DM37x Linux 4.9.y BSP User Guide

BSP Documentation

Logic PD // Products
Published: May 2018

Revision History

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	AF, BSB	-Initial Release	BSB	06/08/16
B	AF, BSB	- Section 2: Update for uImage support. - Sections 4-7: New Throughout: Clean up and merge areas where appropriate. Replace all build machine prompts with '$' instead of full prompt to save space. - Section 4.23: Add WiFi instructions	BSB	08/22/2017
C	AF, BSB	Update kernel to 4.9.81 Section 4.24: Add instruction to enable audio output Section 4.25: Add instructions to enable video capture Section 3.4: Added Falcon Mode	BSB	05/07/2018

Table of Contents

1 Introduction 1
1.1 Nomenclature 1
1.2 Development Resources 1
1.3 Mainstream LTS Linux Kernel 1
1.4 System Prerequisites 2
1.5 Precautionary Statement 2
1.6 Paths to Linux Platform Development 2
2 Build Source 3
2.1 Virtual Machine 3
2.2 Prepare Host PC 4
2.2.1 Ubuntu 16.04 Host Computer Prerequisites 4
2.3 Set Path to the Location of Toolchain 4
2.4 Build Kernel, RootFS, U-Boot and the DTB 5
2.4.1 Image Files Available after Build 6
2.4.2 Location of Source Files after Building 7
2.5 Manually Build Kernel 7
2.5.1 Download and Checkout Kernel 7
2.5.2 Choose Kernel Defconfig 8
2.5.3 Build the Device Tree 8
2.5.4 Build zImage 9
2.5.5 Build uImage 9
2.5.6 Build Kernel Modules 9
2.5.7 Installing Kernel Modules 9
2.5.8 Manually Build kernel from Buildroot location 9
2.6 Appending Device Tree to zImage 10
3 Boot Configuration 11
3.1 Boot Sequence 11
3.2 Boot Options 11
3.2.1 Boot EXT3 RootFS from SD Card 11
3.2.2 Boot RAMDisk Image from SD Card 13
3.2.3 Boot UBI RootFS from NAND 14
3.2.4 Boot RAMDisk Image from NAND 15
3.3 U-Boot 17
3.3.1 Get Started 17
3.3.2 U-Boot help Command 18
3.3.3 U-Boot Environment 20
3.3.4 Shell Variables 22
3.3.5 Updating Kernel Command Line 22
3.3.6 View the device tree in U-Boot 22
3.4 Falcon Mode 23
3.4.1 Common Falcon Mode Setup 23
3.4.2 Falcon Mode NAND 24
3.4.3 Falcon Mode SD 26
4 Kernel 29
4.1 vi Editor 29
4.2 Retrieve BSP Version 30
4.3 Display Linux System Information 30
4.4 Wired Networking 32
4.4.1 Assign Development Kit IP Address 32
4.4.2 Set Speed, Duplex, and Auto-Negotiate 35
4.4.3 Cable Detection 35
4.5 Linux Processes 35
4.5.1 ps Command 35
4.5.2 kill Command 36
4.6 Video Display 36
4.6.1 Framebuffer Tests 37
4.6.2 Backlight 38

4.6.3 Console Blanking 38
4.6.4 Display Message 38
4.7 SD/MMC Interface 39
4.8 Touch Screen 39
4.8.1 Configuration 39
4.8.2 Calibration 39
4.8.3 Test Touch 39
4.9 Built-in Flash Storage via MTD 39
4.10 USB Controller 40
4.11 USB Host Controller 40
4.12 Processor OTG Controller 40
4.12.1 Use MUSB in Host Mode 40
4.12.2 Device Mode Gadgets 41
4.13 UART 43
4.14 Real Time Clock 43
4.15 BQ27000 Gas Gauge Support 44
4.16 Run/Idle/Suspend 45
4.17 Virtual Files 46
4.17.1 echo Command and ">" Operator 46
4.17.2 Debug FS 46
4.18 Shut Down Linux System 48
4.19 CPU Benchmarks 48
4.20 Use devmem to Examine/Modify Registers 51
4.21 Filesystem Commands 51
4.21.1 df Command 51
4.21.2 cat /proc/mtd Command 52
4.22 Using Linux Voltage and Current Regulator Framework 52
4.22.1 Read Regulator Names 53
4.22.2 Read Current Regulator Voltage 53
4.22.3 Read Minimum Regulator Voltage 53
4.22.4 Read Maximum Regulator Voltage 53
4.22.5 Read Regulator State 53
4.23 WL12xx WiFi 54
4.23.1 WiFi MAC Address 54
4.23.2 WiFi Station Mode 55
4.23.3 WiFi Access Point 56
4.23.4 Removing WIFI Support 57
4.24 Audio Output 59
4.25 Camera Capture 62
5 Boot Kernel from TFTP Server and Root Filesystem from NFS Server 66
5.1 Set Up TFTP Server in Ubuntu 16.04 66
5.2 Setup NFS Server in Ubuntu 16.04 67
5.3 Set Up the DM3730/AM3703 Target Platform 68
6 Application Development 70
7 Basic Driver/Kernel Debugging Information 71
7.1 Logs 71
7.2 Debug Modules 71

1 Introduction

This user guide provides information pertaining to Logic PD's DM37x Linux Board Support Package (BSP). This BSP is compatible with the DM3730/AM3703 SOM-LV, DM3730/AM3703 Torpedo SOM, and DM3730/AM3703 Torpedo + Wireless SOM platforms.

1.1 Nomenclature

Within this document, use of "DM3730 Development Kit" suggests text that applies to both the DM3730 SOM-LV Development Kit and DM3730 Torpedo SOM Development Kit; information specific to one development kit will call out the precise name.

1.2 Development Resources

This document does not attempt to provide information about all commands and all features available in Linux, U-Boot, device tree blob or any other utilities included in the BSP. The purpose of this document is to provide information to a developer on how to get started doing development with this BSP. This document will also provide information regarding any commands that help the developer use various features within the BSP. Linux, U-Boot, device tree blob, and all the utilities used in the BSP are open source. The open source community has a wealth of information available to assist developers. If you need additional information beyond what is provided in this document, please consider these sources:

- The source code included in the BSP: No other information source is more detailed or more accurate than the source code that is actually being built. The source code is, however, extensive and impossible to navigate without a text search utility. Most modern source code editors all have multi-file search features.
- Web pages: Use your favorite search engine to find articles on the open-source component you are working with.
- Forums: If you have questions about an open-source component, others have probably had the same questions. By searching available forums, you are likely to not only find answers, but to also find people who have resolved the same task you are working on.
- Logic PD Support: Logic PD provides many free resources that are included with your development kit, once it is registered online, to help with your development process. Please visit the Logic PD *Support* web page[1] for additional information about the offerings available to you.
- Logic PD Design Services Support: For customers requiring continued support with development, Logic PD offers standard and configurable support contract options. Logic PD has a design services division that is not only experienced with the Linux BSP, but also in mechanical engineering, software engineering, hardware engineering, and manufacturing engineering. Please visit the Logic PD *Support Packages* web page[2] for additional information.

1.3 Mainstream LTS Linux Kernel

This document provides instructions on how to build the mainstream kernel. These instructions use an open source tool called 'Buildroot' which can be used to build the cross-compiler for the OMAP3 processor as well as generate the necessary files to make a rootfile system. When directed to build, it can also build the Linux kernel and U-Boot bootloader.

[1] https://support.logicpd.com/
[2] https://support.logicpd.com/TechnicalSupport/SupportPackages.aspx

Because these tools are all in the mainstream, they do not necessarily have Logic PD enhancements or Logic PD specific code and as such, any files listed here are considered 'as-is' and without warranty. No feature of any Logic PD SOM is guaranteed to work with these instructions or code base. Some items on the Logic PD baseboard do not have mainstream drivers, and therefore will not work. For people who wish to have a modern kernel with long-term-support from the mainstream community, this build can form the foundation for the code base. Logic PD can aid in developing additional features and/or functionality for an additional cost with a services contract. Logic PD can also aid with customer specific hardware and software design where necessary for an additional cost with a services contract.

Community involvement is encouraged and users should go to http://support.logicpd.com/TDGForum.aspx or join the open source OMAP mailing list for the Linux kernel community at http://vger.kernel.org/vger-lists.html#linux-omap.

1.4 System Prerequisites

- Zoom DM3730 Development Kit registered on Logic PD's website[3]
  - Registration is required to gain access to your product's download page where the BSP can be downloaded.
- Host PC
- Internet connection

1.5 Precautionary Statement

Caution should be observed when cutting and pasting commands from this document to the Linux prompt. Word or PDF documents may convert the simple hyphen-minus character (hex code 0x2d, uni code \055) into the en dash character (hex code 2013, uni code \342\200\223). Visually, it is difficult to distinguish between the two characters, and the applications being passed these unexpected arguments may simply ignore them.

1.6 Paths to Linux Platform Development

In most cases, customers require an additional driver or other modification to the existing BSP to support hardware on their product. The source for the Linux BSP is provided in its entirety so that customers can add to or modify any part of it as needed to meet their customization needs.

The mainstream BSP build is made available following the procedure described in section 2.

[3] http://support.logicpd.com/TechnicalSupport/RegisterProduct.aspx

2 Build Source

This section describes how to build X-Loader, U-Boot, the kernel, and the RAM-based Linux root filesystem using mainstream Linux. For mainstream linux, Logic PD recommends using a Buildroot environment. Buildroot is an open-source tool used to develop and deploy BSPs for various target platforms.

Buildroot preforms the following tasks:

- Installs the cross compiler. The cross compiler, runs on a desktop Linux PC to compile code for the target SOM.
- Manages package dependencies. A typical Linux build consists of more than just the Linux OS. Most often the features users employ to interact with Linux are handled by some kind of package. The list of packages is vast, and a typical Linux image may contain nearly 100 packages. Some packages used in the DM37x Linux BSP are:
  - Bash
  - Dropbear
  - GDB
  - DirectFB
  - ALSA-utils
  - BlueZ
  - Samba
  - Many others are included, and many others can be added.
- Builds the Linux Kernel from source. Buildroot will use the included cross compiler to build the Linux OS image. This is one of the first images loaded when the SOM boots.
- Builds/includes all the configured packages. Buildroot can be configured to add or remove a long list of included packages. Each package can have binaries and/or source. Buildroot will manage building and linking all the source and binaries of every package.
- Applies patches. At times a change is required for a package or the kernel. These changes may be included as a patch. Buildroot will apply any necessary patches to the build. You may also add your own patches.
- Creates a root filesystem. The Linux OS requires some form of root filesystem. The root filesystem will contain all the packages, utilities, and applications the user will use to interact with the system. Buildroot will create an image of the root filesystem that users can install on their hardware.
- Aids in development. Buildroot allows the user to build the system image or portions of it. Often when doing development, a full OS build is not necessary and can be time consuming.

2.1 Virtual Machine

For customers who wish to build the source with the least amount of effort, Logic PD now provides a VM that is fully configured with the Ubuntu OS, mainstream source and build tools. The Virtual Machine SDK for the DM37x Mainstream Linux BSP[4] has all building capabilities on the host PC as described in Section 2.3 below. The VM also provides a means for building the BSP on a host PC not equipped with Linux. The VM is configured to use VirtualBox as the VM

[4] https://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=3144

manager. See the *Virtual Machine SDK for the DM37x Linux BSP ReadMe*[5] for installation instructions.

