-Boot and Linux kernel are provided as git repositories, so the user can immediately get access to the development trees and keep these components in sync and up to date with **DAVE Embedded Systems** repositories.

Component	Git Remote
U-Boot	git@git.dave.eu:dave/axel/u-boot-imx.git
Linux	git@git.dave.eu:dave/axel/linux-2.6-imx.git
Yocto BSP	git@git.dave.eu:dave/axel/axel-bsp.git

4.2.8 Updating the XELK git repositories

When the git account is enabled (please refer to section 4.2.8.1), the developer can synchronize the source tree entering the repository directory and launching the `git fetch` command.

Please note that git fetch doesn't merge the commits on the current branch. To do that, the developer should run

```
git merge origin/axel
```

or replace the ''fetch-merge'' process with a single `git pull` command. Please note that the recommended method is the ''fetch-merge'' process. For further information on Git, please refer to the official Git Documentation (http://git-scm.com/documentation).

4.2.8.1 RSA key generation

For getting access to the Git repositories, a ssh key is required. Please follow the procedure reported below to generate the RSA ssh key (we assume that the ssh package and the required tools are installed on the Linux development server):

- select your username (ad es. username@myhost.com)
- start a shell session on the Linux host
- enter the `.ssh` subdirectory into your home directory:

  ```
  cd ~/.ssh/
  ```
- launch the following command:

  ```
  ssh-keygen -t rsa -C "username@myhost.com"
  ```
- this command creates the files `~/.ssh/username@myhost.com` ('''private key''') and `~/.ssh/username@myhost.com.pub` ('''public key''')
- edit your `~/.ssh/config` adding the following lines:

  ```
  Host git.dave.eu
       User git
       Hostname git.dave.eu
       PreferredAuthentications publickey
       IdentityFile ~/.ssh/username@myhost.com
  ```

Please send the public key file to the following email support addresses

support-axel@dave.eu

with the request for the creation of a new public git account associated to your username. The support team will enable the account and send you a confirmation as soon as possible.

4.2.8.2 Checking the ssh connection to the git repositories

To check the ssh connection, you can use the following command:

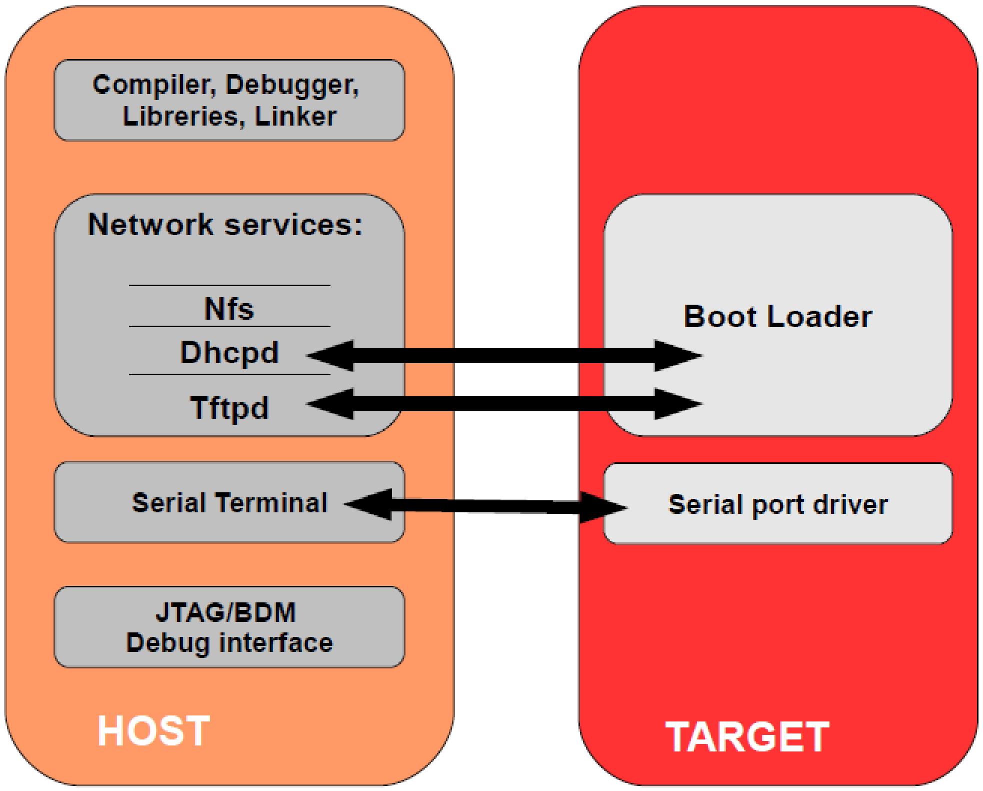
```
ssh -vT git@git.dave.eu
```

You'll get the log messages of the connection, which will help in case of problems.

4.3 Development environment

4.3.1 Introduction

The following figure show the typical development environment for an Embedded Linux system: it is composed of a host machine and a target machine.



The typical developing environment for an Embedded Linux system is composed of a host machine and a target machine. The host is used by the developer to (cross-)compile the code that is to run on the target. In our case the target is the AXEL CPU module, while the host is assumed to be a PC running the Linux operating system, either in a physical installation or as a virtual machine. The bootloader running on the target can download the Linux kernel image through the network (TFTP), as well as the u-boot binary images (useful when an update of the bootloader is required). Moreover, the Linux kernel running on the target is able to mount the root file system from different physical media, for example from a directory exported via Network File System (NFS) by the host. This strategy (kernel image and RFS retrieved from the network) saves time during the development phase, since no flash reprogramming or removable storage (SD, usb pen drives, external disks) is required to test new versions or updates of the software components.

4.3.2 The build system

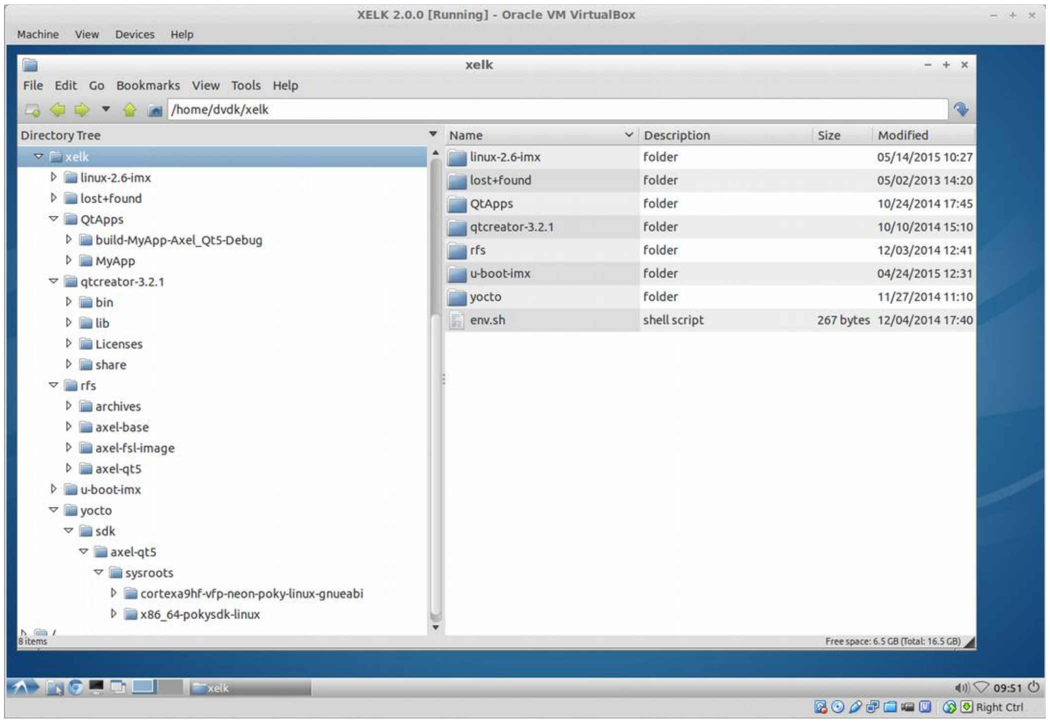
A build system is a set of source trees, Makefiles, patches, configuration files, tools and scripts that makes it easy to generate all the components of a complete embedded Linux system. A build system, once properly set up, automates the configuration and cross-compilation processes, generating all the required targets (userspace packages (libraries, programs), the kernel, the bootloader and root filesystem images) depending on the configuration. Some well known build systems are the following:

- OpenEmbedded (http://wiki.openembedded.net/index.php/Main_Page)
- Yocto (https://www.yoctoproject.org/)
- Buildroot (http://buildroot.uclibc.org)
- LTIB (http://ltib.org/)

For the Linux BSP release , Freescale officially supports Yocto as build system and therefore XELK 2.0.0 and above kits are based on Yocto.

4.3.3 Overview of the installed components

Once the virtual machine is running and the secondary disk is mounted, the actual development kit can be found into the directory `/home/dvdk/xelk`:


XELK 2.0.0 [Running] - Oracle VM VirtualBox
Machine View Devices Help
xelk
File Edit Go Bookmarks View Tools Help
Directory Tree
Right Ctrl

/home/dvdk/xelk

```
xelk
  linux-2.6-imx
  lost+found
  QtApps
    build-MyApp-Axel_Qt5-Debug
    MyApp
  qtcreator-3.2.1
    bin
    lib
    Licenses
    share
  rfs
    archives
    axel-base
    axel-fsl-image
    axel-qt5
  u-boot-imx
  yocto
    sdk
      axel-qt5
        sysroots
          cortexa9hf-vfp-neon-poky-linux-gnueabi
          x86_64-pokysdk-linux
```

Name	Description	Size	Modified
linux-2.6-imx	folder		05/14/2015 10:27
lost+found	folder		05/02/2013 14:20
QtApps	folder		10/24/2014 17:45
qtcreator-3.2.1	folder		10/10/2014 15:10
rfs	folder		12/03/2014 12:41
u-boot-imx	folder		04/24/2015 12:31
yocto	folder		11/27/2014 11:10
env.sh	shell script	267 bytes	12/04/2014 17:40

8 items

Free space: 6.5 GB (Total: 16.5 GB)

The `xelk` directory contains the following subdirectories:

- `linux-2.6-imx`: the Linux source tree
- `u-boot-am33x`: the U-Boot source tree
- `yocto`: the Yocto SDK installation directory
- `qtcreator-x.y.z`: pre-installed and pre-configured QtCreator IDE for Qt application development
- `rfs`: XELK provides three root file systems:
  - `axel-base`: minimal root file system with basic packages (`/home/dvdk/xelk/rfs/axel-base`)
  - `axel-fsl-image`: full root file system with lots of packages (including the gui application from Freescale) useful during the development phase (`/home/dvdk/xelk/rfs/axel-fsl-image`)
  - `axel-qt5`: root file system with Qt 5.3.2 libraries and examples (`/home/dvdk/xelk/rfs/axel-qt5`)
- `env.sh`: a bash script for setting the environment variables, containing the following lines:

```
export
PATH=~/xelk/yocto/sdk/axel-qt5/sysroots/x86_64-po
kysdk-linux/usr/bin/arm-poky-linux-gnueabi:$PATH
export ARCH=arm
export CROSS_COMPILE=arm-poky-linux-gnueabi-
```

4.3.4 Setting up the server environment

During development, user needs to interact with the target system. This section describes the tools that must be installed and configured on the host system for this purpose. Please note that all these tools are already installed and properly configured on the virtual machine image provided with the XELK.

