Linux

For

BCT RM3

Application Notes

Document Reference: BCTRM3 Linux Guide

Document Issue: 1.01

Associated SDK release: 4.0.0.1

Contents

Introduction

The content of this document provides information required to start building Linux operating systems for the BCT RM3 platform. It covers:

- The tools and components required for building a Linux operating system
- How to install the build components
- How to compile U-Boot boot loader
- How to compile the Linux Kernel
- How to generate an Ubuntu root file system
- How to boot embedded Linux on the RM3 platform using U-boot

Environment Setup

The components required for building Linux for the RM3 platform are:

- A cross-compiling tool chain
- U-Boot source code
- Linux kernel source code
- A Linux root file system

A pre-built tool chain is provided (Linaro 2012.06). This runs on an x86 Linux PC and is configured to compile ARM code compatible with RM3.
The Linux kernel is based on the Freescale 3.0.35 (4.0.0) release and is at version **3.0.35-2310-gc27cb38-gbaf8947.** The source code is made available as a git repository.

The root file system is based on Ubuntu 11.10 Oneiric release 4.0.0 from Freescale and is compiled for ARMEL architecture. This is necessary because Freescale libraries for the VPU (video processor) and GPU (graphics processor) are only available in ARMEL, and cannot be mixed with other architectures. This is different from Ubuntu 12.04 LTS Precise, which is compiled for ARMHF architecture.

One user account has been created:
username “linaro”
password “linaro”

The components above have all been tested using an x86 development machine running Ubuntu 12.10.

Installation of the Embedded Linux build components

Create "Embedded" directory

All instructions here assume that there is a directory called "/embedded" on the development PC. So first of all, as root or as a user with root privileges, create a directory called “embedded” in the root of the file system and enter the directory. Issue the following commands to achieve this:

```
cd /
mkdir embedded
cd embedded
```

Copy Components

Copy the latest RM3 Linux components to the “/embedded” directory. Sources can be distributed in different ways but, usually they can be downloaded straight from our FTP server (ftp://drivers.bluechiptechnology.co.uk/Linux/).
Browse the FTP server at location ftp://drivers.bluechiptechnology.co.uk/Linux/RM3-HBx for the latest source code release. Download the Linux build components using the command:

```
wget <Path to RM3 Linux download>
```

Extract Tar ball

Extract the tar ball by issuing the command:

```
tar –xvjf RM3HBx_Linux_v4.0.0.1.tar.bz2
```

Once extracted, the build components will be laid out in the following structure on the development machine. The first directory (“embedded”) is the folder created in the root of the file system.

Directory	Description
/embedded/linux-3.0.35_4.0.0	Linux kernel source code
/embedded/u-boot-2009.08_4.0.0	U-Boot source code
/embedded/rootfs	Staging directory used for holding the root file system of the target device during development

Development Machine Setup

Where possible build scripts have been provided for the various components included with the Linux SDK for BCT RM3.

Before compiling the various components of Linux we must set some environment variables. This is to ensure the configuration tools build for the correct architecture and can find the cross compiling tool chain.

Installing the Tool Chain

Location

By convention, tool chains are usually installed to "/opt", so the following instructions assume that the user has changed to that directory.

```
cd /opt
```

Copy Components

Copy the tool chain to the development PC. Tools can be distributed in different ways but usually they can be downloaded straight from our FTP server (ftp://drivers.bluechiptechnology.co.uk/Linux/).

Browse the FTP server at location ftp://drivers.bluechiptechnology.co.uk/Linux/ for the latest tool chain release. Download using the command:

```
wget <Path to tool chain download>
```

Extract Tar ball

Extract the tar ball by issuing the command:

```
tar –xvjf gcc-linaro-arm-gnueabihf-2012.06-20120625_linux.tar.bz2
```

replace *.tar.bz2 with the latest tar ball downloaded from the FTP site.

Once extracted, the tool chain will be laid out in the following structure on the development machine.