If you choose to use the VM, building occurs just like building on a native host. See Section 2.3 to continue building the BSP.

2.2 Prepare Host PC

The mainstream Linux is designed to be executed on a Linux host PC. The following steps should work for any Linux distribution.

2.2.1 Ubuntu 16.04 Host Computer Prerequisites

The following packages are generally needed by the mainstream build and some of the standard components within the BSP source. Every attempt has been made to make the list as complete as possible. Depending on how your host PC has been configured, some of the packages may already be installed. If that is the case, you should be able to safely ignore them. Conversely, your host PC may require additional packages not included below. In order to determine if this is the case, please review any system build logs to identify errors. By reviewing the programs that failed to build and identifying how they failed, you can ascertain which package your host PC is missing.

The following prerequisites must be installed on Ubuntu before the system can be built.

```
$ sudo apt-get install build-essential
$ sudo apt-get install libncurses5-dev
$ sudo apt-get install git
$ sudo apt-get install libssl-dev
$ sudo apt-get install swig
$ sudo apt-get install python
$ sudo apt-get install python-dev
```

Developers having issues building in their host environment might consider using the DM37x Linux Mainstream VM made available by Logic PD.

2.3 Set Path to the Location of Toolchain

The Toolchain is generated using Buildroot. It will be located: /home/logic/buildroot/output/host/usr/bin. To manually build the kernel or kernel modules, or cross-compile applications, it's easiest to export the PATH of the toolchain.

```
$ export PATH=$PATH:/home/logic/logic/buildroot/output/host/usr/bin
```

From here, building the kernel, modules, and device tree blob is easier.

Optional: Adding the line above in Ubuntu to ~/.bashrc will automatically setup your environment any new xterm windows.

[5] https://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=3048

2.4 Build Kernel, RootFS, U-Boot and the DTB

This will build the filesystem in a couple different methods, RAMDisk or Tarball. RAMDisk will create a rootfs that can be loaded from SD card or NAND and executed from RAM. This provides a method for people who want a consistent bootup each time, because any changes are lost on bootup. For people who want to use the SD card or NAND, we create the tarball which can be extracted to SD card or NAND, and booted from there. Changes to this system are considered permanent. The default build system will automatically download U-Boot and compile it and the kernel device tree blob (dtb) for the DM37 Torpedo.

The Buildroot used for this BSP is 2018.02 and the Kernel used for this BSP is 4.9.89. Checking out the GIT repositories allows people to fix bugs or update the system when updates are released. There is no guarantee that future updates will remain backwards compatibility, so using the specified revisions is recommended.

1. Create Directory Structure:

```
$ mkdir logic
$ cd logic
```

2. Download and checkout Buildroot:

Note: If upgrading from a previous version, skip this step, but make sure to *'cd buildroot'*, *'rm configs/omap3logic_derconfig'* then *'git checkout master'* and finally *'git pull*' before continuing to step 3.

```
$ git clone git://git.buildroot.net/buildroot
$ cd buildroot
```

3. Optional: This was verified using 2018.03 version. To checkout this version using the following:

```
$ git checkout 2018.02
```

4. **Note**: Using 'git tag -l' allows for a list of released version of Buildroot. Use this option with caution as the version tested is seen in the previous command. Copy the patch provided to the buildroot directory.

```
$ cp (path to patch)/0001-Create-LPD-4.9.y-BSP.patch .
```

5. Apply only the omap3logic_buildroot patch to configure the build:

```
$ patch -p1 -i 0001-Create-LPD-4.9.y-BSP.patch
patching file configs/omap3logic_defconfig
patching file lpd-linux-4.9.y.patch
```

6. Skip if this step if this is your first build.

```
$ make uboot-dirclean
$ make linux-dirclean
```


Alternately, you could use the following command to clean everything:

```
$ make clean
```

7. Create the .config file

```
$ make omap3logic_defconfig
```

8. Build System

```
$ make
```

Note: If the build fails when attempting to build U-Boot, please do the following:

```
$ cd output/build/uboot-2018.03
$ export PATH=$PATH:/home/logic/src/buildroot/output/host/usr/bin
$ make ARCH=arm CROSS_COMPILE=arm-linux- -j8
$ cd ../../..
$ make
```

Here is the error being seen when building U-boot the first time:

```
ImportError: No module named _libfdt
scripts/Makefile.spl:270: recipe for target 'spl/dts/dt-platdata.c' failed
make[3]: *** [spl/dts/dt-platdata.c] Error 1
make[3]: *** Waiting for unfinished jobs....
scripts/Makefile.spl:267: recipe for target 'include/generated/dt-structs-gen.h' failed
make[3]: *** [include/generated/dt-structs-gen.h] Error 1
Makefile:1446: recipe for target 'spl/u-boot-spl' failed
make[2]: *** [spl/u-boot-spl] Error 2
package/pkg-generic.mk:247: recipe for target
'/home/logic/logic/buildroot/output/build/uboot-2018.03/.stamp_built' failed
make[1]: *** [/home/logic/logic/buildroot/output/build/uboot-2018.03/.stamp_built] Error 2
Makefile:79: recipe for target '_all' failed
make: *** [_all] Error 2
```

2.4.1 Image Files Available after Build

Upon a successfully completion of the build, the following files will be available in the ~/logic/buildroot/output/images folder:

- logicpd-torpedo-37xx-devkit.dtb – DM37x Torpedo / Torpedo + Wireless device tree
- logicpd-som-lv-37xx-devkit.dtb – DM37x SOM-LV device tree
- MLO – The Secondary Program Loader (SPL) created from U-Boot
- rootfs.cpio – Ram-based rootfs

- rootfs.cpio.uboot – RAM-based root filesystem for loading from U-Boot
- rootfs.tar – root file system with kernel modules
- rootfs.ubifs – Root file system in UBI format
- U-Boot .img – U-Boot with additional header information used to determine how and where to load and execute U-Boot.
- zImage - a compressed version of the Linux kernel image that is self-extracting

2.4.2 Location of Source Files after Building

After building of the images using buildroot the source files for the images above can be located in the following folders from within the ~/buildroot/output/build directory.

- The device tree source files can be found here: linux-4.9.y/arch/arm/boot/dts
- The linux source files can be found here; linux-4.9.y
- The U-Boot source files can be found here: uboot-201x.0x.0x

2.5 Manually Build Kernel

The Buildroot system occasionally creates a larger kernel than manually building it. The reason is currently unknown. The NAND partition size is limited to 4MB, so building manually may be required.

Setup the toolchain PATH as seen in section 2.3. The defconfig in the mainstream is called omap2plus_defconfig. Unfortunately, this is lacking some drivers, so using omap3_logic_kernel_defconfig will help speed up the build. Only copy the defconfig to the config directory once. Then setup the kernel to build from this defconfig. Finally, build the device tree blob, kernel image, and kernel modules.

Note: In the following example, the '-j4' flag represents a computer with 4 processor threads. This number could be raised or lowered depending on the number of available processors threads. If the computer only has 1 available processor thread, the flag may be eliminated.

2.5.1 Download and Checkout Kernel

Newer stable kernels may exist, but this document was tested against the linux-4.9.y stable branch:

```
$ cd ~/logic
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
$ cd linux-stable
$ git checkout linux-4.9.y
```

Switched to branch 'linux-4.9.y'

Your branch is up-to-date with 'origin/linux-4.9.y'

Optional: This was verified using linux-4.9.89 version. To checkout this version using the following:


```
$ git checkout -b stable v4.9.89
```

Copy the lpd-linux4.9.y.patch provided to the linux-stable directory.

```
$ cp (path to patch)/lpd-linux4.9.y.patch .
```

Optional: Logic PD has a small patch set which enables the backlight, touchscreen, configures the default LCD screen to Logic PD type 28, and allows access to the Logic PD Product ID information.

```
$ patch -p1 -i lpd-linux-4.9.y.patch
```

2.5.2 Choose Kernel Defconfig

The default configuration for OMAP2+ processors is omap2plus_defconfig. Logic PD has provided a tailored version of this config file to remove some unnecessary features and add in some missing features called *omap3logic_defconfig*. See section 2.4.

```
$ make  omap3logic_defconfig ARCH=arm
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

2.5.3 Build the Device Tree

Build the device tree. This only needs to happen once unless changes are made to the device tree files. The device tree is made of up two files which represent both the SOM as well as the development kit. The device tree determines which peripherals and devices are connected and which pins are used by the respective modules.

For the DM37x Torpedo or DM37x Torpedo + Wireless:

```
$ make logicpd-torpedo-37xx-devkit.dtb ARCH=arm CROSS_COMPILE=arm-linux- -j4
```

For the DM37x SOM-LV:

```
$ make logicpd-torpedo-37xx-devkit.dtb ARCH=arm CROSS_COMPILE=arm-linux- -j4
```


2.5.4 Build zImage

Newer versions of U-Boot can load a zImage while older versions loaded a uImage. zImage is a compressed version of the kernel while uImge is the kernel with a U-Boot header added. The generated file is located: arch/arm/boot/zImage. See section 2.5.5 for details on building a uImage.

```
$ make zImage ARCH=arm CROSS_COMPILE=arm-linux- -j4
```

2.5.5 Build uImage

Note: For people who wish to make a uImage, Ubuntu 16.04 will need *u-boot-tools* installed.

```
$ sudo apt-get install u-boot-tools
```

Optional: If using the zImage, the uImage is not needed. Building a uImage requires the LOADADDR parameter to be set as an additional parameter. The extra `-j4- parameter is optional and is set based on the number of processor threads available on the build machine. The generated file is located: arch/arm/boot/uImage

```
$ make uImage LOADADDR=0x80008000 ARCH=arm CROSS_COMPILE=arm-linux- -
j4
```

Copy the uImage to the location where the kernel will be loaded (i.e. SD card)

```
$ cp arch/arm/boot/uImage /media/logic/boot
```

2.5.6 Build Kernel Modules

Customer who do not need the kernel modules can skip this step, but most of the drivers are compiled as modules by default.

```
$ make modules ARCH=arm CROSS_COMPILE=arm-linux- -j4
```

2.5.7 Installing Kernel Modules

Install the modules. (i.e. SD card)

```
$ sudo make modules_install ARCH=arm INSTALL_MOD_PATH=/media/logic/rootfs
$ sync
```

2.5.8 Manually Build kernel from Buildroot location

Use this command to access the kernel menu from the Buildroot location


```
$ make linux-menuconfig
```

Next, rebuild the linux kernel

```
$ make linux-rebuild
```

Make the full image (including copying the new kernel to the output/images

```
$ make
```

2.6 Appending Device Tree to zImage

First manually build the Linux Kernel as seen in section 2.4.2. Next you must append DTB to the zImage. This merged zImage + DTB necessary due to the partition structure of NAND and is needed regardless of whether it's a RAMDisk boot or an UBI FS boot. It can also be useful when booting over a network.

Please see the section 2.4.2 on manually building the Kernel and device tree blob. The following example assumes the zImage has been manually built, and copied to the *buildroot/output/images* directory.

```
$ cd ~/logic/linux-stable/arch/arm/boot/
$ cp zImage ~/logic/buildroot/output/images/zImage_Manual
$ cd ~/logic/buildroot/output/images
$ cat logicpd-torpedo-37xx-devkit.dtb  >> zImage_Manual
$ mv zImage_Manual zImage_dtb
```

zImage_dtb now contains both the image and the device tree blob. Confirm the size of the combined zImage+DTB is less than 6MB if booting from a NAND partition.