4.3.4.1 TFTP Server

One of the most useful features of a bootloader during development is the capability to download the Linux kernel from the network. This saves a lot of time because developer doesn't have to program the image in flash every time he/she modifies it. U-Boot implements the TFTP protocol (see the tftp command), so the host system must be configured to enable the TFTP service. Installation and configuration of a TFTP server depends on the host Linux distribution.

The default DVDK tftp installation has `/srv/tftp` as work directory. A subdirectory dedicated to the image files associated to the XELK (`/srv/tftp/xelk`) is available, but developers can add their custom subdirectories when required.

4.3.4.2 NFS Server

One of the most important components of a Linux system is the root file system. A good development root file system provides the developer with all the useful tools that can help developers on their work. Such a root file system can become very big in size, so it's hard to store it in flash memory. User could split the file system in different parts, mounting them from different media (flash, network, usb...). But the most convenient thing is to mount the whole root file system from the network, allowing the host system and the target to share the same files. In this way, developers can quickly modify the root file system, even “on the fly” (meaning that the file system can be modified while the system is running). The most common way to setup a system like the one described is through NFS (Network File System). As for TFTP, installation and configuration depends on the host Linux distribution.

The default DVDK NFS installation is configured for sharing `/home` directory and all the subdirectories.

4.3.4.3 Pre-built toolchain

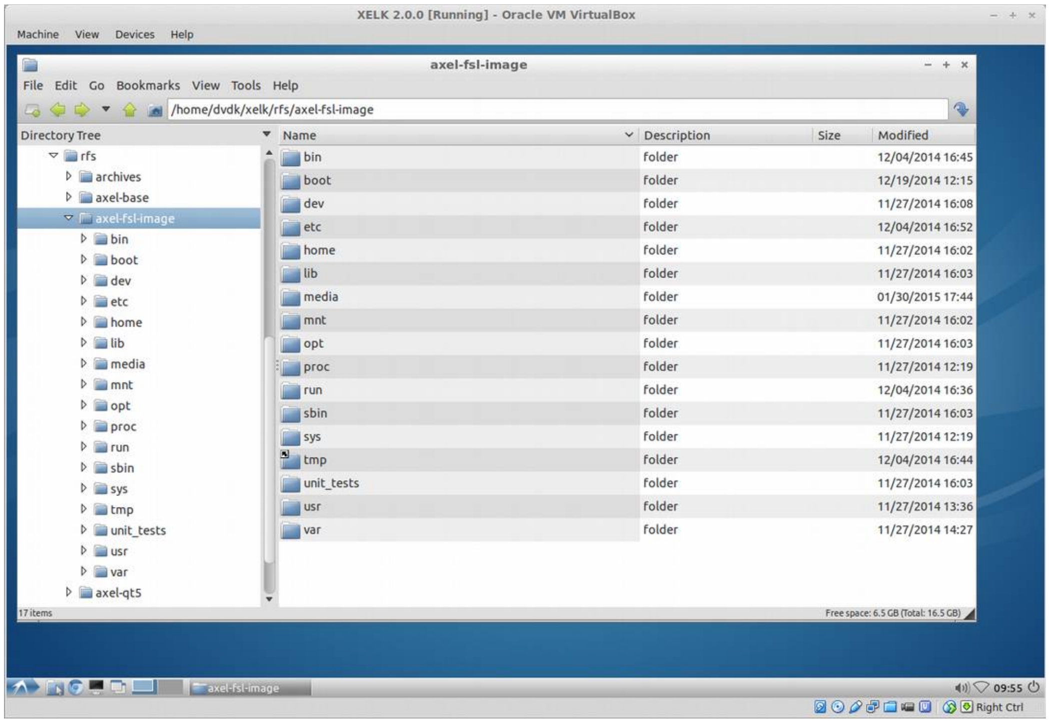
To start developing software for the AXEL platform, users need a proper toolchain, which can be pre-built or built-from-scratch. Building a toolchain from scratch is not a trivial task (though using a recent build system is easier than in the past), so the recommended approach consists in using a pre-built toolchain.

XELK provides the Poky toolchain built with the Yocto 1.5 version (**GCC version is 4.8.1**).

4.3.4.4 Pre-built root file system

Linux needs a root file system: a root file system must contain everything needed to support the Linux system (applications, settings, data, ..). The root file system is the file system that is contained on the same partition on which the root directory is located. The Linux kernel, at the end of its startup stage, mounts the root file system on the configured root device and finally launches the /sbin/init, the first user space process and "father" of all the other processes. An example of root file system is shown below:

For more information on the Linux filesystem, please refer to http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/.



/home/dvdk/xelk/rfs/axel-fsl-image

Name	Description	Size	Modified
bin	folder		12/04/2014 16:45
boot	folder		12/19/2014 12:15
dev	folder		11/27/2014 16:08
etc	folder		12/04/2014 16:52
home	folder		11/27/2014 16:02
lib	folder		11/27/2014 16:03
media	folder		01/30/2015 17:44
mnt	folder		11/27/2014 16:02
opt	folder		11/27/2014 16:03
proc	folder		11/27/2014 12:19
run	folder		12/04/2014 16:36
sbin	folder		11/27/2014 16:03
sys	folder		11/27/2014 12:19
tmp	folder		12/04/2014 16:44
unit_tests	folder		11/27/2014 16:03
usr	folder		11/27/2014 13:36
var	folder		11/27/2014 14:27

17 items

Free space: 6.5 GB (Total: 16.5 GB)

XELK provides pre-built root file systems, that can be used during the evaluation/development phase, since they contains the software packages for working with the AXEL platform.

XELK root file systems are built with the Yocto build system and are stored into the following directories:

- `/home/dvdk/xelk/rfs/axel-base`
- `/home/dvdk/xelk/rfs/axel-fsl-image`
- `/home/dvdk/xelk/rfs/axel-qt5`

4.4 Building the software components with Yocto

The build process creates an entire Linux distribution from source. The build process can be summarized as follows:

- Make sure that all the prerequisites are met
- Initialize the build environment, as described in 4.4.2.
- Optionally ensure the conf/local.conf configuration file, which is found in the Build Directory, is set up how you want it. This file defines many aspects of the build environment including the target machine architecture through the MACHINE variable, the development machine's processor use through the BB_NUMBER_THREADS and PARALLEL_MAKE variables, and a centralized tarball download directory through the DL_DIR variable.
- Build the image using the bitbake command. If you want information on BitBake, see the BitBake User Manual.

N.B. Since the XELK virtual machine is already configured to match all the requirements for using the Yocto build system, developers who wants to quickly build a Yocto image can directly go to section 4.4.3.

4.4.1 Prerequisites

The following prerequisites are required and only need to be done once. Please note that the XELK virtual machine is already configured to match all the requirements for using the Yocto build system.

Some generic development tools are required on the host Linux machine:

- git
- curl
- build-essential
- diffstat
- texinfo

- gawk
- chrpath
- ia32-libs (if the host machine is running a 64-bit OS)
- python-m2crypto

These packages can be installed with the following command:

```
sudo apt-get install git build-essential diffstat
texinfo gawk chrpath ia32-libs python-m2crypto
```

It is also recommended to switch the system shell from Ubuntu's standard dash to more universal bash:

```
$ sudo dpkg-reconfigure dash
```

4.4.2 Initializing the build environment

In the XELK, we have simplified the Yocto initialization phase, relying on the repo tool and on a Axel bsp git repository, so that the initialization can be completed with a few commands as reported below:

```
$ curl
http://commondatastorage.googleapis.com/git-repo-do
wnloads/repo > repo

$ chmod +x repo

./repo init -u
git@git.dave.eu:dave/axel/axel-bsp.git -b axel-dora

$ ./repo sync
```

4.4.3 Build the Yocto image

Please note that even building the basic root file system requires a few hours to complete the process on a mid-hi range desktop PC (4-6 cores, 8-12 GiB RAM), also depending on the Internet connection speed (all source are fetched from the network). Nearly 20GiB of disk space is required for the build. Moreover, building inside the DVDK adds some overhead, since the performances of a virtual machine are reduced if compared to the physical hardware. Thus, it's recommended to check the hardware capabilities of the host system and, when building with Yocto is required, developers should consider the following options:

- migrating the build system to a physical machine
- assuming that the host system has the required resources, extending the hardware capabilities of the default DVDK (eg: adding more cores and disk space)

Once completed the initialization phase, developers can launch the Yocto image build process with the following commands:

```
$ cd ~/xelk/axel-bsp
$ source source axel-bsp-init-env.sh build
$ bitbake axel-fsl-image-gui
```

Please note that three different images are available:

- axel-fsl-image-gui (includes the gui application from Freescale)
- axel-fsl-qt5-image (root file system with Qt 5.3.2 libraries and examples)
- base-rootfs-image (minimal root file system)

The resulting files (kernel, device tree and u-boot binaries, plus root file system in a .tar.gz archive) will then be available inside the `build/tmp/deploy/images/axel` directory.

4.5 Building the software components outside Yocto

4.5.1 Build/configure U-Boot

NOTE: before building U-Boot, please check for updates of the source code published in the git repositories, using the `git fetch/merge` or the `git pull` commands. For further details, please refer to section 4.2.7.

Assuming that you've configured the environment variables sourcing the `env.sh` script with the following command

```
dvdk@dvdk-vm:~/xelk$ source env.sh
```

enter the U-Boot sources directory (`~/xelk/u-boot-imx`) and run the following command:

```
dvdk@dvdk-vm:~/xelk/u-boot-imx$ make TARGET
```

where the available targets are listed in the following table:

SOC	Environment	Target	Notes
i.MX6Q	SPI NOR	mx6qaxel_spi	
i.MX6Q	MMC	mx6qaxel	

For example, to build U-Boot for SPI boot, please enter the following command:

```
dvdk@dvdk-vm:~/xelk/u-boot-imx$ make mx6qaxel_spi
```

Once the build process is complete, the binary images can be copied to the `/srv/tftp/xelk/` directory with the following command:

```
dvdk@dvdk-vm:~/xelk/u-boot-imx$ sudo cp
u-boot.imx /srv/tftp/xelk/
```

4.5.2 Build/configure Linux kernel

NOTE: before building Linux, please check for updates of the source code published in the git repositories, using the `git fetch/merge` or the `git pull` commands. For further details, please refer to section 4.2.7.

Assuming that you've configured the environment variables sourcing the `env.sh` script with the following command

```
dvdk@dvdk-vm:~/xelk$ source env.sh
```

enter the Linux sources directory (`~/xelk/linux-2.6-imx`) and run the following commands:

```
dvdk@dvdk-vm:~/xelk/linux-2.6-imx$ make
imx_v7_axel_defconfig
```

```
dvdk@dvdk-vm:~/xelk/linux-2.6-imx$ make
UIMAGE_LOADADDR=0x10008000 uImage imx6q-xelk-l.dtb
imx6q-xelk-h.dtb imx6q-xelk-l-2.0.0.dtb
```

The former command selects the default AXEL kernel configuration, while the latter builds the Linux binary image with the required U-Boot header and the binary device tree (.dtb) files for the following platforms:

- `imx6q-xelk-l.dtb`: XELK-L kit equipped with AXEL LITE SOM
- `imx6q-xelk-h.dtb`: XELK-H kit equipped with AXEL ULTRA SOM
- `imx6q-xelk-l-2.0.0.dtb`: XELK-L kit with hardware released with XELK version 2.0.0 and older

Default linux kernel configuration can be changed by using the standard *menuconfig*, *xconfig*, *gconfig* make targets.