Directory	Directory	Directory	Comments
/opt	gcc-linaro-arm-linux-gnueabihf-2012.06-20120325_linux	arm-linux-gnueabihf	
		bin	The build tools are in here
		lib	
		libexec	
		share	

The extracted gcc* directory shown above may be different depending on the version of tool chain downloaded

Add tool chain to the PATH

The gcc* directory extracted should be added to the PATH variable. For the example above "/opt/gcc-linaro-arm-linux-gnueabihf-2012.06-20120325_linux/bin" the following command can be used:

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-2012.06-20120325_linux/bin:$PATH
```

Building U-Boot

Environment variables

These environment variables need to be set before building U-Boot:

```
export CROSS_COMPILE=arm-linux-gnueabihf-
```

Make targets

There are three targets for "make" that can be used:

Command	Description
make distclean	Cleans all intermediate files and outputs
make bctrm3_config	Configures the source code for RM3
make	Builds the configured source code

A script file is provided (called "build.sh"), which contains all target commands and does a complete build from scratch. This can be run from the u-boot directory:

```
cd /embedded/u-boot-2009.08_4.0.0
./build.sh
```

The binary output can then be found at "/embedded/u-boot-2009.08_4.0.0/u-boot.bin".

Building the Linux Kernel

Environment variables

These environment variables need to be set before building the kernel:

```
export CROSS_COMPILE=arm-linux-gnueabihf-

export ARCH=arm
```

To create an image compatible with U-Boot, the U-Boot utility "mkimage" must be in the PATH.

```
export PATH=/embedded/u-boot-2009.08_4.0.0/tools:$PATH
```

Host tool chain

The kernel build process will also build a few utilities that run on the development PC as part of the build (for example, the source code configuration is done by a utility). In order to build these utilities, the development PC needs to have a native (x86) tool chain as well as the cross-compiler (ARM) tool chain.

Suitable tools are included in the standard Ubuntu 12.04 LTS distribution, so no additional actions are required. Other distributions may differ.

Make targets

There are three targets for "make" that can be used:

Command	Description
make distclean	Cleans all intermediate files and outputs
make bctrm3hb2_defconfig	Configures the source code for RM3 & HB2/HB3
make uImage modules	Builds the configured source code

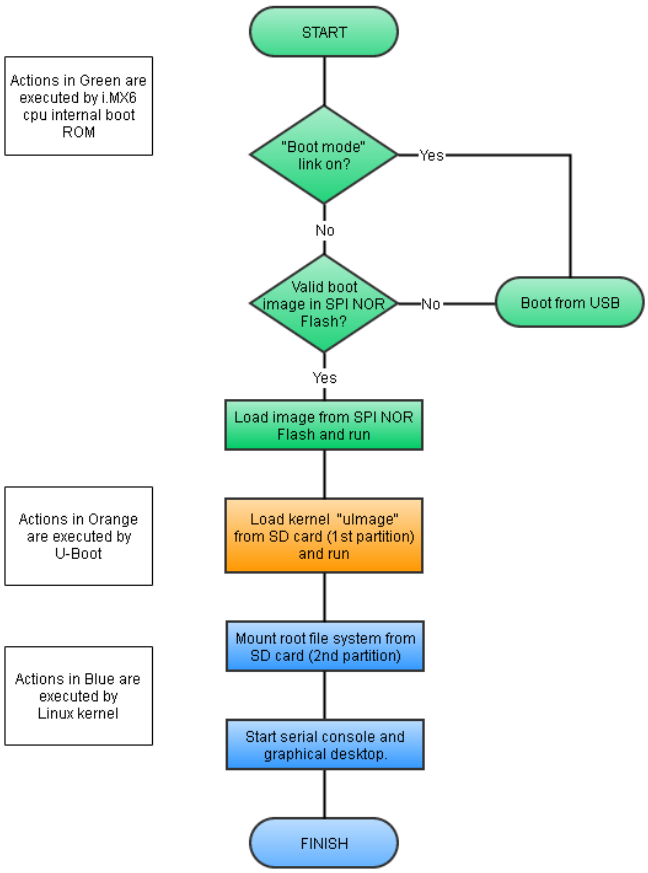
A script file is provided (called "fullbuild.sh"), which contains all target commands and does a complete build from scratch. This can be run from the kernel directory:

```
cd /embedded/linux-3.0.15_4.0.0

./fullbuild.sh
```

RM3 Linux Boot Process

The following diagram shows how Linux boots up on a RM3 module:



Preparing the SD Card

The following details have been tested on a 4GB SD card, but other card sizes should also work.

Partitions

The SD card needs at least two partitions for a minimum system.

Partition	Partition Type	File System	Size	Contents	Read by
1	Primary, type c, W95 FAT32 (LBA)	msdos	60MiB	kernel image	U-Boot
2	Primary, type 83	ext3	Remainder of SD card	Linux root file system	Linux kernel

1. Plug an SD card into the development machine via a USB adapter
2. Issue command, “dmesg | tail”, to determine the device name of the SD Card (e.g. "/dev/sdf")
3. Issue the following commands followed by return.
    1. fdisk /dev/sdf - replace /dev/sdf with device name of SD Card determined in step 2.
    2. p – print the current partition table
    3. d – Delete command
    4. 1 – Delete existing partition 1(will be selected by default if it is the only partition)
    5. Repeat steps c & d to delete any other partitions 2 – 4 as necessary
    6. n – Create new partition
    7. p – Primary partition
    8. 1 – Partition 1
    9. Return for default start cylinder
    10. +60M - Create a 60MB partition
    11. t – Set partition type (will select partition 1 automatically)
    12. c – Set partition type to fat
    13. n – Create new partition
    14. p – Primary partition
    15. 2 – Partition 2
    16. Return for default start cylinder
    17. Return for default end cylinder
    18. t – Set partition type
    19. 2 - Set partition type of partition 2
    20. 83 – Set partition type to Linux
    21. a – Set partition active
    22. 1 - Set partition 1 active
    23. w – Write partitions tables to SD-Card
4. Issue command, “sync”, to flush the SD-Card
5. Unplug and reconnect the SD-Card
6. Issue command, “dmesg | tail”, to determine the device name of each partition (e.g. "/dev/sdf1", "/dev/sdf2")
7. Issue command, “mkfs.msdos /dev/sdf1”, to format the first SD card partition as FAT.
8. Issue command, “mkfs.ext3 /dev/sdf2”, to format the second SD card partition as EXT3.

The SD card is now ready for the Linux files to be copied (see Copying Linux files to the SD card).

Copying Linux Files to the SD Card

Once the SD card has been prepared (see Preparing the SD card) the Linux kernel and root file system can be copied as described below.

First, create a suitable directory for mounting the SD card:

```
mkdir /media/sdcard
```

Next, mount the first partition and copy the kernel image to it:

```
mount /dev/sdf1 /media/sdcard
cp /embedded/linux-3.0.35_4.0.0arch/arm/boot/uImage /media/sdcard/
umount /media/sdcard
```

When the kernel was built, the dynamic load modules were also built. These need to be updated in the root file system so that the kernel can access them once it boots. There is a special "make" target for this purpose:

```
cd /embedded/linux-3.0.35_4.0.0
make INSTALL_MOD_PATH=/embedded/rootfs modules_install
```

Now, the root file system (complete with updated modules) can be copied to the second partition on the SD card:

```
mount /dev/sdf2 /media/sdcard/
cp -rp /embedded/rootfs/* /media/sdcard/
umount /media/sdcard
```

Now, the SD card can be removed from the development PC and used in the host board with RM3.

BCT RM3 Hardware Setup

Linux and U-boot for BCT RM3 heavily rely on access to a serial console. By default U-boot and Linux are configured to use the UARTB port available. By default the board is set to communicate at 115200, 8, n, 1. Before turning on the BCT-RM3 for the first time it is recommended that this port be connected to a PC with terminal emulator software running. E.g. HyperTerminal.

Booting Ubuntu Linux on BCT RM3 platforms from SD-Card

Setup the BCT RM3 hardware on the required peripherals, and with UARTB connected to a PC terminal. Insert the SD-Card, and power on the BCT RM3. Booting the entire Linux system from SD-Card is the default configuration of U-boot, so the system should boot the Ubuntu desktop without requiring special configuration.