```
$ ls -l zImage_dtb
```


3 Boot Configuration

Linux can be configured to boot in an infinite number of ways. This section attempts to describe the boot process and some common boot configurations.

3.1 Boot Sequence

Booting occurs in three stages.

1. The first-stage bootloader is an application that runs inside the Central Processing Unit (CPU). When a reset occurs or power is first turned on, the boot ROM inside the processor identifies the environment state and launches the second-stage bootloader.

   The boot ROM has different requirements for each boot source. For example, when booting from an SD card, the boot ROM mandates a specific partition, filesystem, and file name for the second-stage bootloader. When booting from NAND flash, the boot ROM mandates the second-stage bootloader be located in the first four blocks of NAND flash and use a 1-bit hamming ECC algorithm. For other boot ROM requirements, see the reference manual for your processor.

2. The second-stage bootloader is the 1st U-Boot stage as known as the Secondary Program Loader (SPL) or MLO. It must be located in non-volatile storage or on an external link (e.g., UART, USB). The second-stage bootloader is loaded into the CPU SRAM and run from there. This bootloader then starts the SDRAM and loads the third-stage bootloader to SDRAM. The SDRAM is not usable until the second-stage bootloader configures and starts it. SPL is a non-interactive loader and is a specially built version of U-Boot and build concurrently when building U-boot.

3. The third-stage bootloader is U-Boot. It usually has a simple runtime user interface for debugging the SOM, loading the kernel, and writing flash memory. The third-stage bootloader tends to be configurable with splash screens and scripts to customize the boot sequence. This bootloader loads the filesystem, and loads and starts the kernel.

4. Finally, the Linux kernel is started.

The second and third-stage bootloaders must both stored in the same media type (e.g., SD card or NAND). The first-stage bootloader is always on the same substrate as the CPU. For the remainder of this document, the focus will be on SD card and NAND locations. For additional information on loading the bootloader from other locations such as USB, UART, eMMC, please post a question to the Logic PD TDG forum.

3.2 Boot Options

This section describes different methods to boot the Linux OS using a Logic PD DM37x SOM product.

It is assumed all the steps above have been followed, and the system has generated a zImage kernel, device tree blob, U-boot, MLO, and rootfs.

3.2.1 Boot EXT3 RootFS from SD Card

Follow these steps on the Host PC in your Download directory.

```
$ cd ~/Downloads/
$ ~/Downloads$ wget
http://support.logicpd.com/Portals/0/Users/049/05/305/create_sdcard.zip
```



```
--2016-05-12 10:29:14--
http://support.logicpd.com/Portals/0/Users/049/05/305/create_sdcard.zip
Resolving support.logicpd.com (support.logicpd.com)... 10.1.18.38
Connecting to support.logicpd.com (support.logicpd.com)|10.1.18.38|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3076 (3.0K) [application/x-zip-compressed]
Saving to: 'create_sdcard.zip'

create_sdcard.zip
100%[==========================================================
==================================>]   3.00K  --.-KB/s    in 0.06s

2016-05-12 10:29:14 (50.2 KB/s) - 'create_sdcard.zip' saved [3076/3076]

$ unzip create_sdcard.zip
Archive:  create_sdcard.zip
  inflating: create_sdcard.sh
$ chmod +x create_sdcard.sh
$ ./create_sdcard.sh
Devices available:
  sdf is    3.6GB - Card  Reader
Enter device: sdf
[sudo] password for logic: logic
Unmounting /media/logic/BOOT
Setting up sdf

Do you wish to continue? (y/N) y
Partitioning sdf.
mke2fs 1.42.13 (17-May-2015)
Missing "/home/logic/Downloads/rootfs/boot/MLO"! Cannot continue.
$ ~/Downloads$
```

Ignore the error showing that MLO is missing. We are going to copy these files manually.

Remove and reinsert the SD Card so that Ubuntu will mount all partitions.

```
$ cd ~/logic/buildroot
$ cp output/images/MLO /media/logic/boot/
$ cp output/images/u-boot.img /media/logic/boot/

For DM37x Torpedo and Torpedo + Wireless:
$ cp output/images/logicpd-torpedo-37xx-devkit.dtb /media/logic/boot/
For DM37x SOM-LV:
$ cp output/images/logicpd-som-lv-37xx-devkit.dtb /media/logic/boot/

$ cp output/images/zImage /media/logic/boot/
$ sudo tar xvf output/images/rootfs.tar -C /media/logic/rootfs/
$ sync
```

section_3_2_1.sh

Remove the SD Card from your host computer and insert it into the DM37x Development Kit.

Power on the system and press any key to enter U-Boot.

```
OMAP Logic # nand unlock
OMAP Logic # nand erase.chip
OMAP Logic # reset

(Interrupt U-Boot again by pressing a key)

OMAP Logic # nand unlock
OMAP Logic # setenv mmcroot /dev/mmcblk1p2 rw
OMAP Logic # setenv defaultboot 'run mmcbootz'
OMAP Logic # saveenv
OMAP Logic # reset
```

The system should now successfully boot from SD Card using an EXT3 rootfs on the second partition of the SD card.

3.2.2 Boot RAMDisk Image from SD Card

The SD card should be formatted in FAT or FAT32 with 'boot' as the label.

Copy the following files to the SD card. Be sure to always copy the MLO file first.

```
$ cd ~/logic/buildroot
$ cp output/images/MLO /media/logic/boot/
$ cp output/images/u-boot.img /media/logic/boot/

For DM37x Torpedo and Torpedo + Wireless:
$ cp output/images/logicpd-torpedo-37xx-devkit.dtb /media/logic/boot/
For DM37x SOM-LV:
$ cp output/images/logicpd-som-lv-37xx-devkit.dtb /media/logic/boot/

$  cp output/images/zImage /media/logic/boot/
$  cp output/images/rootfs.cpio.uboot /media/logic/boot
$ sync
```

Remove the SD Card from your host computer and insert it into the DM37x Development Kit.

Power on the system and press any key to enter U-Boot .

```
OMAP Logic # nand unlock
OMAP Logic # nand erase.chip
OMAP Logic # reset
```

Pause U-Boot

```
OMAP Logic # nand unlock
OMAP Logic # setenv ramdisksize 100000
OMAP Logic # setenv ramdiskimage rootfs.cpio.uboot
```



```
OMAP Logic # setenv defaultboot 'mmc rescan; run mmcrambootz'
OMAP Logic # setenv fdtaddr 0x88000000
OMAP Logic # setenv mmcrambootz 'setenv bootfile zImage; run
mmcrambootcommon; bootz ${loadaddr} ${rdaddr} ${fdtaddr}'
OMAP Logic # saveenv
OMAP Logic # reset
```

The system will now successfully booted from SD Card using a RAMDisk image.

3.2.3 Boot UBI RootFS from NAND

Buildroot is able to construct a UBIFS image in a way that can be programmed by U-Boot and read my Linux.

The SD card should be formatted in FAT or FAT32 with 'boot' as the label.

Copy the following files to the SD card. Be sure to always copy the MLO file first.

```
$ cd ~/logic/buildroot
$ cp output/images/MLO /media/logic/boot/
$ cp output/images/u-boot.img /media/logic/boot/

For DM37x Torpedo and Torpedo + Wireless:
$ cp output/images/logicpd-torpedo-37xx-devkit.dtb /media/logic/boot/
For DM37x SOM-LV:
$ cp output/images/logicpd-som-lv-37xx-devkit.dtb /media/logic/boot

$ cp output/images/zImage /media/logic/boot/
$ cp output/images/rootfs.ubifs /media/logic/boot
$ sync
```

1. Remove the SD card and boot to U-Boot.
2. From U-Boot, erase the contents of flash, and reset U-Boot environmental variables.

```
OMAP Logic # nand unlock
OMAP Logic # nand erase.chip
OMAP Logic # env default -a
```

3. Program UBIFS image to NAND

```
OMAP Logic # fatload mmc 0 $loadaddr rootfs.ubifs
OMAP Logic # ubi part fs
OMAP Logic # ubi create rootfs
OMAP Logic # ubi write $loadaddr rootfs $filesize
```

4. Program the Linux kernel into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $loadaddr zImage
OMAP Logic # nand write $loadaddr kernel $filesize
```


5. Program the DTB into the corresponding NAND partition.

```
For DM37x Torpedo and Torpedo + Wireless:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-torpedo-37xx-devkit.dtb

For DM37x SOM-LV:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-som-lv-37xx-devkit.dtb

OMAP Logic # nand write $fdtaddr spl-os $filesize
```

6. Program U-Boot into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $loadaddr u-boot.img
OMAP Logic # nand write $loadaddr u-boot $filesize
```

7. With U-Boot, the Kernel, the DTB, and the RootFS programmed into NAND, set the environmental variables to boot from NAND using the UBI partition as the RootFS.

```
OMAP Logic # setenv loadzimage 'nand read $loadaddr kernel'
OMAP Logic # setenv loadfdtimage 'nand read $fdtaddr spl-os'
OMAP Logic # setenv nandbootz 'nand unlock; run nandargs; run common_bootargs;
run dump_bootargs; run loadzimage; run loadfdtimage; bootz ${loadaddr} -
${fdtaddr}'
OMAP Logic # setenv defaultboot 'run nandbootz'
OMAP Logic # saveenv
```

8. Program the MLO with HW ECC last into the corresponding NAND partition.

Note: MLO must be done after all the other changes are written to NAND, because the ECC for MLO is different than the ECC for the rest of the NAND partitions.

```
OMAP Logic # nandecc hw
OMAP Logic # fatload mmc 0 $loadaddr MLO
OMAP Logic # nand write $loadaddr MLO $filesize
```

1. Remove the SD Card from the target system.
2. The system must be reset or power cycled to reset the ECC configuration.

```
OMAP Logic # reset
```

The system should now successfully boot from NAND using the generated UBI rootfs.

3.2.4 Boot RAMDisk Image from NAND

1. To boot using RAMDisk image from NAND the followings steps are required.
2. Create an SD Card that can boot from RAMDisk Image as described in section 3.2.2
3. Reset Environmental Variables to Default


```
OMAP Logic # nand unlock
OMAP Logic # nand erase.chip
OMAP Logic # reset
```

4. Pause U-boot

```
OMAP Logic # nand unlock
```

5. Program the Linux kernel into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $loadaddr zImage
OMAP Logic # nand write $loadaddr kernel $filesize
```

6. Program the device tree into the corresponding NAND partition

```
OMAP Logic # setenv fdtaddr 0x88000000

For DM37x Torpedo and Torpedo + Wireless:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-torpedo-37xx-devkit.dtb

For DM37x SOM-LV:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-som-lv-37xx-devkit.dtb

OMAP Logic # nand write $fdtaddr spl-os $filesize
```

7. Program U-Boot into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $loadaddr u-boot.img
OMAP Logic # nand write $loadaddr u-boot $filesize
```

8. Program RAMDisk rootfs into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $ramdisk_addr_r rootfs.cpio.uboot
OMAP Logic # nand write $ramdisk_addr_r  fs $filesize
OMAP Logic # setenv ramdisksize $filesize
OMAP Logic # saveenv
```

9. Program the MLO with HW ECC last into the corresponding NAND partition.

Note: MLO must be done after all the other changes are written to NAND, because the ECC for MLO is different than the ECC for the rest of the NAND partitions.

```
OMAP Logic # nandecc hw
OMAP Logic # fatload mmc 0 $loadaddr MLO
OMAP Logic # nand write $loadaddr MLO $filesize
```

10. Remove the SD Card from the target system.
11. The system must be reset or power cycled to remove the 'nandecc hw' configuration.


```
OMAP Logic # reset
```

12. Set U-Boot Parameters to Load RAMDisk from NAND

```
OMAP Logic # nand unlock
OMAP Logic # setenv loadzimage 'nand read $loadaddr kernel'
OMAP Logic # setenv loadfdtimage 'nand read $fdtaddr spl-os'
OMAP Logic # setenv loadramdisk 'nand read $ramdisk_addr_r fs $ramdisksize'
OMAP Logic # setenv defaultboot 'run ramargs; run common_bootargs; run
dump_bootargs; run loadzimage; run loadfdtimage; run loadramdisk; bootz
${loadaddr} ${ramdisk_addr_r} ${fdtaddr}'
OMAP Logic # saveenv
OMAP Logic # boot
```

The system should now successfully boot from NAND using a RAMDisk rootfs.