Subsequent builds just require *uImage* make target to update the binary image.

Once the build process is completed, the kernel binary image is stored into the `linux-2.6-imx/arch/arm/boot/uImage` file, while the device tree binary is stored in:

XELK type	DTB file
XELK-L kit equipped with AXEL LITE SOM	`linux-2.6-imx/arch/arm/boot/dts/imx6q-xelk-l.dtb`
XELK-H kit equipped with AXEL ULTRA SOM	`linux-2.6-imx/arch/arm/boot/dts/imx6q-xelk-h.dtb`
XELK-L kit with hardware released with XELK version 2.0.0 and older	`linux-2.6-imx/arch/arm/boot/dts/imx6q-xelk-l-2.0.0.dtb`

These files can be copied to the `/srv/tftp/xelk/` directory with the following commands:

```
dvdk@dvdk-vm:~/xelk/linux-2.6-imx$ sudo cp arch/arm/boot/uImage /srv/tftp/xelk/
```

```
dvdk@dvdk-vm:~/dvdk/xelk/linux-2.6-imx$ sudo cp arch/arm/boot/dts/imx6q-xelk-*.dtb /srv/tftp/xelk/
```

4.5.3 Build a custom application

Some users may prefer to cross-compile their applications outside of the Yocto flow. It maybe specifically useful and easier for new projects in their prototyping and proof-of-concept stages or for any smaller applications in general. This way users don't have to worry about creating Yocto "recipes" for their applications and becoming familiar with the entire Yocto build system.

In order to cross-compile an application, written in C/C++, the cross-toolchain provided with the XELK is required.

Assuming that you've configured the environment variables sourcing the `env.sh` script with the following command

```
dvdk@dvdk-vm:~/xelk$ source env.sh
```

developers can write a simple "Hello world" application, called for example `hello.c`:

```
#include <stdio.h>

int main()

{

        printf("Hello world\n");

        return 0;

}
```

To cross-compile it:

```
$ arm-poky-linux-gnueabi-gcc -o hello hello.c
```

Then the executable file can be copied to the root file system, and executed from the AXEL system:

```
:~#./hello

Hello world
```

4.6 Programming the flash memory

4.6.1 Flashing binary images in NOR/NAND flash

4.6.1.1 U-Boot

```
run load
run spi_update
```

4.6.1.2 Linux kernel

```
run loadk
run spi_updatek – to store the kernel in SPI NOR flash
run nand_updatek – to store the kernel in NAND flash
```

4.6.1.3 Device tree

```
run loadfdt
run spi_updatefdt – to store the dtb in SPI NOR flash
run nand_updatefdt – to store the dtb in NAND flash
```

4.6.2 Flashing root file systems

The recommended procedure is the following:

- boot the system from network (NFS)
- format the flash MTD partition that have the size required to store the rfs
- create and mount the MTD partition, using the file system of choice (JFFS2, UBIFS, ...)
- transfer the root file system on the mounted partition (typically by uncompressing a tar.gz archive)

Please find below an example of the command sequence that can be used for an UBIFS file system (after booting the board via nfs). Please note that X is the mtd partition number (you can read the `/proc/mtd` file to find the partition mapping):

```
ubiformat /dev/mtdX
ubiattach /dev/ubi_ctrl -m X
ubimkvol /dev/ubi0 -N rootfs -m
```

```
mount -t ubifs /dev/ubi0_0 /mnt/ubifs
cd /mnt/ubifs
tar -zxvf <path to rfs archive>
cd /
sync
umount /mnt/ubifs
ubidetach /dev/ubi_ctrl -m X
reboot
```

4.7 Customizing the splash screen

Starting from version 2013.04-xelk-2.2.0, U-Boot for Axel SOMs provides support for a customizable splash screen. The following sections describe how to use this feature.

4.7.1 Customizing the splash screen

The splash screen image can be downloaded via tftp to a specific RAM address (`splashimage`) and then stored in the flash memory.

The following U-Boot environment variables are required:

- `splashimage`: RAM address where the BMP image is loaded. Please note that it must be a 32-bit aligned address with a 0x2 offset (eg: 0x20000002)
- `loadsplash`: comand for loading the BMP image from the storage device (e.g flash memory) to RAM. This command is automatically run by U-Boot at startup
- `splashpos`: image position (eg: splashpos=m,m, for centering the image)

4.7.2 Additional resources

For further details on splash screen support in U-Boot, please refer to:

- http://www.denx.de/wiki/DULG/UBootSplashScreen
- http://www.denx.de/wiki/DULG/UbootBitmapSupport

4.7.3 Splash image in NOR SPI flash

4.7.3.1 U-Boot variables

```
loadsplash=run spi_loadsplash
spi_loadsplash=sf probe; sf read ${splashimage} 0x800000 $
{splashsize}
splashfile=splash_image.bmp
splashimage=0x20000002
splashpos=m,m
splashsize=0x400000
```

```
loadsplashfile=tftpboot ${loadaddr} axel/${splashfile}
spi_updatesplash=sf probe; sf erase 0x800000 +${filesize}; sf
write ${loadaddr} 0x800000 ${filesize}
```

4.7.3.2 Commands

The following commands are used to store in NOR SPI flash a BMP image loaded via tftp:

```
run loadsplashfile
run spi_updatesplash
```

4.7.4 Splash image in NAND flash

4.7.4.1 U-Boot variables

```
mtdparts=mtdparts=gpmi-nand:8M(nand-uboot),1M(nand-env1),1M(n
and-env2),1M(nand-fdt),1M(nand-spare),8M(nand-kernel),4M(nand
-splash),-(nand-ubi)
loadsplash=run nand_loadsplash
nand_loadsplash=nand read ${splashimage} nand-splash
splashfile=splash_image.bmp
splashimage=0x20000002
splashpos=m,m
splashsize=0x400000
loadsplashfile=tftpboot ${loadaddr} axel/${splashfile}
nand_updatesplash=nand erase.part nand-splash; nand write $
{loadaddr} nand-splash ${filesize}
```

Please note that the NAND mtd partition for the splash image ('''nand-splash''') is defined using the `mtdparts` parameter, and then referenced by the `nand {erase,read,write}` commands.

4.7.4.2 Commands

The following commands are used to store in NAND flash a BMP image loaded via tftp:

```
run loadsplashfile
run nand_updatesplash
```

4.8 Building Qt applications

Qt Creator is pre-installed and pre-configured in the DVDK. This means that developers can quickly build

4.8.1 Launching Qt Creator

To launch QtCreator, simply open the main menù clicking on the LXDE icon on the bottom-left corner and select Programming->Qt Creator

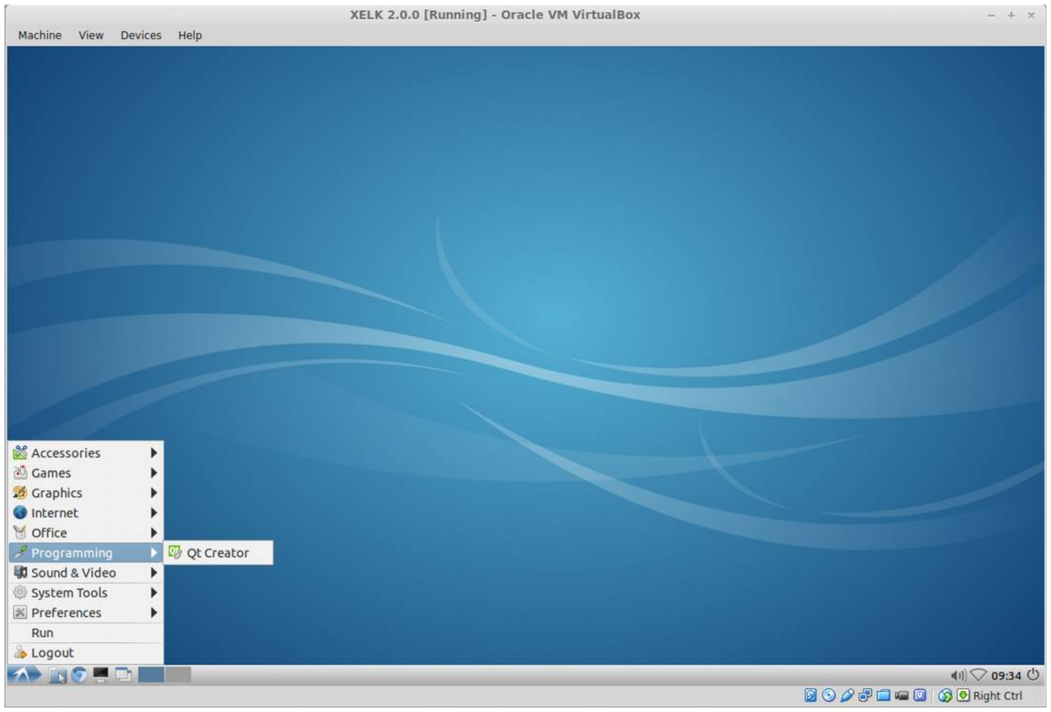
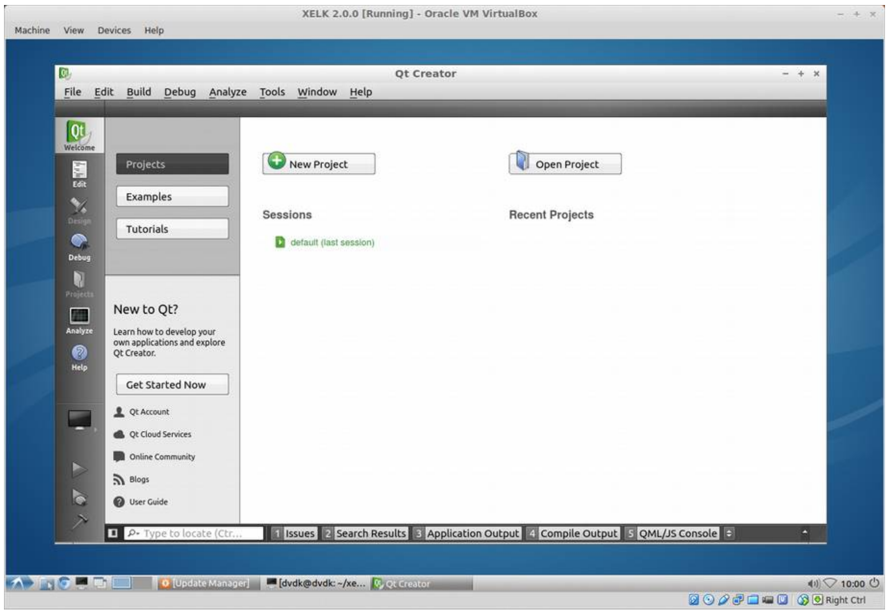
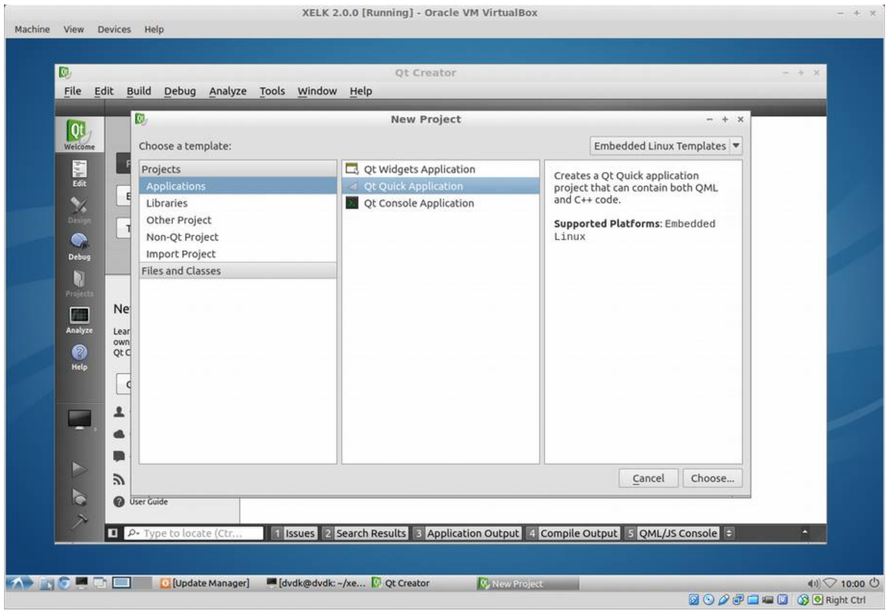


Fig. 3: Launching QtCreator

4.8.2 Building a QtQuick “Hello World!”

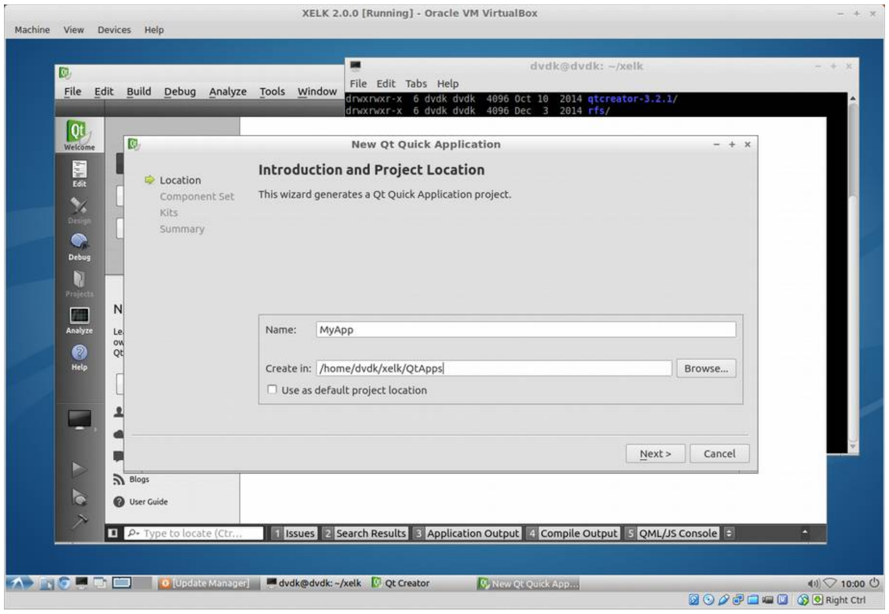
Follow the wizard to create a simple “Hello World!” application using the QtQuick 2 library:





Creates a Qt Quick application project that can contain both QML and C++ code.

Supported Platforms: Embedded Linux



```
drwxrwxr-x  6 dvdk dvdk  4096 Oct 10  2014 qtcreator-3.2.1/
drwxrwxr-x  6 dvdk dvdk  4096 Dec  3  2014 rfs/
```

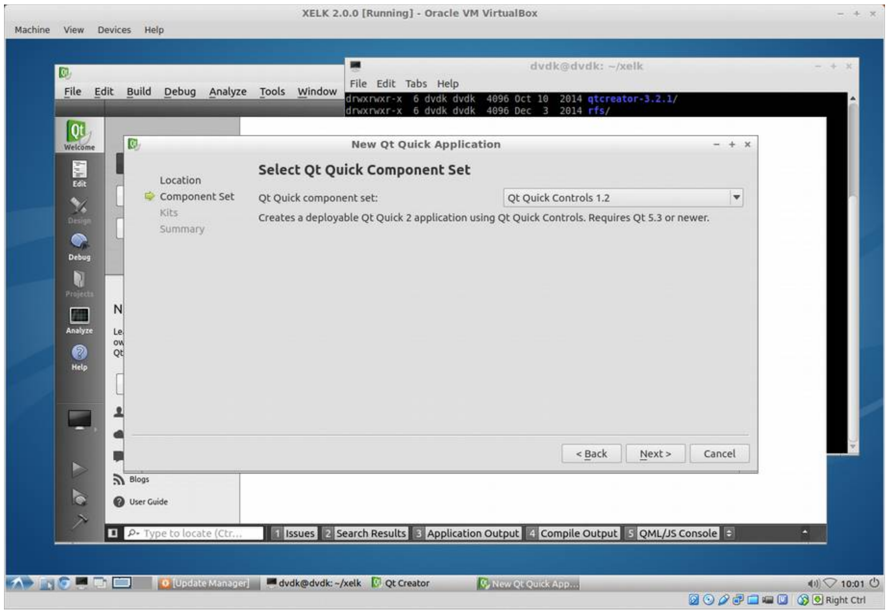
Introduction and Project Location

This wizard generates a Qt Quick Application project.

Name: MyApp

Create in: /home/dvdk/xelk/QtApps

☐ Use as default project location

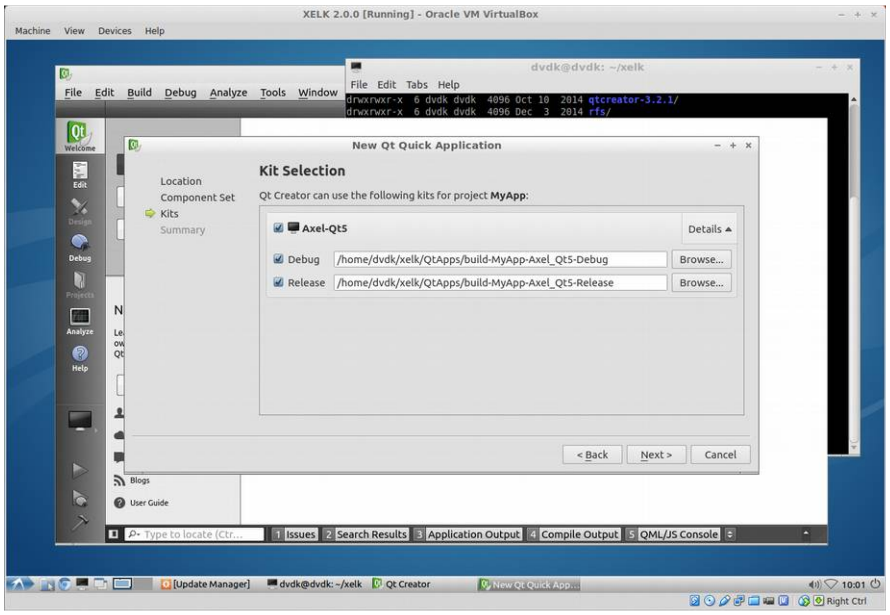


```
drwxrwxr-x  6 dvdk dvdk  4096 Oct 10  2014 qtcreator-3.2.1/
drwxrwxr-x  6 dvdk dvdk  4096 Dec  3  2014 rfs/
```

Select Qt Quick Component Set

Qt Quick component set: Qt Quick Controls 1.2

Creates a deployable Qt Quick 2 application using Qt Quick Controls. Requires Qt 5.3 or newer.

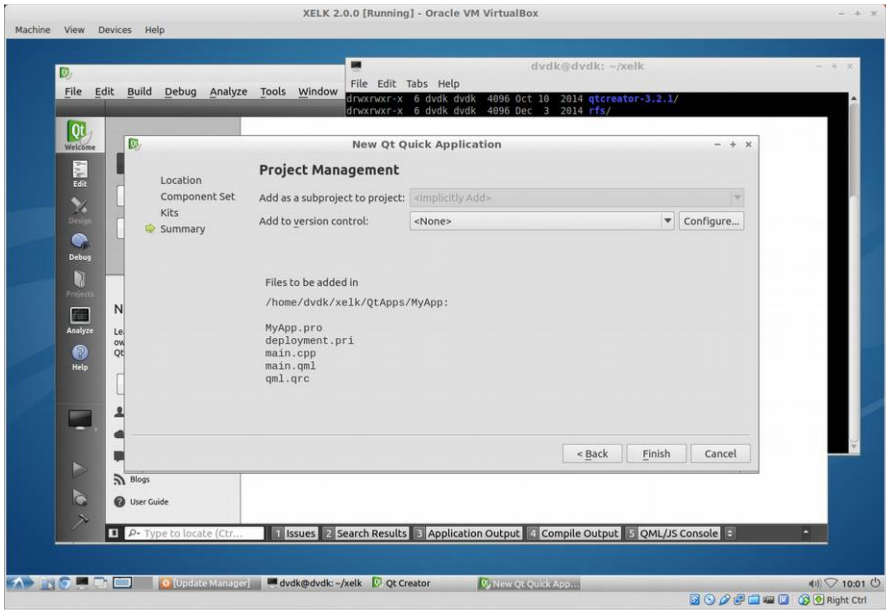


```
drwxrwxr-x  6 dvdk dvdk  4096 Oct 10  2014 qtcreator-3.2.1/
drwxrwxr-x  6 dvdk dvdk  4096 Dec  3  2014 rfs/
```

Kit Selection

Qt Creator can use the following kits for project **MyApp**:

- [✓] **Axel-Qt5**
- [✓] Debug /home/dvdk/xelk/QtApps/build-MyApp-Axel_Qt5-Debug
- [✓] Release /home/dvdk/xelk/QtApps/build-MyApp-Axel_Qt5-Release



```
drwxrwxr-x  6 dvdk dvdk  4096 Oct 10  2014 qtcreator-3.2.1/
drwxrwxr-x  6 dvdk dvdk  4096 Dec  3  2014 rfs/
```