3.3 U-Boot

The intent of this section is to familiarize users with the most frequently used commands of the U-Boot shell. Logic PD provides all U-Boot source code to kit users so that detailed understanding and modification can be made with the source. A complete U-Boot manual can be found on the DENX website.[6]

Under normal circumstances when U-Boot runs, it waits a few seconds for keyboard input; if none is received, U-Boot proceeds to load and launch the Linux kernel. If, however, a key press is detected, U-Boot will start a shell and wait for further input from the user.

U-Boot has many capabilities for performing simple debug and maintenance tasks, including:

- Reading and writing RAM and flash devices
- Loading images using SD/MMC, and NAND
- Displaying splash screens on the display (available upon request)
- Reading and writing SD/MMC
- Accessing a UBI NAND flash filesystem
- Performing scripting tasks
- Configuring a Linux boot parameters

3.3.1 Get Started

When using U-Boot, it is important to remember the following:

- U-Boot will check NAND flash for environment variables at startup. This can sometimes produce confusing results when developers switch from booting from NAND flash to booting from an SD card. When booting from any source, U-Boot checks NAND flash for the environment state and uses it if found.
- U-Boot shell variables contain values, as well as a list of commands that can be run as a script.
- Linux requires the following to launch:
    - Loading the kernel image (z*Image or uImage*) to RAM.

[6] http://www.denx.de/wiki/DULG/Manual

- Loading the RAM-based filesystem to RAM, if using a RAM-based filesystem.
- Setting up the *bootargs* environment variable for Linux. If using a network-based filesystem, that information needs to be included in the *bootargs* variable.
- Flattened Device Tree which can optionally be appended to zImage or uImage.

3.3.2 U-Boot *help* Command

U-Boot supports the *help* command. At the U-Boot prompt, typing *help* will display all the U-Boot commands and a brief description of each command. In addition, typing *help* <*command*> provides a more detailed description of a specific command and its usage.

NOTE: Depending on the U-Boot version being used, the list of commands may differ from the example shown below.

```
OMAP Logic # help
?       - alias for 'help'
askenv  - get environment variables from stdin
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootefi - Boots an EFI payload from memory
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
bootz   - boot Linux zImage image from memory
chpart  - change active partition
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
dcache  - enable or disable data cache
dfu     - Device Firmware Upgrade
dhcp    - boot image via network using DHCP/TFTP protocol
dm      - Driver model low level access
echo    - echo args to console
editenv - edit environment variable
env     - environment handling commands
exit    - exit script
ext2load- load binary file from a Ext2 filesystem
ext2ls  - list files in a directory (default /)
ext4load- load binary file from a Ext4 filesystem
ext4ls  - list files in a directory (default /)
ext4size- determine a file's size
ext4write- create a file in the root directory
false   - do nothing, unsuccessfully
fastboot- use USB Fastboot protocol
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fatsize - determine a file's size
fatwrite- write file into a dos filesystem
```



```
fdt     - flattened device tree utility commands
fstype  - Look up a filesystem type
go      - start application at address 'addr'
gpio    - query and control gpio pins
gpt     - GUID Partition Table
help    - print command description/usage
i2c     - I2C sub-system
icache  - enable or disable instruction cache
imxtract- extract a part of a multi-image
itest   - return true/false on integer compare
load    - load binary file from a filesystem
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loadx   - load binary file over serial line (xmodem mode)
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
mtdparts- define flash/nand partitions
mw      - memory write (fill)
nand    - NAND sub-system
nandecc - switch OMAP3 NAND ECC calculation algorithm
nboot   - boot from NAND device
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
part    - disk partition related commands
ping    - send ICMP ECHO REQUEST to network host
poweroff- Perform POWEROFF of the device
printenv- print environment variables
pxe     - commands to get and boot from pxe files
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
save    - save file to a filesystem
saveenv - save environment variables to persistent storage
setenv  - set environment variables
setexpr - set environment variable as the result of eval expression
showvar - print local hushshell variables
size    - determine a file's size
sleep   - delay execution for some time
source  - run script from memory
spl     - SPL configuration
sspi    - SPI utility command
sysboot - command to get and boot from syslinux files
test    - minimal test like /bin/sh
tftpboot- boot image via network using TFTP protocol
time    - run commands and summarize execution time
true    - do nothing, successfully
ubi     - ubi commands
ubifsload- load file from an UBIFS filesystem
ubifsls - list files in a directory
```



```
ubifsmount- mount UBIFS volume
ubifsumount- unmount UBIFS volume
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor, compiler and linker version
OMAP Logic #
```

3.3.3 U-Boot Environment

The U-Boot environment consists of all variables in the shell. The shell variables can contain numbers, strings, or a sequence of commands. Shell variables can be defined in the U-Boot source code, at the shell prompt, or loaded from NAND flash at boot time.

To display the U-Boot environment, use the *printenv* command. **NOTE:** Your output may be different than that shown below.

```
OMAP Logic # printenv
arch=arm
autoboot=mmc dev ${mmcdev}; if mmc rescan; then if run loadbootscript;
then run bootscript; else run defaultboot;fi; else run defaultboot; fi
autoload=no
baudrate=115200
board=omap3som
board name=omap3som
boot fdt=try
bootcmd=run autoboot
bootdelay=2
bootm size=0x10000000
bootscript=echo 'Running bootscript from mmc ...'; source ${loadaddr}
common bootargs=run setconsole; setenv bootargs ${bootargs}
console=${console} ${mtdparts} ${optargs}; run vrfb arg
consoledevice=ttyO0
cpu=armv7
defaultboot=run mmcramboot
dump bootargs=echo 'Bootargs: '; echo $bootargs
fdt addr r=0x88000000
fdtaddr=0x88000000
kernel addr r=0x82000000
loadaddr=0x82000000
loadbootscript=load mmc ${mmcdev} ${loadaddr} boot.scr
loadfdt=mmc rescan; load mmc ${mmcdev} ${fdtaddr} ${fdtimage}
loadimage=mmc rescan; load mmc ${mmcdev} ${loadaddr} ${bootfile}
loadramdisk=mmc rescan; load mmc ${mmcdev} ${rdaddr} ${ramdiskimage}
mmcargs=setenv bootargs root=${mmcroot} rootfstype=${mmcrootfstype}
mmcboot=setenv bootfile uImage; run mmcbootcommon; bootm ${loadaddr} -
${fdtaddr}
mmcbootcommon=echo Booting with DT from mmc${mmcdev} ...; run mmcargs;
run common bootargs; run dump bootargs; run loadimage; run loadfdt;
mmcbootz=setenv bootfile zImage; run mmcbootcommon; bootz ${loadaddr} -
${fdtaddr}
mmcdev=0
mmcramboot=setenv bootfile uImage; run mmcrambootcommon; bootm
${loadaddr} ${rdaddr} ${fdtimage}
```



```
mmcrambootcommon=echo 'Booting kernel from MMC w/ramdisk...'; run
ramargs; run common bootargs; run dump bootargs; run loadimage; run
loadfdt; run loadramdisk
mmcrambootz=setenv bootfile zImage; run mmcrambootcommon; bootz
${loadaddr} ${rdaddr} ${fdtimage}
mmcroot=/dev/mmcblk0p2 rw
mmcrootfstype=ext4 rootwait
mtdids=nand0=omap2-nand.0
mtdparts=mtdparts=omap2-nand.0:512k(MLO),1792k(U-Boot ),128k(spl-
os),128k(U-Boot -env),6m(kernel),-(fs)
nandargs=setenv bootargs root=${nandroot} rootfstype=${nandrootfstype}
nandboot=run nandbootcommon; bootm ${loadaddr} - ${fdtaddr}
nandbootcommon=echo 'Booting kernel from NAND...';run nandargs;run
common bootargs;run dump bootargs;nand read ${loadaddr} kernel;nand
read ${fdtaddr} spl-os;
nandbootz=run nandbootcommon; bootz ${loadaddr} - ${fdtaddr}
nandroot=ubi0:rootfs rw ubi.mtd=fs noinitrd
nandrootfstype=ubifs rootwait
nfsargs=setenv serverip ${tftpserver}; setenv bootargs root=/dev/nfs
nfsroot=${nfsrootpath}
ip=${ipaddr}:${tftpserver}:${gatewayip}:${netmask}::eth0:off
nfsrootpath=/opt/nfs-exports/omap
optargs=ignore loglevel early printk no console suspend
preboot=setenv preboot;saveenv;
pxefile addr r=0x80100000
ramargs=setenv bootargs root=/dev/ram rw ramdisk size=${ramdisksize}
ramdisk addr r=0x88080000
ramdiskimage=rootfs.ext2.gz.uboot
ramdisksize=64000
rdaddr=0x88080000
rotation=0
scriptaddr=0x80000000
setconsole=setenv console ${consoledevice},${baudrate}n8
soc=omap3
tftpboot=echo 'Booting kernel/ramdisk rootfs from tftp...'; run
ramargs; run common bootargs; run dump bootargs; tftpboot ${loadaddr}
${zimage}; tftpboot ${rdaddr} ${ramdiskimage}; bootm ${loadaddr}
${rdaddr}
tftpbootz=echo 'Booting kernel NFS rootfs...'; dhcp;run nfsargs;run
common bootargs;run dump bootargs;tftpboot $loadaddr zImage;bootz
$loadaddr
vendor=logicpd
vrfb arg=if itest ${rotation} -ne 0; then setenv bootargs ${bootargs}
omapfb.vrfb=y omapfb.rotate=${rotation}; fi

Environment size: 3421/131068 bytes
OMAP Logic #
```

To save the environment to NAND so that the variables are available on the next boot, use the *saveenv* command.

```
OMAP Logic # saveenv
Saving Environment to NAND...
```



```
Erasing NAND...
Erasing at 0x260000 -- 100% complete.
Writing to NAND... OK
OMAP Logic #
```

To reset the environment to the default, use the *env default* command.

```
OMAP Logic # env default -f -a
```

NOTE: When booting a different version of U-Boot, consider resetting the environment to the default and then saving the default environment. Different versions of U-Boot and different Operating Systems often have different environment requirements, making different U-Boot environments incompatible. The U-Boot environment, whether booting from SD or NAND, is always loaded from NAND flash. So having the wrong environment in NAND flash can make SD booting from a different version of U-Boot fail.