Project Management

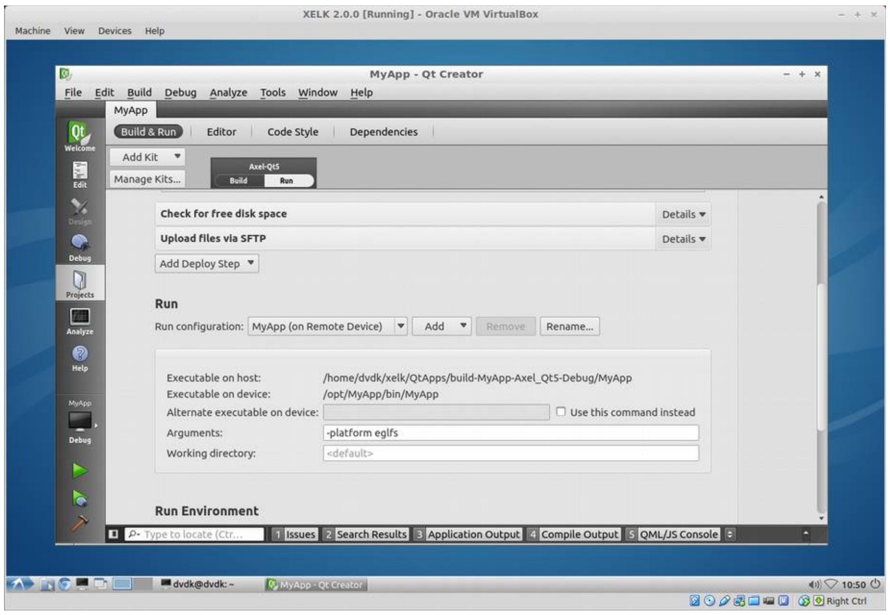
Add as a subproject to project: <Implicitly Add>

Add to version control: <None>

Files to be added in

```
/home/dvdk/xelk/QtApps/MyApp:

MyApp.pro
deployment.pri
main.cpp
main.qml
qml.qrc
```


XELK 2.0.0 [Running] - Oracle VM VirtualBox
MyApp - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Build & Run | Editor | Code Style | Dependencies
Add Kit
Manage Kits...
Axel-Qt5
Build Run
Add Deploy Step
Issues Search Results Application Output Compile Output QML/JS Console

Check for free disk space
Upload files via SFTP

Run

Run configuration: MyApp (on Remote Device)

```
Executable on host:              /home/dvdk/xelk/QtApps/build-MyApp-Axel_Qt5-Debug/MyApp
Executable on device:            /opt/MyApp/bin/MyApp
Alternate executable on device:                          ☐ Use this command instead
Arguments:                       -platform eglfs
Working directory:               <default>
```

Run Environment

5 Frequently Asked Questions

5.1 Q: Where can I found AXEL SOM information?

A: please refer to the following table:

Document	Location
AXEL main page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/Category:Axel
AXEL LITE main page on **DAVE Embedded Systems** Developers Wiki	http://wiki.dave.eu/index.php/Category:AxelLite
AXEL Hardware Manual	http://www.dave.eu/sites/default/files/files/axel-hm.pdf
AXEL LITE Hardware Manual	http://www.dave.eu/sites/default/files/files/axellite-hm.pdf
AXEL product page	http://www.dave.eu/products/som/freescale/imx6_axel-ultra
AXEL LITE product page	http://www.dave.eu/products/som/freescale/imx6_axel-lite

5.2 Q: I've received the XELK package. How am I supposed to start working with it?

A: You can follow the steps listed below:

1. Check the kit contents with the packing list included in the box
2. Insert the SD into the card slot on the carrier board
3. Connect the power supply adapter and the serial cable as described in Section 3.2
4. Start your terminal emulator program
5. Switch on the power supply

6. Monitor the boot process on the serial console
7. Install the DVDK virtual machine image (please refer to Section 3.5)
8. Check the XELK documentation (`doc` and `hw` directories of the microSD)
9. Check the virtual machine components (please refer to Section 3.5.1)
10. Check for updates of the public git repositories (please refer to Section 4.2.8)

5.3 Q: How can I update the XELK version?

A: please refer to Section 4.2.8 and the following page on the **DAVE Embedded Systems** Developer's Wiki: http://wiki.dave.eu/index.php/Software_Manual_%28Axel%29#XELK_Updates

5.4 Q: How can I update the git repositories provided with XELK 2.0.0 to XELK 2.2.0 version?

A: please update the git repositories, using the `git fetch/merge` or the `git pull` commands. For further details, please refer to section 4.2.8 and the **DAVE Embedded Systems** Developer's Wiki page: http://wiki.dave.eu/index.php/Software_Manual_%28Axel%29#XELK_Updates

5.5 Q: How can I work with the XYZ peripheral/interface?

A: please refer to the "i.MX 6Dual/6Quad Linux Reference Manual" provided with the L3.10.17_1.0.0 documentation package (https://www.freescale.com/webapp/Download?colCode=L3.10.17_1.0.0_IMX6QDLS_BUNDLE&appType=license&location=null&WT_TYPE=Board%20Support%20Packages&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=gz&WT_ASSET=Downloads&fileExt=.gz&Parent_nodeId=1337637154535695831062&Parent_pageType=product).

5.6 Q: How can I configure the AXEL system to boot from network?

A: booting from network is very helpful during the software development (both for kernel and applications). The kernel image is downloaded via TFTP while the root file system is remotely mounted via NFS from the host. It is assumed that the development host:

- is connected with the target host board through an Ethernet LAN
- exports the directory containing the root file system for the target through the NFS server
- runs a TFTP server
- has a proper subnet IP address

If your system does not match this configuration, just change the necessary variables and store them permanently with the u-boot `setenv/saveenv` commands. To do that, from the U-boot shell, please check the following parameters and set them accordingly with your host and target configuration:

Parameter	Description	Default
serverip	IP address of the host machine running the tftp/nfs server	192.168.0.13
ipaddress	IP address of the target	192.168.0.x
ethaddr	MAC address of the target	00:50:c2:1e:af:e0
netmask	Netmask of the target	255.255.255.0
gatewayip	IP address of the gateway	192.168.0.254
netdev	Ethernet device name	eth0
rootpath	Path to the NFS-exported directory	/opt/nfsroot/axel/xelk
bootfile	Path to the kernel binary image on the tftp server	axel/uImage
fdtfile	Path to the device tree binary image on the tftp server	axel/imx6q-xelk-h.dtb

Parameter	Description	Default
nfsargs	Kernel command line with parameters for loading the root file system through NFS	setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath} rootdelay=2

To run this configuration, just enter the command

```
run net_nfs
```

5.7 Q: Can you suggest some guidelines for the carrier board design?

A: As a starting point, you can refer to the Wiki page dedicated to the carrier board design guidelines (http://wiki.dave.eu/index.php/Carrier_board_design_guidelines_%28SOM%29), that will highlight some best practices that applies to all SOMs. For specific information on AXEL, please refer to the AXEL Integration Guide (http://wiki.dave.eu/index.php/Integration_guide_%28Axel%29)

5.8 Q: How can I change the CPU clock frequency?

A: The frequency of the CPU can be changed on the run using the Cpufreq framework (please refer to the documentation included into the Documentation/cpu-freq directory of the kernel source tree). The cpufreq framework works in conjunction with the related driver & governor.

Cpufreq implementation controls the Linux OPP (Operating Performance Points) adjusting the CPU core voltages and frequencies. CPUFreq is enabled by default in the AXEL kernel configuration.

To view the available governors:

```
root@axel-lite:~# cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_govern
ors

interactive conservative ondemand userspace powersave
performance
```

To view the supported OPP's (frequency in KHz):

```
root@axel-lite:~# cat
```

```
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_freque
ncies

396000 792000 996000
```

To change the OPP:

```
root@axel-lite:~# echo 396000 >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

Please note that OPP can be changed only using the userspace governor. If governors like ondemand is used, OPP change happens automatically based on the system load.

Please also note that the imx6q-cpufreq driver (http://lxr.free-electrons.com/source/drivers/cpufreq/imx6q-cpufreq.c?v=3.10) works on a per-SOC policy (and not on a per-core one), so the cpufreq governor changes the clock speed for all the ARM cores simultaneously.

5.9 Q: How can I limit the number of active CPU cores?

A: To evaluate the performances of the system with reduced number of CPU cores, the user can pass the maxcpus parameter to the kernel, by setting the command line arguments in u-boot.

For example, to set the number of active cores to 2, add the maxcpus parameter to the addmisc environment variable:

```
setenv addmisc 'setenv bootargs ${bootargs} maxcpus=2'
```

And add the addmisc variable to the boot macro:

```
setenv net_nfs 'run loadk loadfdt nfsargs addip addcons
addmisc; bootm ${buf} ${dtb_addr}'
```

For further details, please refer to the Documentation/kernel-parameters.txt file of the kernel source tree provided with the XELK.

5.10 Q: How can I modify the IP address of the board?

A: The network configuration is managed by a mix of init scripts and configuration files. In particular, the `/etc/network/interfaces` provides the interface settings used by the `ifup` and `ifdown` scripts, which are in turn activated by the `/etc/init.d/networking` init script. A basic strategy for IP address management is:

set a factory default

allow customers to change it by modifying the `/etc/network/interfaces` file.

An example is the following, that sets a static ip address and reads the MAC address from the kernel command line:

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface

auto lo

iface lo inet loopback

# Wired or wireless interfaces

auto eth0

iface eth0 inet static

        pre-up ifconfig eth0 hw ether $(cat /proc/cmdline | sed 's/.*eth=\([^ ]*\).*/\1/')

        address 192.168.1.1

        netmask 255.255.255.0
```

6 Appendices

6.1 U-Boot startup messages

The following messages will be printed on the serial console during U-Boot startup (please note that messages may vary for different U-Boot releases):

```
U-Boot 2013.04 (Jan 12 2016 - 15:51:23)-xelk-2.2.0

CPU:   Freescale i.MX6Q rev1.5 at 792 MHz
CPU:   Temperature 41 C, limits (-40 C, 125 C), calibration data: 0xc0
Reset cause: POR
Environment: SPI Flash
I2C:   ready
DRAM:  2 GiB
Now running in RAM - U-Boot at: 8ff28000
NAND:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
SF: Detected S25FL256S with page size 64 KiB, total 32 MiB
CB ConfigID CRC mismatch for 0x00000000 (was 0x00000000, expected 0x2144df1c) at block 3
(offset 96): using default
Display: LDB-AM-800480STMQW-TA1 (800x480)
SF: Detected S25FL256S with page size 64 KiB, total 32 MiB
In:    serial
Out:   serial
Err:   serial
SOM ConfigID#: 00000003
SOM UniqueID#: df646299:0b0579d4
CB ConfigID CRC mismatch for 0x00000000 (was 0x00000000, expected 0x2144df1c) at block 3
(offset 96): using default
CB ConfigID#: ffffffff
CB UniqueID#: 00000000:00000000
Board: MX6Q-AxelLiteCB ConfigID CRC mismatch for 0x00000000 (was 0x00000000, expected
0x2144df1c) at block 3 (offset 96): using default
 on XELK
CB ConfigID CRC mismatch for 0x00000000 (was 0x00000000, expected 0x2144df1c) at block 3
(offset 96): using default
Power: found PFUZE100 (devid=10, revid=21)
HW ver#: 0x1 (module_id)
CB ConfigID CRC mismatch for 0x00000000 (was 0x00000000, expected 0x2144df1c) at block 3
(offset 96): using default
unsupported boot devices
Net:   FEC
Normal Boot
Hit any key to stop autoboot:  0
U-Boot >
```

6.2 U-Boot default environment

The following messages will be printed on serial console entering the `print` command from the U-Boot shell (please note that messages may vary for different XELK releases):

```
addandroidargs=setenv bootargs ${bootargs} init=/init androidboot.console=ttymxc2
androidboot.hardware=freescale
addcons=setenv bootargs ${bootargs} console=ttymxc2,115200n8
adddisp0=setenv bootargs ${bootargs} video=mxcfb0:dev=lcd,${panel},if=RGB666
addhdmi=setenv bootargs ${bootargs} video=mxcfb0:dev=hdmi,1920x1080M@60,bpp=32
addip=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:$
{hostname}:${netdev}:off panic=1 fec_mac=${ethaddr}
```

```
addlvds0=setenv bootargs ${bootargs} video=mxcfb0:dev=ldb,${panel},if=RGB666 ldb=sin0
addlvds1=setenv bootargs ${bootargs} video=mxcfb0:dev=ldb,${panel},if=RGB666 ldb=sin1
addmisc=setenv bootargs ${bootargs} vmalloc=400M ${mtdparts}\\;${mtdparts_spi}
baudrate=115200
bootcmd=run usbrecovery; run mmcrecovery; run ${normalboot}
bootdelay=3
bootfile=axel/uImage
bootscript=echo Running bootscript from ${recoverydev} ...; source
configid_fixupfdt=if configid checkfdt ${fdtaddr} som_configid ${som_configid#}; then if
configid checkfdt ${fdtaddr} cb_configid ${cb_configid#}; then configid fdt_uniqueid $
{fdtaddr}; fi; fi
ethaddr=00:50:c2:1e:af:e0
ethprime=FEC0
fdt_high=FFFFFFFF
fdtaddr=0x18000000
fdtfile=axel/imx6q-xelk-h.dtb
hostname=xelk
initrd_high=0xffffffff
ipaddr=192.168.0.89
load=tftp ${loadaddr} ${uboot}
loadaddr=0x12000000
loadbootscript=fatload ${recoverydev} 0:1 ${loadaddr} ${script};
loadfdt=tftpboot ${fdtaddr} ${serverip}:${fdtfile}
loadk=tftpboot ${loadaddr} ${serverip}:${bootfile}
loadsplash=run spi_loadsplash
loadsplashfile=tftpboot ${loadaddr} axel/${splashfile}
mmc_loadfdt=fatload mmc 0:1 ${fdtaddr} imx6q-xelk-h.dtb
mmc_loadk=fatload mmc 0:1 ${loadaddr} uImage
mmc_loadsplash=fatload mmc 0:1 ${loadaddr} ${splashfile}; cp.b ${loadaddr} ${splashimage} $
{filesize}
mmcargs=setenv bootargs root=${mmcroot}
mmcboot=run mmcargs addcons addmisc; if run mmc_loadk; then if run mmc_loadfdt; then if run
configid_fixupfdt; then bootm ${loadaddr} - ${fdtaddr}; fi ; fi; fi
mmcrecovery=mmc dev 0; mmc rescan; setenv recoverydev mmc; run recovery
mmcroot=/dev/mmcblk0p2 rootwait rw
mtdids=nand0=gpmi-nand
mtdparts=mtdparts=gpmi-nand:8M(nand-uboot),1M(nand-env1),1M(nand-env2),1M(nand-fdt),1M(nand-
spare),8M(nand-kernel),4M(nand-splash),-(nand-ubi)
mtdparts_spi=spi32766.0:1M(spi-uboot),256k(spi-env1),256k(spi-env2),512k(spi-dtb),6M(spi-ker
nel),4M(spi-splash),-(spi-free)
nand_andr_nand=run nand_loadk nand_loadfdt nandargs addcons addmisc addandroidargs; if run
configid_fixupfdt; then bootm ${loadaddr} - ${fdtaddr}; fi
nand_loadfdt=nand read ${fdtaddr} nand-fdt
nand_loadk=nand read ${loadaddr} nand-kernel
nand_loadsplash=nand read ${splashimage} nand-splash
nand_nand=run nand_loadk nand_loadfdt nandargs addcons addmisc; if run configid_fixupfdt;
then bootm ${loadaddr} - ${fdtaddr}; fi
nand_update=echo update  for NAND boot using kobs-ng from Linux
nand_updatefdt=nand erase.part nand-fdt; nand write ${fdtaddr} nand-fdt ${filesize}
nand_updatek=nand erase.part nand-kernel; nand write ${loadaddr} nand-kernel ${filesize}
nand_updatesplash=nand erase.part nand-splash; nand write ${loadaddr} nand-splash $
{filesize}
nandargs=setenv bootargs ubi.mtd=7 root=ubi0_0 rootfstype=ubifs rw
net_andr_nfs=run loadk loadfdt nfsargs addip addcons addmisc addandroidargs; if run
configid_fixupfdt; then bootm ${loadaddr} - ${fdtaddr}; fi
net_nfs=run loadk loadfdt nfsargs addip addcons addmisc; if run configid_fixupfdt; then
bootm ${loadaddr} - ${fdtaddr}; fi
netdev=eth0
netmask=255.255.255.0
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath},v3,tcp
normalboot=net_nfs
panel=LDB-AM-800480STMQW-TA1
recovery=if run loadbootscript; then run bootscript; fi
rootpath=/opt/nfsroot/axel/xelk
script=boot.scr
serverip=192.168.0.13
spi_andr_nand=sf probe; run spi_loadk spi_loadfdt nandargs addcons addmisc addandroidargs;
if run configid_fixupfdt; then bootm ${loadaddr} - ${fdtaddr}; fi
spi_loadfdt=sf read ${fdtaddr} 180000 80000
spi_loadk=sf read ${loadaddr} 200000 600000
spi_loadsplash=sf probe; sf read ${splashimage} 0x800000 ${splashsize}
spi_nand=sf probe; run spi_loadk spi_loadfdt nandargs addcons addmisc; if run
configid_fixupfdt; then bootm ${loadaddr} - ${fdtaddr}; fi
```

```
spi_update=sf probe; sf erase 0 +${filesize};sf write ${loadaddr} 400 ${filesize}
spi_updatefdt=sf erase 180000 80000; sf write ${fdtaddr} 180000 ${filesize}
spi_updatek=sf erase 200000 600000; sf write ${loadaddr} 200000 ${filesize}
spi_updatesplash=sf probe; sf erase 0x800000 +${filesize}; sf write ${loadaddr} 0x800000 $
{filesize}
splashfile=splash_image.bmp
splashimage=0x20000002
splashpos=m,m
splashsize=0x400000
uboot=axel/u-boot.imx
usbrecovery=usb start; usb dev 0; setenv recoverydev usb; run recovery
```