3.3.4 Shell Variables

Shell variables can be used as a means to configure inherent U-Boot operation, such as the *defaultecc* variable, or to change NAND flash.

To set a variable at the shell, use the *setenv* command.

```
OMAP Logic # setenv foo 5
OMAP Logic # echo $foo
5
```

In the above example, the *echo* command and the "$" character are used to display the content of a variable, where $ indicates that the content of the specified variable *foo* should be used, rather than the word *foo* itself.

3.3.5 Updating Kernel Command Line

To add a parameter to the kernel command line you will need to update the $optargs variable within U-Boot. To do this, you can reboot the system, pause U-Boot, and add the above value to the *optargs* environment variable following the steps below.

The <new_kernel_option> is being added to the kernel command line in this example below.

```
OMAP Logic # echo $optargs
ignore loglevel early printk no console suspend
OMAP Logic # setenv optargs ${optargs} new_kernel_option
OMAP Logic # echo $optargs
ignore loglevel early printk no console suspend new kernel option
```

3.3.6 View the device tree in U-Boot

The commands below can be used to view the device tree while in U-Boot .


```
run loadfdt
fdt addr $fdtaddr
fdt print /display
```

3.4 Falcon Mode

Falcon mode allows the system to bypass U-Boot and let the MLO load the kernel directly. It is a feature of U-Boot's SPL.

3.4.1 Common Falcon Mode Setup

This section describes the steps to rebuild the kernel in preparation for use by MLO in Falcon Mode.

1. Go to the Buildroot → Kernel menu.

```
$ cd ~/logic/buildroot
$ make menuconfig
```

2. Change the Kernel Options as follows:

   a. Kernel from zImage to uImage

```
[*] Linux Kernel
      Kernel version (Custom version)  --->
(4.9.89) Kernel version
(lpd-linux-4.9.y.patch) Custom kernel patches
      Kernel configuration (Using an in-tree defconfig file)  --->
(omap3logic) Defconfig name
()    Additional configuration fragment files
      Kernel binary format (uImage)  --->
      Kernel compression format (lzo compression)  --->
(0x82000000) load address (for 3.7+ multi-platform image)
[*]   Build a Device Tree Blob (DTB)
        Device tree source (Use a device tree present in the kernel)  --->
(logicpd-som-lv-37xx-devkit logicpd-torpedo-37xx-devkit) Device Tree Source file names
[ ]   Install kernel image to /boot in target
      Linux Kernel Extensions  --->
      Linux Kernel Tools  --->
```


b. Load Address 0x82000000

```
[*] Linux Kernel
      Kernel version (Custom version)  --->
(4.9.89) Kernel version
(lpd-linux-4.9.y.patch) Custom kernel patches
      Kernel configuration (Using an in-tree defconfig file)  --->
(omap3logic) Defconfig name
()    Additional configuration fragment files
      Kernel binary format (uImage)  --->
      Kernel compression format (lzo compression)  --->
(0x82000000) load address (for 3.7+ multi-platform image)
[*]   Build a Device Tree Blob (DTB)
        Device tree source (Use a device tree present in the kernel)  --->
(logicpd-som-lv-37xx-devkit logicpd-torpedo-37xx-devkit) Device Tree Source file names
[ ]   Install kernel image to /boot in target
      Linux Kernel Extensions  --->
      Linux Kernel Tools  --->
```

3. Build the kernel with the updated parameters:

```
$ make linux-rebuild
$ make
```

3.4.2 Falcon Mode NAND

This section describes how to quickly boot a UBI filesystem from NAND by bypassing the U-Boot environment and having MLO boot the kernel directly. It requires a little reconfiguration of the kernel and some U-boot parameter changes, but it decreases the load time and users who want a non-volatile system with faster boot times.

4. Complete section 3.4.1
5. Format the SD card per Section 3.2.1
6. Remove and reinsert the SD Card so that Ubuntu will mount all partitions
7. Copy the necessary files to the SD card:.

```
$ cd ~/logic/buildroot
$ cp output/images/MLO /media/logic/boot/
$ cp output/images/u-boot.img /media/logic/boot/

For DM37x Torpedo and Torpedo + Wireless:
$ cp output/images/logicpd-torpedo-37xx-devkit.dtb /media/logic/boot/
For DM37x SOM-LV:
$ cp output/images/logicpd-som-lv-37xx-devkit.dtb /media/logic/boot/

$ cp output/images/uImage /media/logic/boot/
$ cp output/images/rootfs.ubifs /media/logic/boot
$ sync
```

8. Remove the SD card and boot the board from SD card.
9. Erase the flash chip and reset the environmental variables.


```
OMAP Logic # nand erase.chip
OMAP Logic # env default -a
```

10. Program UBIFS image to NAND

```
OMAP Logic # fatload mmc 0 $loadaddr rootfs.ubifs
OMAP Logic # ubi part fs
OMAP Logic # ubi create rootfs
OMAP Logic # ubi write $loadaddr rootfs $filesize
```

11. Program the Linux kernel into the corresponding NAND partition. For Falcon mode, this must be the uImage and cannot be the default zImage.

```
OMAP Logic # fatload mmc 0 $loadaddr uImage
OMAP Logic # nand write $loadaddr kernel $filesize
```

12. Load the DTB and Bootargs into RAM. For this example, the 'optargs' will be set to quiet as this will help speed up the boot process. This can be combined with other parameters if desired.

```
For DM37x Torpedo and Torpedo + Wireless:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-torpedo-37xx-devkit.dtb
For DM37x SOM-LV:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-som-lv-37xx-devkit.dtb

OMAP Logic # setenv optargs quiet
OMAP Logic # run nandargs
OMAP Logic # run common_bootargs
```

The SPL export passes the bootargs kernel parameter as well as the device tree to the kernel upon startup.

```
 OMAP Logic # spl export fdt $loadaddr - $fdtaddr
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.9.89
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4844704 Bytes = 4.6 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Kernel Image ... OK
   Loading Device Tree to 8dede000, end 8def2098 ... OK
subcommand not supported
subcommand not supported
   Loading Device Tree to 8dec6000, end 8dedd098 ... OK
Argument image is now in RAM: 0x8dec6000
```


Note the new address for this example is **0x8dec6000**. This becomes the address stored to spl-os partition.

```
OMAP Logic # nand write 0x8dec6000 spl-os
NAND write: device 0 offset 0x240000, size 0x20000
```

131072 bytes written: OK

13. Program U-Boot into the corresponding NAND partition.

```
OMAP Logic # fatload mmc 0 $loadaddr u-boot.img
OMAP Logic # nand write $loadaddr u-boot $filesize
```

14. Program the MLO with HW ECC last into the corresponding NAND partition.

Note: MLO must be done after all the other changes are written to NAND, because the ECC for MLO is different than the ECC for the rest of the NAND partitions.

```
OMAP Logic # nandecc hw
OMAP Logic # fatload mmc 0 $loadaddr MLO
OMAP Logic # nand write $loadaddr MLO $filesize
```

15. Remove the SD Card from the target system.
16. The system must be reset or power cycled to reset the ECC configuration.

```
OMAP Logic # reset
```

The system should now successfully boot from NAND using the generated UBI rootfs.

```
U-Boot SPL 2018.03 (Apr 10 2018 - 08:25:15 -0500)
Trying to boot from NAND
[    0.000000] Booting Linux on physical CPU 0x0
…
```

3.4.3 Falcon Mode SD

This section describes how to quickly boot an EXT filesystem on an SD card by bypassing the U-Boot environment and having MLO boot the kernel directly. It requires a little reconfiguration of the kernel and some U-boot parameter changes, but it decreases the load time and users who want a non-volatile system with faster boot times.

1. Complete section 3.4.1
2. Format the SD card per Section 3.2.1
3. Remove and reinsert the SD Card so that Ubuntu will mount all partitions
4. Copy the necessary files to the SD card:

```
$ cd ~/logic/buildroot
$ cp output/images/MLO /media/logic/boot/
$ cp output/images/u-boot.img /media/logic/boot/
```



```
For DM37x Torpedo and Torpedo + Wireless:
$ cp output/images/logicpd-torpedo-37xx-devkit.dtb /media/logic/boot/
For DM37x SOM-LV:
$ cp output/images/logicpd-som-lv-37xx-devkit.dtb /media/logic/boot/

$ cp output/images/uImage /media/logic/boot/
$ sudo tar xvf output/images/rootfs.tar -C /media/logic/rootfs/
$ sync
```

5. Remove the SD card and boot the board from SD card.
6. Erase the flash chip and reset the environmental variables.

```
OMAP Logic # nand erase.chip
OMAP Logic # env default -a
```

7. Load the Linux kernel into RAM. For Falcon mode, this must be the uImage and cannot be the default zImage.

```
OMAP Logic # fatload mmc 0 $loadaddr uImage
```

8. Load the DTB and Bootargs into RAM. For this example, the 'optargs' will be set to quiet as this will help speed up the boot process. This can be combined with other parameters if desired.

```
For DM37x Torpedo and Torpedo + Wireless:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-torpedo-37xx-devkit.dtb
For DM37x SOM-LV:
OMAP Logic # fatload mmc 0 $fdtaddr logicpd-som-lv-37xx-devkit.dtb

OMAP Logic # setenv mmcroot /dev/mmcblk1p2 rw
OMAP Logic # setenv optargs quiet
OMAP Logic # run mmcargs
OMAP Logic # run common_bootargs
```

The SPL export passes the bootargs kernel parameter as well as the device tree to the kernel upon startup.

```
 OMAP Logic # spl export fdt $loadaddr - $fdtaddr
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.9.89
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4844704 Bytes = 4.6 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Kernel Image ... OK
   Loading Device Tree to 8dede000, end 8def2098 ... OK
```



```
subcommand not supported
subcommand not supported
   Loading Device Tree to 8dec6000, end 8dedd098 ... OK
Argument image is now in RAM: 0x8dec6000
```

Note the new address for this example is **0x8dec6000** for the Torpedo SOMs and **0x8dec8000** for the SOM-LV. The actual address may vary.

```
For DM37x Torpedo and Torpedo + Wireless:
OMAP Logic # fatwrite mmc 0:1 0x8dec6000 args 0x20000
For DM37x SOM-LV:
OMAP Logic # fatwrite mmc 0:1 0x8dec8000 args 0x20000
```

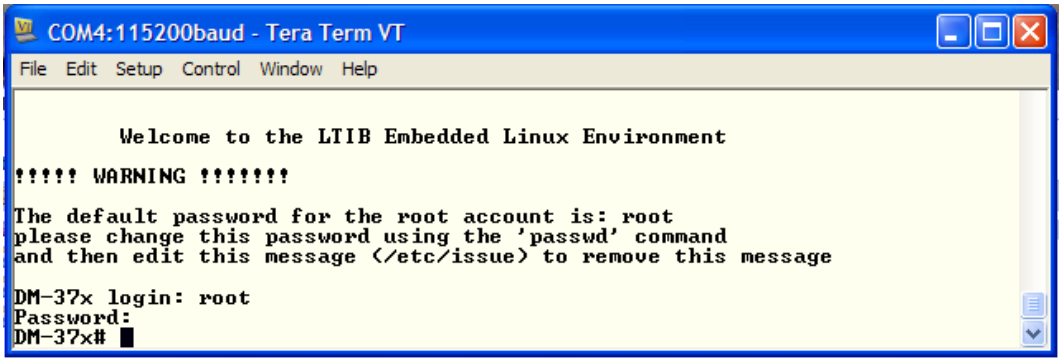
9. Reset the board, and the device should boot fromMLO directly to the Kernel, bypassing U-Boot.

```
OMAP Logic # reset
```


4 Kernel

This section describes commonly used Linux shell commands. In no way is this section comprehensive; Linux is always changing, and many packages can be added or removed. Visit the Linux Kernel Archives website[7] and the open source community for information on additional commands.

To begin, log in to the kernel in Tera Term to get to the prompt; the password is set to *root*.



For more information on the Linux kernel, look inside the *rpm/BUILD/linux/Documentation* sub-directory.

Note: On Boot, a message that appears to be an error may appear. It is perfectly harmless and U-Boot should load.

```
U-Boot SPL 2018.03 (Mar 23 2018 - 14:03:53 -0500)
Trying to boot from MMC1
spl load image fat os: error reading image args, err - -2


U-Boot 2018.03 (Mar 23 2018 - 14:03:53 -0500)
```

4.1 vi Editor

Much of Linux is configured with various text files. The vi Editor is a simple, light-weight editor commonly used to edit text files; it should be used to edit the text files as described in the following sections. It is beyond the scope of this document to describe how to use the vi Editor; however, there are many documents available on the Internet that provide instructions.

To start the vi Editor, simply enter the *vi <file>* command at the Linux prompt, where *<file>* is the name of the file you wish to edit.

```
# vi /etc/network/interfaces
```

[7] http://www.kernel.org/

4.2 Retrieve BSP Version

To obtain the BSP version, log in to Linux on the SOM and enter the command below.

```
# cat /proc/version
# cat /proc/version
Linux version 4.9.89 (logic@logic-OptiPlex-990) (gcc version 7.3.0
(Buildroot 2018.02) ) #1 Fri Mar 23 14:09:03 CDT 2018
```

4.3 Display Linux System Information

The Linux commands below provide additional system information.

Display the kernel version and build date information

```
# uname -a
Linux buildroot 4.9.89 #1 Fri Mar 23 14:09:03 CDT 2018 armv7l GNU/Linux
```

Provided cmdline passed in by U-Boot to the kernel

```
# cat /proc/cmdline
root=/dev/mmcblk1p2 rw rootfstype=ext4 rootwait console=ttyO0,115200n8
mtdparts=omap2-nand.0:512k(MLO),1792k(u-boot),128k(spl-os),128k(u-boot-
env),6m(kernel),-(fs) ignore loglevel early printk no console suspend
```

Memory space

```
# cat /proc/meminfo
MemTotal:         249172 kB
MemFree:          212284 kB
MemAvailable:     228276 kB
Buffers:            9012 kB
Cached:             9780 kB
SwapCached:            0 kB
Active:            12380 kB
Inactive:           7328 kB
Active(anon):        972 kB
Inactive(anon):      120 kB
Active(file):      11408 kB
Inactive(file):     7208 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         249172 kB
LowFree:          212284 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                32 kB
Writeback:             0 kB
```



```
AnonPages:           936 kB
Mapped:             2784 kB
Shmem:               180 kB
Slab:              11712 kB
SReclaimable:       5580 kB
SUnreclaim:         6132 kB
KernelStack:         672 kB
PageTables:          112 kB
NFS Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      124584 kB
Committed AS:       2716 kB
VmallocTotal:     778240 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
CmaTotal:          16384 kB
CmaFree:           15600 kB
```

Partition information on NAND and NOR flash (does not display SD card or USB memory stick partitions)

```
# cat /proc/mtd
ev:    size   erasesize  name
mtd0: 00080000 00020000 "MLO"
mtd1: 001c0000 00020000 "U-Boot "
mtd2: 00020000 00020000 "spl-os"
mtd3: 00020000 00020000 "U-Boot -env"
mtd4: 00600000 00020000 "kernel"
mtd5: 1f780000 00020000 "fs"
```

Partition size availability

```
# df -k
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root              3488352     99284   3208540   3% /
devtmpfs                115880         0    115880   0% /dev
tmpfs                   124584         0    124584   0% /dev/shm
tmpfs                   124584        56    124528   0% /tmp
tmpfs                   124584       124    124460   0% /run Mounted
Partitions
```

```
# mount
/dev/mmcblk1p2 on / type ext4 (rw,relatime,data=ordered)
devtmpfs on /dev type devtmpfs
(rw,relatime,size=115880k,nr inodes=28970,mode=755)
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts
(rw,relatime,gid=5,mode=620,ptmxmode=666)
```



```
tmpfs on /dev/shm type tmpfs (rw,relatime,mode=777)
tmpfs on /tmp type tmpfs (rw,relatime)
tmpfs on /run type tmpfs (rw,nosuid,nodev,relatime,mode=755)
sysfs on /sys type sysfs (rw,relatime) #
```

4.4 Wired Networking

4.4.1 Assign Development Kit IP Address

The pre-built Linux images do not automatically configure a network interface. However, configuring a network interface by hand is rather straight forward and there are several options from which to choose. In the following sections, we will set the kernel command line argument *ip* shown below.

ip=<client_ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>

The kernel command line is passed into the kernel from U-Boot via the U-Boot *optargs* environment variable. See Section 3.3.5 for information about how to set the kernel command line from U-Boot.

NOTE: The kernel will use the default value of any argument omitted.

4.4.1.1 Use *ifconfig* Command to Report Ethernet Status

The *ifconfig* command with no arguments can be used to report the status of all Ethernet interfaces.

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:08:EE:06:3E:65
          inet addr:10.0.5.39  Bcast:10.0.7.255  Mask:255.255.252.0
          inet6 addr: fe80::208:eeff:fe06:3e65/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:158 errors:0 dropped:4 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16852 (16.4 KiB)  TX bytes:1192 (1.1 KiB)
          Interrupt:241

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

4.4.1.2 Add DHCP from Kernel Command Line

In U-Boot, add **ip=::::::dhcp** to the kernel command line. To do this, you can reboot the system, pause U-Boot, and add the above value to the *optargs* environment variable.

You can then execute the boot command or save the environment for future boot from power up. See Section 3.3.4 for more information on setting U-Boot environment variables.

```
OMAP Logic # echo $optargs
ignore_loglevel early_printk no_console_suspend
OMAP Logic # setenv optargs ${optargs} ip=::::::dhcp
OMAP Logic # echo $optargs
ignore_loglevel early_printk no_console_suspend ip=::::::dhcp
```

When booting, the kernel will wait to acquire an IP address. Be sure to have the Ethernet cable connected when booting.

NOTE: If you want this configuration to be used on a future power cycle, be sure to use the U-Boot *saveenv* command. See Section 3.3.3 for more information.

'Starting network: ip: RTNETLINK answers: File exists FAIL' error seen can be eliminated by removing the /etc/init.d/S40network file.

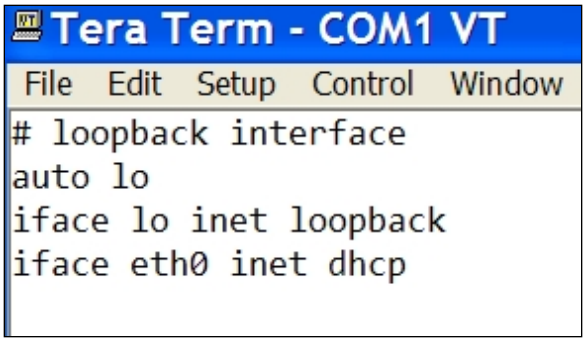
4.4.1.3 Add Static IP Address from Kernel Command Line

To add a static IP address to the kernel command line, follow the example in Section 4.4.1.2, but add **`ip=ip-address::gateway-address:netmask:::`** to the kernel command line in place of **`ip=::::::dhcp`**. Replace *ip-address*, *gateway-address*, and *netmask* with your unique values.

4.4.1.4 Using DHCP

Edit the */etc/network/interfaces* file to configure *eth0* for DHCP operation by including the following line:

```
iface eth0 inet dhcp
```



Next, use the *ifup* command to bring up the interface.

```
# ifup eth0
```


4.4.1.5 Using Static IP

Edit the */etc/network/interfaces* file to configure *eth0* with a static IP address by adding the following lines, where <> indicates where your unique user values should be inserted.

```
iface eth0 inet static
address <>
netmask <>
gateway <>
```

Configure the DNS server(s)

```
# mv /etc/resolv.conf /etc/resolv.conf.old
# vi /etc/resolv.conf
```

Edit the */etc/resolv.conf* file to assign the DNS server by adding the following lines where <> indicates where your unique user values should be inserted.

```
# nameserver <DNS server #1>
# nameserver <DNS server #2>
nameserver 8.8.8.8
```

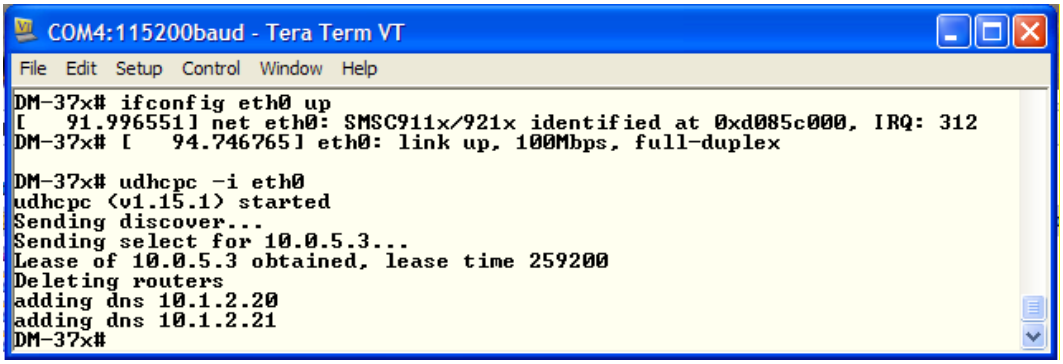
Next, use the *ifup* command to bring up the interface.

```
# ifup eth0
```

4.4.1.6 Use *ifconfig* Command with DHCP

Bring up the network and obtain an IP address from the DHCP server using the commands below.

```
# ifconfig eth0 up
# udhcpc -i eth0
```

4.4.2 Set Speed, Duplex, and Auto-Negotiate

By default, the wired Ethernet supports auto-negotiation. Should you need to manually force the speed or duplex settings of the interface, the demo image includes the *ethtool* command.

Use *autoneg off* when moving from 100 Mbs to 10 Mbs or from full duplex to half duplex. Use *autoneg on* when moving from 10 Mbs to 100 Mbs or from half duplex to full duplex. Your network equipment may respond differently, so this may vary in your situation. Examples are included below.

```
# ethtool -s eth0 autoneg off speed 10 duplex half
# ethtool -s eth0 autoneg on speed 100 duplex full
```

Use the *following* command to view the current Ethernet settings:

```
# ethtool eth0 | less
```

4.4.3 Cable Detection

Two determine the state of the physical state of the interface, read the contents of /sys/class/net/eth0/carrier

0: physical link is down
1: physical link is up

```
# cat /sys/class/net/eth0/carrier
1
```

4.5 Linux Processes

Linux manages many concurrent processes. The following tools can help users manage those processes.