6.3 Boot messages on the serial console

The following messages will be printed on serial console during the Linux boot process (please note that messages may vary for different Linux releases):

```
U-Boot > run net_nfs
Using FEC device
TFTP from server 192.168.0.13; our IP address is 192.168.0.123
Filename 'axel/xelk-2.2.0/xelk-2.2.0_uImage'.
Load address: 0x12000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################
         1.9 MiB/s
done
Bytes transferred = 5741528 (579bd8 hex)
Using FEC device
TFTP from server 192.168.0.13; our IP address is 192.168.0.123
Filename 'axel/xelk-2.2.0/xelk-2.2.0_imx6q-xelk-l.dtb'.
Load address: 0x18000000
Loading: ##########
         1.3 MiB/s
done
Bytes transferred = 46202 (b47a hex)
## Booting kernel from Legacy Image at 12000000 ...
   Image Name:   Linux-3.10.17-xelk-2.2.0
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    5741464 Bytes = 5.5 MiB
   Load Address: 10008000
   Entry Point:  10008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 18000000
   Booting using the fdt blob at 0x18000000
   Loading Kernel Image ... OK
OK
Power: using LDO bypass mode!
Frame buffer: configure splashscreen reserved memory to 0x8f600000 (1 MiB)
   Using Device Tree in place at 18000000, end 1800e479

Starting kernel ...
```

```
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 3.10.17-xelk-2.2.0 (jenkins@linuxserver2) (gcc version 4.8.1
(GCC) ) #1 SMP PREEMPT Tue Jan 12 15:57:48 CET 2016
[    0.000000] Machine: Freescale i.MX6 Quad/DualLite (Device Tree), model: AxelLite Quad on
XELK-L
[    0.000000] Reserved memory: created ipuv3_fb memory pool at 0x8f600000, size 1 MiB
[    0.000000] Reserved memory: initialized node splashscreen, compatible id fsl,ipuv3-fb
[    0.000000] cma: CMA: reserved 320 MiB at 62000000
[    0.000000] PERCPU: Embedded 8 pages/cpu @81df7000 s8896 r8192 d15680 u32768
[    0.000000] Kernel command line: root=/dev/nfs rw
nfsroot=192.168.0.13:/opt/nfsroot/axel/xelk-2.1.0-qt5,v3,tcp
ip=192.168.0.123:192.168.0.13::255.255.255.0:xelk:eth0:off panic=1
fec_mac=00:50:c2:1e:af:e3)
[    0.000000] PID hash table entries: 4096 (order: 2, 16384 bytes)
[    0.000000] Dentry cache hash table entries: 262144 (order: 8, 1048576 bytes)
[    0.000000] Inode-cache hash table entries: 131072 (order: 7, 524288 bytes)
[    0.000000] Memory: 2038MB 8MB = 2046MB total
[    0.000000] Memory: 1734884k/1734884k available, 362268k reserved, 424484K highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xe6800000 - 0xff000000   ( 392 MB)
[    0.000000]     lowmem  : 0x80000000 - 0xe6000000   (1632 MB)
[    0.000000]     pkmap   : 0x7fe00000 - 0x80000000   (   2 MB)
[    0.000000]     modules : 0x7f000000 - 0x7fe00000   (  14 MB)
[    0.000000]       .text : 0x80008000 - 0x80ccf544   (13086 kB)
[    0.000000]       .init : 0x80cd0000 - 0x80d102c0   ( 257 kB)
[    0.000000]       .data : 0x80d12000 - 0x80d74020   ( 393 kB)
[    0.000000]        .bss : 0x80d74020 - 0x80ddbbf8   ( 415 kB)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] L310 cache controller enabled
[    0.000000] l2x0: 16 ways, CACHE_ID 0x410000c7, AUX_CTRL 0x32070000, Cache size: 1048576
B
[    0.000000] sched_clock: 32 bits at 3000kHz, resolution 333ns, wraps every 1431655ms
[    0.000000] CPU identified as i.MX6Q, silicon rev 1.5
[    0.000000] Console: colour dummy device 80x30
[    0.000719] Calibrating delay loop... 1581.05 BogoMIPS (lpj=7905280)
[    0.090122] pid_max: default: 32768 minimum: 301
[    0.090314] Mount-cache hash table entries: 512
[    0.098643] CPU: Testing write buffer coherency: ok
[    0.098911] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.099015] Setting up static identity map for 0x806bfa90 - 0x806bfae8
[    0.228960] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.338960] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.398959] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.399058] Brought up 4 CPUs
[    0.399087] SMP: Total of 4 processors activated (6324.22 BogoMIPS).
[    0.399096] CPU: All CPU(s) started in SVC mode.
[    0.399782] devtmpfs: initialized
[    0.404009] pinctrl core: initialized pinctrl subsystem
[    0.404307] regulator-dummy: no parameters
[    0.427959] NET: Registered protocol family 16
[    0.438095] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.438837] Use WDOG2 as reset source
[    0.447072] syscon 20c8000.anatop: regmap [mem 0x020c8000-0x020c8fff] registered
[    0.447318] vdd1p1: 800 <--> 1375 mV at 1125 mV
[    0.447626] vdd3p0: 2800 <--> 3150 mV at 3000 mV
[    0.447905] vdd2p5: 2000 <--> 2750 mV at 2425 mV
[    0.448180] cpu: 725 <--> 1450 mV
[    0.448452] vddpu: 725 <--> 1450 mV
[    0.448731] vddsoc: 725 <--> 1450 mV
[    0.450489] syscon 20e0000.iomuxc-gpr: regmap [mem 0x020e0000-0x020e0037] registered
[    0.452762] syscon 21bc000.ocotp-ctrl: regmap [mem 0x021bc000-0x021bffff] registered
[    0.455992] hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
[    0.456004] hw-breakpoint: maximum watchpoint size is 4 bytes.
[    0.457037] imx6q-pinctrl 20e0000.iomuxc: initialized IMX pinctrl driver
[    0.465931] bio: create slab <bio-0> at 0
[    0.467768] mxs-dma 110000.dma-apbh: initialized
[    0.468398] usb_otg_vbus: 5000 mV
[    0.468632] usb_h1_vbus: 5000 mV
[    0.468809] 3P3V: 3300 mV
[    0.469000] 1P8V: 1800 mV
[    0.469269] vgaarb: loaded
[    0.469916] SCSI subsystem initialized
[    0.470297] usbcore: registered new interface driver usbfs
[    0.470352] usbcore: registered new interface driver hub
[    0.470459] usbcore: registered new device driver usb
[    0.472274] i2c i2c-0: IMX I2C adapter registered
[    0.472493] imx-i2c 21a8000.i2c: cannot look up pinctrl state recovery: -19 (bus recovery
disabled)
[    0.473099] i2c i2c-1: IMX I2C adapter registered
[    0.473192] Linux video capture interface: v2.00
[    0.473237] pps_core: LinuxPPS API ver. 1 registered
[    0.473248] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti
<giometti@linux.it>
[    0.473272] PTP clock support registered
[    0.489025] imx-ipuv3 2400000.ipu: IPU DMFC NORMAL mode: 1(0~1), 5B(4,5), 5F(6,7)
[    0.509018] imx-ipuv3 2800000.ipu: IPU DMFC NORMAL mode: 1(0~1), 5B(4,5), 5F(6,7)
[    0.509846] imx6q-pinctrl 20e0000.iomuxc: pin MX6Q_PAD_GPIO_19 already requested by
20e0000.iomuxc; cannot claim for 21dc000.mipi_csi
[    0.509863] imx6q-pinctrl 20e0000.iomuxc: pin-149 (21dc000.mipi_csi) status -22
[    0.509876] imx6q-pinctrl 20e0000.iomuxc: could not request pin 149 on device
20e0000.iomuxc
[    0.509889] mxc_mipi_csi2 21dc000.mipi_csi: Error applying setting, reverse things back
[    0.509956] mxc_mipi_csi2 21dc000.mipi_csi: i.MX MIPI CSI2 driver probed
[    0.509968] mxc_mipi_csi2 21dc000.mipi_csi: i.MX MIPI CSI2 dphy version is 0x3130302a
[    0.510033] MIPI CSI2 driver module loaded
[    0.510161] Advanced Linux Sound Architecture Driver Initialized.
[    0.510885] Bluetooth: Core ver 2.16
[    0.510929] NET: Registered protocol family 31
[    0.510937] Bluetooth: HCI device and connection manager initialized
[    0.510956] Bluetooth: HCI socket layer initialized
[    0.510969] Bluetooth: L2CAP socket layer initialized
[    0.511000] Bluetooth: SCO socket layer initialized
[    0.511284] cfg80211: Calling CRDA to update world regulatory domain
[    0.512210] Switching to clocksource mxc_timer1
[    0.753556] imx6q-pcie 1ffc000.pcie: phy link never came up
[    0.753755] PCI host bridge to bus 0000:00
[    0.753776] pci_bus 0000:00: root bus resource [io  0x1000-0x10000]
[    0.753788] pci_bus 0000:00: root bus resource [mem 0x01000000-0x01efffff]
[    0.753804] pci_bus 0000:00: No busn resource found for root bus, will use [bus 00-ff]
[    0.755206] PCI: bus0: Fast back to back transfers disabled
[    0.756055] PCI: bus1: Fast back to back transfers enabled
[    0.756266] pci 0000:00:00.0: BAR 0: assigned [mem 0x01000000-0x010fffff]
[    0.756313] pci 0000:00:00.0: BAR 6: assigned [mem 0x01100000-0x0110ffff pref]
[    0.756327] pci 0000:00:00.0: PCI bridge to [bus 01]
[    0.765607] NET: Registered protocol family 2
[    0.766225] TCP established hash table entries: 16384 (order: 5, 131072 bytes)
[    0.766597] TCP bind hash table entries: 16384 (order: 5, 131072 bytes)
[    0.766958] TCP: Hash tables configured (established 16384 bind 16384)
[    0.767178] TCP: reno registered
[    0.767198] UDP hash table entries: 1024 (order: 3, 32768 bytes)
[    0.767303] UDP-Lite hash table entries: 1024 (order: 3, 32768 bytes)
[    0.767691] NET: Registered protocol family 1
[    0.768039] RPC: Registered named UNIX socket transport module.
[    0.768052] RPC: Registered udp transport module.
[    0.768059] RPC: Registered tcp transport module.
[    0.768066] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.768603] hw perfevents: enabled with ARMv7 Cortex-A9 PMU driver, 7 counters available
[    0.769502] pureg-dummy: no parameters
[    0.770453] imx6_busfreq busfreq.15: DDR medium rate not supported.
[    0.770899] Bus freq driver module loaded
[    0.777544] VFS: Disk quotas dquot_6.5.2
[    0.780105] NFS: Registering the id_resolver key type
[    0.780157] Key type id_resolver registered
[    0.780166] Key type id_legacy registered
[    0.780206] NTFS driver 2.1.30 [Flags: R/W].
[    0.780614] fuse init (API version 7.22)
[    0.781002] msgmni has been set to 3199
[    0.782370] io scheduler noop registered
[    0.782381] io scheduler deadline registered
[    0.782414] io scheduler cfq registered (default)
[    0.782678] imx-weim 21b8000.weim: WEIM driver registered.
[    0.784346] MIPI DSI driver module loaded
[    0.785224] mxc_sdc_fb fb.23: register mxc display driver ldb
[    0.785247] mxc_ldb 20e0000.ldb: change IPU DI1 to IPU DI0 for LDB channel0.
[    0.785460] mxc_sdc_fb fb.23: using reserved memory region at 0x8f600000, size 1 MiB
[    0.785474] mxc_sdc_fb fb.23: assigned reserved memory node splashscreen
[    0.785486] mxc_sdc_fb fb.23: using memory region 0x8f600000 0x8f776fff
[    0.827938] imx-sdma 20ec000.sdma: no iram assigned, using external mem
[    0.828742] imx-sdma 20ec000.sdma: loaded firmware 1.1
[    0.831117] imx-sdma 20ec000.sdma: initialized
[    0.832576] pfuze100-regulator 0-0008: Full lay: 2, Metal lay: 1
[    0.833178] pfuze100-regulator 0-0008: FAB: 0, FIN: 0
[    0.833191] pfuze100-regulator 0-0008: pfuze100 found.
[    0.834616] SW1AB: 300 <--> 1875 mV at 1150 mV
[    0.836536] SW1C: 300 <--> 1875 mV at 1175 mV
[    0.838453] SW2: 800 <--> 3300 mV at 3300 mV
[    0.840367] SW3A: 400 <--> 1975 mV at 1500 mV
[    0.842278] SW3B: 400 <--> 1975 mV at 1500 mV
[    0.845204] SW4: ramp_delay not set
[    0.845218] SW4: 1800 mV
[    0.846563] SWBST: 5000 <--> 5150 mV at 5000 mV
[    0.847888] VSNVS: 1200 <--> 3000 mV at 3000 mV
[    0.848621] VREFDDR: 750 mV
[    0.849345] VGEN1: 800 <--> 1550 mV at 1500 mV
[    0.850668] VGEN2: 800 <--> 1550 mV at 1500 mV
[    0.853732] VGEN3: 2500 mV
[    0.856829] VGEN4: 1800 mV
[    0.859892] VGEN5: 2800 mV
[    0.862378] VGEN6: 3300 mV
[    0.863225] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.864316] Serial: IMX driver
[    0.864672] 21ec000.serial: ttymxc2 at MMIO 0x21ec000 (irq = 60) is a IMX