4.5.1 *ps* Command

The *ps* command is used to display the Linux processes. Since there tends to be a large number of processes running, the *ps* command accepts several arguments to help sort

through the process list. Use *ps -help* at the Linux prompt to see all the arguments available to *ps*.

The following example shows a list of all the processes running. **NOTE:** This list may differ from the list on your DM3730 Development Kit.

```
# ps
PID   USER     COMMAND
    1 root     init
    2 root     [kthreadd]
    3 root     [ksoftirqd/0]
    5 root     [kworker/0:0H]
    6 root     [kworker/u2:0]
    7 root     [rcu sched]
    8 root     [rcu bh]
   …
  251 root     udhcpc -R -n -p /var/run/udhcpc.eth0.pid -i eth0
  259 root     [kworker/u2:2]
  269 root     ps
```

4.5.2 *kill* Command

The *kill* command is used to stop a process. Processes are referenced by their process ID (PID). The PID of a process can be found using the *ps* command. Below, we will use the *kill* command with the *-9* argument to force the process to stop immediately without question.

```
# kill -9 269
```

4.6 Video Display

Configuring the display type is done in U-Boot. Once the *$display* variable is set, the boot sequence will pass that display type to the kernel in the *$bootargs* environment variable. Please refer to Section 3.2 on U-Boot for further information on configuring the display.

To set the display in U-Boot to display 28, pause U-Boot and execute the following commands:

```
OMAP Logic #  setenv display 28
OMAP Logic #  saveenv
Saving Environment to NAND... Erasing NAND...
Erasing at 0x260000 -- 100% complete.
Writing to NAND... OK
OK
#
```


4.6.1 Framebuffer Tests

4.6.1.1 Framebuffer Test Pattern

To perform a video test on an LCD panel display #28, connect the LCD to the DM3730 Development Kit and enter the command below.

```
# fb-test
```

Your LCD panel will display a pattern similar to that shown in Figure 4.1 below.

Figure 4.1: LCD Draw Test Display Pattern

4.6.1.2 Framebuffer Test Performance

Fb-test-rect will draw a series of rectangles on the screen.

```
# fb-test-perf /dev/fb0 fb-log.log
perf 1.1.0 (rosetta)
sequential horiz singlepixel read: 24414720 pix, 3480256 us, 7015208
pix/s
sequential horiz singlepixel write: 810255360 pix, 4703430 us,
172269037 pix/s
sequential vert singlepixel read: 31726080 pix, 4992736 us, 6354447
pix/s
sequential vert singlepixel write: 235008000 pix, 4960114 us, 47379556
pix/s
sequential line read: 364915200 pix, 4941253 us, 73850741 pix/s
sequential line write: 2487690240 pix, 4838013 us, 514196683 pix/s
nonsequential singlepixel write: 32640000 pix, 4988099 us, 6543575
pix/s
nonsequential_singlepixel_read: 15797760 pix, 4980591 us, 3171864 pix/s
```

4.6.1.3 Framebuffer Test Rectangles

Fb-test-rect will draw a series of rectangles on the screen. To exit, press Control-C

```
# fb-test-rect
rect 1.1.0 (rosetta)
```


4.6.2 Backlight

The backlight can be controlled using the commands below. The range can be anywhere from 0 (off) to 10 (full brightness).

```
# echo 0 > /sys/class/backlight/backlight/brightness
# echo 10 > /sys/class/backlight/backlight/brightness
```

4.6.3 Console Blanking

The console backlight blanks after ten minutes. Below is information about how to prevent console backlight blanking or to recover from the console backlight blanking.

To prevent the console backlight from blanking after ten minutes, add *consoleblank=0* to the kernel command line.

```
OMAP Logic # setenv optargs $optargs consoleblank=0
```

To bring back the backlight after it has turned off due to console backlight blanking, use the following command in the Linux debug console.

```
# echo -e '\033[13]' > /dev/tty0
```

The following command sets the screen blank timeout to <n> minutes.

```
# echo -e '\033[9;<n>]' > /dev/tty0

Example: This commend changes the screen blank timeout to 5 minutes.

# echo -e '\033[9;5]' > /dev/tty0
```

The following command cancels screen blanking.

```
# echo -e '\033[9]' > /dev/tty0
```

4.6.4 Display Message

The *echo* command can be used to display text messages on the LCD. In this example, "*Hello World*" text will be seen on the LCD assigned to *tty0*.

Note: If some Applications have taken over the Framebuffer, accessing /dev/tty0 may not function. The fb-test apps are an example of this. Users are cautioned not to run the fb-test apps if data is to be out to tty0.

```
# echo "Hello World" > /dev/tty0
```


4.7 SD/MMC Interface

The demo image includes support for SD/MMC memory cards.

The SD/MMC interface is found at */dev/mmcblk1* and the first partition is located at */dev/mmcblk1p1*. Use the commands below to mount the SD/MMC card.

NOTE: The location where the SD card is mounted is arbitrary. The example below mounts the SD card to the location */mnt/sdcard*.

```
# mkdir /mnt/sdcard
# mount -t vfat /dev/mmcblk1p1 /mnt/sdcard
```

4.8 Touch Screen

The demo image includes support for the DM3730 Development Kit's touch screen via the special file */dev/input/event0*.

4.8.1 Configuration

To configure the touch screen, use the commands below.

```
# export TSLIB_TSDEVICE=/dev/input/by-path/platform-48060000.i2c-event
# export TSLIB_CALIBFILE=/etc/pointercal
# export TSLIB_CONSOLEDEVICE=none
# export TSLIB_FBDEVICE=/dev/fb0
```

4.8.2 Calibration

Verify touches respond appropriately and then enter the command below.

```
# ts_calibrate
```

4.8.3 Test Touch

Running *ts_test* after calibrating the screen will allow this application to work as expected. The test application allows uses to use the draw and drag tests.

```
# ts_test
```

4.9 Built-in Flash Storage via MTD

To determine where flash partitions are, enter the command below and look for something similar to "nor-flash" or "NAND fs" in the output. For mounting UBI filesystem and/or booting from flash, see section *3.2.3 Boot UBI RootFS from NAND* of this guide.

```
```



```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00020000 "MLO"
mtd1: 001c0000 00020000 "U-Boot "
mtd2: 00020000 00020000 "spl-os"
mtd3: 00020000 00020000 "U-Boot -env"
mtd4: 00600000 00020000 "kernel"
mtd5: 1f780000 00020000 "fs"
```

NOTE: The above partition map may differ depending on the Linux version, device tree, U-Boot, and the hardware installed.

4.10 USB Controller

Use the commands below to display information on all USB devices connected to either the USB Host or USB OTG controllers.

```
# lsusb
Bus 003 Device 001: ID 1d6b:0002
Bus 001 Device 001: ID 1d6b:0002
Bus 002 Device 001: ID 1d6b:0001
```

4.11 USB Host Controller

The DM3730 Development Kit's standard USB Type A connector (flat, rectangular) is attached to the host controller. The device driver for the ISP1763 controller driver is not currently available in this Linux release. Contact Logic PD for other USB Host support options to access using various USB hubs, flash drives, keyboards, and mouse devices from an external USB Host controller.

4.12 Processor OTG Controller

The DM3730 and AM3703 processors include an Inventra high-speed, dual-role controller commonly referred to as MUSB. This peripheral is attached to the type mini-A connector on the DM3730 Development Kit baseboard. Do support dual-role a USB gadget must be loaded. See the next section for details how to load a USB gadget driver.

4.12.1 Use MUSB in Host Mode

When attempting to use the MUSB in host mode, it is important that to use a proper OTG cable with a mini-A connector that utilizes the ID pin when connecting to the DM3730 Development Kit. A mini-B connector looks very similar to a mini-A connector and will mate with a mini-A; however, the mini-B connectors will not properly configure the ID pin that forces the OTG controller into host mode. One acceptable mini-A cable is the 2 meter USB OTG cable from Lindy[8] (PN 31634).

Once the proper cable is connected, load and boot the standard demo image. You can then use the interface normally.

[8] http://www.lindy-usa.com/

Although it's counter-intuitive, to enable the USB Host a gadget driver must be loaded first. By default, g_zero can provide the basic host services.

```
# modprobe g_zero
[  335.354064] zero gadget: Gadget Zero, version: Cinco de Mayo 2008
[  335.360687] zero gadget: zero ready

(connect USB Device)
```

USB gadget driver can be included in to the kernel or as a module using the menuconfig option:

```
-> Device Drivers
    -> USB support (USB_SUPPORT [=y])
        -> USB Gadget Support (USB_GADGET [=y])
            -> USB Gadget Drivers (<choice> [=y])
```

4.12.2 Device Mode Gadgets

Only one gadget can be loaded at one time. Therefore, if you want to use any of the device gadgets below you must unload any prior gadgets. For example, if you tested the g_zero gadget in the section above, you must unload the g_zero gadget before you can install any other gadget. It unload the g_zero gadget use the following command:

```
# rmmod g_zero
```

4.12.2.1 Ethernet Gadget

To get started, build the kernel with the Ethernet gadget compiled as a module.

To use the device as an Ethernet-over-USB gadget, follow the steps below.

1. Load the Ethernet Gadget

```
$ modprobe g_ether
```

1. To find the device name enter 'udevadm monitor –-udev' on the Host PC command prompt before connecting the cable.

```
$ udevadm monitor --udev
UDEV  [5707038.709412] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7 (usb)
UDEV  [5707038.710841] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-1.7:1.0 (usb)
UDEV  [5707038.714214] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-1.7:1.1 (usb)
```



```
UDEV  [5707038.737482] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-
1.7:1.0/net/enp0s29u1u7 (net)
UDEV  [5707038.738099] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-
1.7:1.0/net/usb0/queues/rx-0 (queues)
UDEV  [5707038.738121] add
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-
1.7:1.0/net/usb0/queues/tx-0 (queues)
UDEV  [5707038.739783] move
/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.7/2-
1.7:1.0/net/enp0s29u1u7 (net)

[1315282.362580] cdc ether 1-1:1.0 enp0s20f0u1: renamed from usb0
```

2. Connect the DM3730 Development Kit USB-OTG port to a Linux host PC using the included USB mini-B to Standard-A cable.
3. Configure the new network connection on the Linux host PC.

```
#
# ifconfig <interface> <IP_address>
$ sudo ifconfig enp0s29u1u7 192.168.1.2
```

4. Configure the network connection on the DM3730 Development Kit.

```
# ifconfig usb0 <IP_address>
$ ifconfig usb0 192.168.1.1
```

5. Ping Linux host PC to verify connection

```
$ ping 192.168.1.2
```

4.12.2.2 Mass Storage Gadget

To get started, build the kernel with the Mass Storage Gadget compiled as a module. To use the device as a file-backed storage gadget, follow the steps below. This example assumes that you have an FAT filesystem on an SD/MMC card attached to */dev/mmcblk1*.