[    1.806014] console [ttymxc2] enabled
[    1.810006] serial: Freescale lpuart driver
[    1.815543] [drm] Initialized drm 1.1.0 20060810
[    1.820607] [drm] Initialized vivante 1.0.0 20120216 on minor 0
[    1.833301] brd: module loaded
[    1.839845] loop: module loaded
[    1.846775] nand: device found, Manufacturer ID: 0x01, Chip ID: 0xdc
[    1.853177] nand: AMD/Spansion S34ML04G1
[    1.857129] nand: 512MiB, SLC, page size: 2048, OOB size: 64
[    1.862996] gpmi-nand 112000.gpmi-nand: mode:4 ,failed in set feature.
[    1.869566] Scanning device for bad blocks
[    2.312288] Bad eraseblock 4010 at 0x00001f540000
[    2.326396] 8 cmdlinepart partitions found on MTD device gpmi-nand
[    2.332624] Creating 8 MTD partitions on "gpmi-nand":
[    2.337710] 0x000000000000-0x000000800000 : "nand-uboot"
[    2.343986] 0x000000800000-0x000000900000 : "nand-env1"
[    2.350066] 0x000000900000-0x000000a00000 : "nand-env2"
[    2.356187] 0x000000a00000-0x000000b00000 : "nand-fdt"
[    2.362162] 0x000000b00000-0x000000c00000 : "nand-spare"
[    2.368298] 0x000000c00000-0x000001400000 : "nand-kernel"
[    2.374554] 0x000001400000-0x000001800000 : "nand-splash"
[    2.380826] 0x000001800000-0x000020000000 : "nand-ubi"
[    2.387169] gpmi-nand 112000.gpmi-nand: driver registered.
[    2.394047] m25p80 spi32766.0: s25fl256s1 (32768 Kbytes)
[    2.399375] 7 cmdlinepart partitions found on MTD device spi32766.0
[    2.405667] Creating 7 MTD partitions on "spi32766.0":
[    2.410818] 0x000000000000-0x000000100000 : "spi-uboot"
[    2.416994] 0x000000100000-0x000000140000 : "spi-env1"
[    2.423009] 0x000000140000-0x000000180000 : "spi-env2"
[    2.429016] 0x000000180000-0x000000200000 : "spi-dtb"
[    2.434946] 0x000000200000-0x000000800000 : "spi-kernel"
[    2.441071] 0x000000800000-0x000000c00000 : "spi-splash"
[    2.447243] 0x000000c00000-0x000002000000 : "spi-free"
[    2.453265] spi_imx 2008000.ecspi: probed
[    2.458038] CAN device driver interface
[    2.462688] flexcan 2090000.can: device registered (reg_base=e69e0000, irq=142)
[    2.475230] libphy: fec_enet_mii_bus: probed
[    2.479934] fec 2188000.ethernet eth0: registered PHC device 0
[    2.486529] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    2.493108] ehci-pci: EHCI PCI platform driver
[    2.497978] usbcore: registered new interface driver usb-storage
[    2.512236] ci_hdrc ci_hdrc.1: doesn't support gadget
[    2.517350] ci_hdrc ci_hdrc.1: EHCI Host Controller
[    2.522299] ci_hdrc ci_hdrc.1: new USB bus registered, assigned bus number 1
[    2.542285] ci_hdrc ci_hdrc.1: USB 2.0 started, EHCI 1.00
[    2.547792] usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
[    2.554629] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    2.561881] usb usb1: Product: EHCI Host Controller
[    2.566802] usb usb1: Manufacturer: Linux 3.10.17-xelk-2.2.0 ehci_hcd
[    2.573283] usb usb1: SerialNumber: ci_hdrc.1
[    2.578239] hub 1-0:1.0: USB hub found
[    2.582037] hub 1-0:1.0: 1 port detected
[    2.586789] mousedev: PS/2 mouse device common for all mice
[    2.593074] i2c-core: driver [egalax_i2c] using legacy suspend method
[    2.599546] i2c-core: driver [egalax_i2c] using legacy resume method
[    2.606609] input: TSC2007 Touchscreen as
/devices/soc0/soc.1/2100000.aips-bus/21a8000.i2c/i2c-1/1-0048/input/input0
[    2.618038] snvs_rtc 20cc034.snvs-rtc-lp: rtc core: registered 20cc034.snvs-rtc-lp as
rtc0
[    2.626453] i2c /dev entries driver
[    2.632415] VGEN4: operation not allowed
[    2.639407] VGEN3: operation not allowed
[    2.742480] ov5640_read_reg:write reg error:reg=300a
[    2.747475] camera ov5640_mipi is not found
[    2.752224] mxc_v4l2_output v4l2_out.26: V4L2 device registered as video16
[    2.759295] mxc_v4l2_output v4l2_out.26: V4L2 device registered as video17
[    2.767710] ina2xx 1-0044: power monitor ina226 (Rshunt = 10000 uOhm)
[    2.775235] imx2-wdt 20bc000.wdog: IMX2+ Watchdog Timer enabled. timeout=60s (nowayout=0)
[    2.783591] Bluetooth: HCI UART driver ver 2.2
[    2.788065] Bluetooth: HCI H4 protocol initialized
[    2.792907] Bluetooth: HCI BCSP protocol initialized
[    2.797900] Bluetooth: HCILL protocol initialized
[    2.802750] cpuidle: using governor ladder
[    2.806876] cpuidle: using governor menu
[    2.810854] sdhci: Secure Digital Host Controller Interface driver
[    2.817079] sdhci: Copyright(c) Pierre Ossman
[    2.821463] sdhci-pltfm: SDHCI platform and OF driver helper
[    2.827890] mmc0: no vqmmc regulator found
[    2.832023] mmc0: no vmmc regulator found
[    2.872281] mmc0: SDHCI controller on 2190000.usdhc [2190000.usdhc] using ADMA
[    2.881944] mmc1: no vqmmc regulator found
[    2.890142] mmc1: no vmmc regulator found
[    2.944297] mmc1: SDHCI controller on 2194000.usdhc [2194000.usdhc] using ADMA
[    3.603028] Galcore version 4.6.9.9754
[    3.636156] mxc_vdoa 21e4000.vdoa: i.MX Video Data Order Adapter(VDOA) driver probed
[    3.644724] mxc_asrc 2034000.asrc: mxc_asrc registered
[    3.650584] mxc_vpu 2040000.vpu: VPU initialized
[    3.786682] caam 2100000.caam: device ID = 0x0a16010000000000 (Era -524)
[    3.793436] caam 2100000.caam: job rings = 2, qi = 0
[    3.798928] caam 2100000.caam: authenc-hmac-md5-cbc-aes-caam
[    3.804749] caam 2100000.caam: authencesn-hmac-md5-cbc-aes-caam
[    3.810803] caam 2100000.caam: authenc-hmac-sha1-cbc-aes-caam
[    3.816701] caam 2100000.caam: authencesn-hmac-sha1-cbc-aes-caam
[    3.822854] caam 2100000.caam: authenc-hmac-sha224-cbc-aes-caam
[    3.828903] caam 2100000.caam: authencesn-hmac-sha224-cbc-aes-caam
[    3.835228] caam 2100000.caam: authenc-hmac-sha256-cbc-aes-caam
[    3.841289] caam 2100000.caam: authencesn-hmac-sha256-cbc-aes-caam
[    3.847619] caam 2100000.caam: authenc-hmac-md5-cbc-des3_ede-caam
[    3.853856] caam 2100000.caam: authencesn-hmac-md5-cbc-des3_ede-caam
[    3.860342] caam 2100000.caam: authenc-hmac-sha1-cbc-des3_ede-caam
[    3.866668] caam 2100000.caam: authencesn-hmac-sha1-cbc-des3_ede-caam
[    3.873255] caam 2100000.caam: authenc-hmac-sha224-cbc-des3_ede-caam
[    3.879748] caam 2100000.caam: authencesn-hmac-sha224-cbc-des3_ede-caam
[    3.886499] caam 2100000.caam: authenc-hmac-sha256-cbc-des3_ede-caam
[    3.893004] caam 2100000.caam: authencesn-hmac-sha256-cbc-des3_ede-caam
[    3.899750] caam 2100000.caam: authenc-hmac-md5-cbc-des-caam
[    3.905548] caam 2100000.caam: authencesn-hmac-md5-cbc-des-caam
[    3.911601] caam 2100000.caam: authenc-hmac-sha1-cbc-des-caam
[    3.917497] caam 2100000.caam: authencesn-hmac-sha1-cbc-des-caam
[    3.923648] caam 2100000.caam: authenc-hmac-sha224-cbc-des-caam
[    3.929702] caam 2100000.caam: authencesn-hmac-sha224-cbc-des-caam
[    3.936033] caam 2100000.caam: authenc-hmac-sha256-cbc-des-caam
[    3.942088] caam 2100000.caam: authencesn-hmac-sha256-cbc-des-caam
[    3.948414] caam 2100000.caam: ecb-des-caam
[    3.952744] caam 2100000.caam: ecb-arc4-caam
[    3.957142] caam 2100000.caam: ecb-aes-caam
[    3.961439] caam 2100000.caam: ctr-aes-caam
[    3.965756] caam 2100000.caam: cbc-aes-caam
[    3.970074] caam 2100000.caam: ecb-des3-caam
[    3.974489] caam 2100000.caam: cbc-3des-caam
[    3.978872] caam 2100000.caam: cbc-des-caam
[    3.983104] caam 2100000.caam: fsl,sec-v4.0 algorithms registered in /proc/crypto
[    3.994234] platform 2101000.jr0: registering rng-caam
[    4.000556] platform caam_sm: caam_sm_test: 8-byte key test match OK
[    4.007091] platform caam_sm: caam_sm_test: 16-byte key test match OK
[    4.013707] platform caam_sm: caam_sm_test: 32-byte key test match OK
[    4.020624] platform caam_secvio.28: security violation service handlers armed
[    4.028081] usbcore: registered new interface driver usbhid
[    4.033697] usbhid: USB HID core driver
[    4.040466] TCP: cubic registered
[    4.044318] NET: Registered protocol family 10
[    4.049740] sit: IPv6 over IPv4 tunneling driver
[    4.055026] NET: Registered protocol family 17
[    4.059520] can: controller area network core (rev 20120528 abi 9)
[    4.065827] NET: Registered protocol family 29
[    4.070341] can: raw protocol (rev 20120528)
[    4.074657] can: broadcast manager protocol (rev 20120528 t)
[    4.080354] can: netlink gateway (rev 20130117) max_hops=1
[    4.086188] Bluetooth: RFCOMM TTY layer initialized
[    4.091119] Bluetooth: RFCOMM socket layer initialized
[    4.096304] Bluetooth: RFCOMM ver 1.11
[    4.100083] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    4.105435] Bluetooth: BNEP filters: protocol multicast
[    4.110696] Bluetooth: BNEP socket layer initialized
[    4.115702] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    4.121658] Bluetooth: HIDP socket layer initialized
[    4.126706] 8021q: 802.1Q VLAN Support v1.8
[    4.131219] Key type dns_resolver registered
[    4.135712] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
[    4.146875] VGEN6: disabling
[    4.151391] VGEN1: disabling
[    4.156288] input: gpio-keys.22 as /devices/soc0/gpio-keys.22/input/input1
[    4.165577] snvs_rtc 20cc034.snvs-rtc-lp: setting system clock to 1970-01-01 00:00:01 UTC
(1)
[    4.185004] fec 2188000.ethernet eth0: Freescale FEC PHY driver [Micrel KSZ9031 Gigabit
PHY] (mii_bus:phy_addr=2188000.ethernet:07, irq=-1)
[    4.197650] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    7.182589] libphy: 2188000.ethernet:07 - Link is Up - 100/Full
[    7.202338] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    7.222798] IP-Config: Complete:
[    7.226089]      device=eth0, hwaddr=00:50:c2:1e:af:e3, ipaddr=192.168.0.123,
mask=255.255.255.0, gw=255.255.255.255
[    7.236688]      host=xelk, domain=, nis-domain=(none)
[    7.241873]      bootserver=192.168.0.13, rootserver=192.168.0.13, rootpath=
[    7.249049] ALSA device list:
[    7.252057]   No soundcards found.
[    7.270816] VFS: Mounted root (nfs filesystem) on device 0:11.
[    7.279473] devtmpfs: mounted
[    7.283499] Freeing unused kernel memory: 256K (80cd0000 - 80d10000)
INIT: version 2.88 booting
Starting udev
[    9.359826] udevd[150]: starting version 182
[   11.258760] ERROR: v4l2 capture: slave not found!
[   11.277313] ERROR: v4l2 capture: slave not found!
Starting Bootlog daemon: bootlogd: cannot allocate pseudo tty: No such file or directory
bootlogd.
Populating dev cache
ALSA: Restoring mixer settings...
Configuring network interfaces... /usr/sbin/alsactl: load_state:1729: No soundcards found...
ifup skipped for nfsroot interface eth0
run-parts: /etc/network/if-pre-up.d/nfsroot exited with return code 1
Starting rpcbind daemon...done.
Sat Jun 13 11:18:00 UTC 2015
INIT: Entering runlevel: 5
Starting system message bus: dbus.
Starting OpenBSD Secure Shell server: sshd
done.
Starting advanced power management daemon: No APM support in kernel
(failed.)
```

```
Starting syslogd/klogd: done
 * Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
   ...done.
Starting Telephony daemon
Starting Linux NFC daemon
Stopping Bootlog daemon: bootlogd.
Using calibration data stored in /etc/pointercal

Poky (Yocto Project Reference Distro) 1.5.1 xelk-l /dev/ttymxc2

xelk-l login:
```