NOTE: The card must not already be mounted. If the SD/MMC card is mounted, use the command below to unmount it before proceeding.

```
# umount /mnt/mmcblk1p1
```

1. Load the kernel module.

```
#  rmmod g_ether
#  modprobe g mass storage file=/dev/mmcblk1
```

2. Connect the device to a Linux host PC.

3. Do whatever is necessary on your host PC to mount the newly inserted device. Most modern Linux distributions such as Fedora and Ubuntu as well as systems running will automatically detect the device and treat it like a USB flash drive.
4. Transfer some files over to the device.
5. Disconnect the device from the host PC.
6. Mount the SD/MMC card and verify that the files were transferred.

```
# mkdir /mnt/mmcblk1p1
# mount /dev/mmcblk1p1 /mnt/mmcblk1p1
# ls /mnt/mmcblk1p1
```

4.13 UART

For all versions of the Torpedo Launcher Baseboard included with the DM3730 Torpedo Development Kit, RS-232 transceivers and connector cables are available for *(/dev/ttyO2)* UARTB and *(/dev/ttyO1)* UARTC. To use UARTC the jumper connecting J31.28 to J31.30 must be removed so that U25 is not powered down.

For the SDK2 Baseboard included with the DM3730 SOM-LV Development Kit, custom UART transceivers can be connected to the High Density Breakout Board. By default, the serial ports are enabled at 9600 baud and are available on */dev/ttyO1* (UARTC) and */dev/ttyO2* (UARTB). */dev/ttyO0* is used for the console at 115200 baud.

1. To change the port speed, use the *stty* command. For example, to change UARTB to 115200, enter the command below.

```
# stty 115200 < /dev/ttyO2
```

2. Attach the UART connector to J25/UARTB and use the *echo* command to send test output.

```
# echo "This is a test message" > /dev/ttyO2
```

3. Use the *cat* command to receive test data. **NOTE:** The *cat* command does not output the data until it receives a carriage return.

```
# cat /dev/ttyO2
```

4.14 Real Time Clock

The Real Time Clock (RTC) in the DM3730 and AM3703 processor is not backed up by battery, but the RTC in the PMIC is. The *hwclock* command writes between the two RTCs. The Torpedo Launcher 2 Baseboard and later versions have a backup battery; however, by default the DM37x Linux BSP does not enable charging.

For more information, see the *TPS65950 OMAP PMIC Technical Reference Manual*.[9]

[9] http://www.ti.com/product/tps65950#technicaldocuments

The commands below set the clock on the DM3730 and AM3703 processor RTC and then use *hwclock* to write it into the PMIC. If BACKUP_BATT is powered, you can shut off MAIN_BATTERY and the RTC will continue to run in the PMIC. Upon reboot, the system automatically reads the value from the PMIC.

NOTE: When booting with the rootfs on SD (EXT3) or in NAND (UBI), the system will not respond to the S5 power switch or the S1 reset switch on the DM37x development board. With all boot options, the system will resume following the press of the S2 switch.

```
# date +%s -s @1476887342
1476887342
# date
Wed Oct 19 14:29:05 UTC 2016
# hwclock -w
# hwclock
Wed Oct 19 14:29:58 2016  0.000000 seconds
# poweroff

(Turn system power switch off)
(Wait for a short time)
(Turn on the system power switch)
…
(Login to the BSP)

# date
Wed Oct 19 14:32:52 UTC 2016
#
```

4.15 BQ27000 Gas Gauge Support

By default the BQ27000 Gas Gauge is enabled. If the battery is connected to the baseboard, enable gas gauge support with the following:

1. Enter the kernel configuration with one of the following options:

   From Linux Manual build directory

   - *make menuconfig ARCH=arm.*

   From Buildroot build directory

   - *make linuxmod-menuconfig*

2. Add support for 1-wire device drivers under Device Drivers > Dallas's 1-wire support (CONFIG_W1).
3. Add support for 1-wire master OMAP HDQ driver under Device Drivers > Dallas's 1-wire support > 1-wire Bus Masters > OMAP HDQ driver (CONFIG_HDQ_MASTER_OMAP).
4. Add support for 1-wire slave BQ27000 under Device Drivers > Dallas's 1-wire support > 1-wire slaves > BQ27000 slave support (CONFIG_W1_SLAVE_BQ27000).
5. Add support for the BQ27xxx family under Device Drivers > Power Supply Class Support > BQ27xxx battery driver (CONFIG_BATTERY_BQ27XXX).
6. Add support for the generic PDA power driver under Device Drivers > Power Supply Class Support > Generic PDA/phone power driver (CONFIG_PDA_POWER).

To check the status of the gas gauge:

```
# cd /sys/devices/w1_bus_master1/01-000000000000/bq27000-
battery/power_supply/bq27000-battery
# ls
capacity            energy_now          technology
capacity_level      health              temp
charge_full         manufacturer        time_to_empty_avg
charge_full_design  power               time_to_empty_now
charge_now          power_avg           time_to_full_now
current_now         present             type
cycle_count         status              uevent
device              subsystem           voltage_now
# cat voltage_now
4115000
#
```

The BQ27000 has specific requirements for learning the battery capacity. Please review the latest TI documentation for the BQ27000 and the Logic PD design checklist application note for your SOM for additional details:

- *AN 490 DM3730/AM3703 SOM-LV Design Checklist*[10]
- *AN 493 DM3730/AM3703 Torpedo SOM Design Checklist*[11]
- *AN 498 DM3730/AM3703 Torpedo + Wireless SOM Design Checklist*[12]

4.16 Run/Idle/Suspend

With power states such as run, idle, suspend, and off, there is always a balance between the power being saved and the amount of time it takes to go into and out of those states. Typically, the lowest power states take the longest time to wake up. Tuning the timing of transition into various states depends greatly on the system application the OS is designed to fit into. The power states below can be utilized with the DM37x Linux BSP.

- Run – The kernel is in the run state when the kernel scheduler finds a job to do. The cpufreq governor may increase or decrease the processor speeds depending on load.
- Idle – The kernel is in the idle state when the kernel scheduler doesn't have a job to do. If there are no background tasks being performed and the system is waiting at the prompt, then the kernel is most often in the idle state. The cpufreq governor may lower the CPU frequency to conserve power.
- Suspend –The DM37x Linux BSP can also enter suspend mode from the command line using the command below.

To enable the ability to suspend, disable the kernel parameter 'no_console_suspend' in U-Boot which prevents the console from sleeping.

```
OMAP Logic # nand unlock
OMAP Logic # setenv optargs 'ignore_loglevel early_printk'
OMAP Logic # saveenv
```

[10] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=696
[11] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=586
[12] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=637


```
OMAP Logic # reset
```

To suspend the system from the shell prompt, enter the following:

```
# echo mem > /sys/power/state
```

To exit the suspend state, press the S2 button.

NOTE: The backlight has a timeout independent of the run/idle/suspend power modes.

For more information, please see the appropriate Logic PD thermal management white paper for your SOM:

- *WP 540 DM3730/AM3703 SOM-LV Thermal Management*[13]
- *WP 491 DM3730/AM3703 Torpedo SOM Thermal Management*[14]
- *WP 530 DM3730/AM3703 Torpedo + Wireless SOM Thermal Management*[15]

4.17 Virtual Files

Linux is built on the foundation of a virtual filesystem. That is, the filesystem contains program files, data files, and virtual files. Virtual files are those files that don't actually store data in any memory, but rather are interfaces into a driver. Most Linux commands that operate on files also operate on virtual files.

4.17.1 *echo* Command and ">" Operator

Consider the following example. The ">" character is used to direct data to a file. In this case, we use it to direct the output of the *echo* command to the virtual file */dev/console*. The */dev/console* driver manages the interface to the UART terminal.

```
# echo "virtual file test" > /dev/console
virtual file test
DM-37x#
```

By directing the output of the *echo* command to the virtual file */dev/console*, we see the output of the *echo* command displayed on the terminal.

4.17.2 Debug FS

The */sys/kernel/debug* directory contains a vast list of virtual files, all of which are intended to provide debug information. It is beyond the scope of this manual to describe every file.

```
# mount -t debugfs none /sys/kernel/debug
```

[13] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=717
[14] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=605
[15] http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=656

To see a description of what a specific virtual file may contain, please consult the open source community, post a question to the Logic PD TDG forum, and/or search for the virtual file in the source code.

4.17.2.1 Clock Tree

There are many clocks running in the CPU, and developing or debugging a driver may require knowledge of the frequencies to which these clocks are set. The */sys/kernel/debug/clk* directory is the root of the clock tree. The virtual files are constructed in a tree structure, just as the clock tree is constructed in the CPU. To traverse the clock tree, see the CPU reference manual.

In the example below, we traverse the clock tree and display the rate of clocks DPLL1, DPLL2, DPLL3, DPLL4, and DPLL5.

```
# cat /sys/kernel/debug/clk/dpll1_ck/clk_rate
300000000
# cat /sys/kernel/debug/clk/dpll2_ck/clk_rate
260000000
# cat /sys/kernel/debug/clk/dpll3_ck/clk_rate
400000000
# cat /sys/kernel/debug/clk/dpll4_ck/clk_rate
864000000
# cat /sys/kernel/debug/clk/dpll5_ck/clk_rate
959833333
```

4.17.2.2 Pin Mux

Below is an example in which we view the pin mux of the CPU. The pin muxing is setup by the base of the register used to configure the pin *48002030.pinmux, 48002a00.pinmux,* and *480025a0.pinmux* are directories located in /sys/kernel/debug/pinctrl. Within each of those directories are the following directories: *gpio-ranges, pinmux-functions, pins, pingroups,* and *pinmux-pins*.

Pinmux-pins shows which mux register is assigned to which function. For example, the following shows that 480020e8 busing used as the of the dss pins.

```
# cat /sys/kernel/debug/pinctrl/48002030.pinmux/pinmux-pins

…
pin 92 (480020e8.0): 48050000.dss (GPIO UNCLAIMED) function pinmux_dss_dpi_pins1 group
pinmux_dss_dpi_pins1
…
```

Pin groups breaks the pins into groups and shows what device tree entries use which pins within a given control register range. Below shows which pins are assigned to the mcspi1_pins group.

```
# cat /sys/kernel/debug/pinctrl/48002030.pinmux/pingroups

…
```



```
group: pinmux_mcspi1_pins
pin 204 (480021c8.0)
pin 205 (480021ca.0)
pin 206 (480021cc.0)
pin 207 (480021ce.0)
…
```

To read the muxing configuration of a given pin read the contents of *pins*. The value shown corresponds to the value. For a description of the contents of the registers, see *Table 13-82: CONTROL_PADCONF_X* in the DM3730 Technical Reference Manual.

```
# cat /sys/kernel/debug/pinctrl/48002030.pinmux/pins

…
pin 204 (480021c8.0) 00000100 pinctrl-single
pin 205 (480021ca.0) 00000000 pinctrl-single
pin 206 (480021cc.0) 00000118 pinctrl-single
pin 207 (480021ce.0) 00000000 pinctrl-single
…
```

An alternative to reading the debugfs is to directly read using devmem.

```
# devmem 0x480021c8
0x00000100
```

4.18 Shut Down Linux System

To properly shut down the DM3730 Development Kit while running the Linux OS, use either the *poweroff* or *reboot* command. It is important to use one of these commands when shutting down the system, as they ensure any cached data is flushed out to the hardware prior to removing power. This is most important with non-volatile memory devices. If the cache is not flushed, there is a potential to corrupt the data in that memory device.

Note: The Linux community has configured the PMIC to require the power button (S2) on the system after the system has been powered down.

4.19 CPU Benchmarks

Adding the miscellaneous benchmark package will add dhrystone and whetstone benchmark tools to the */usr/bin* sub-directory.

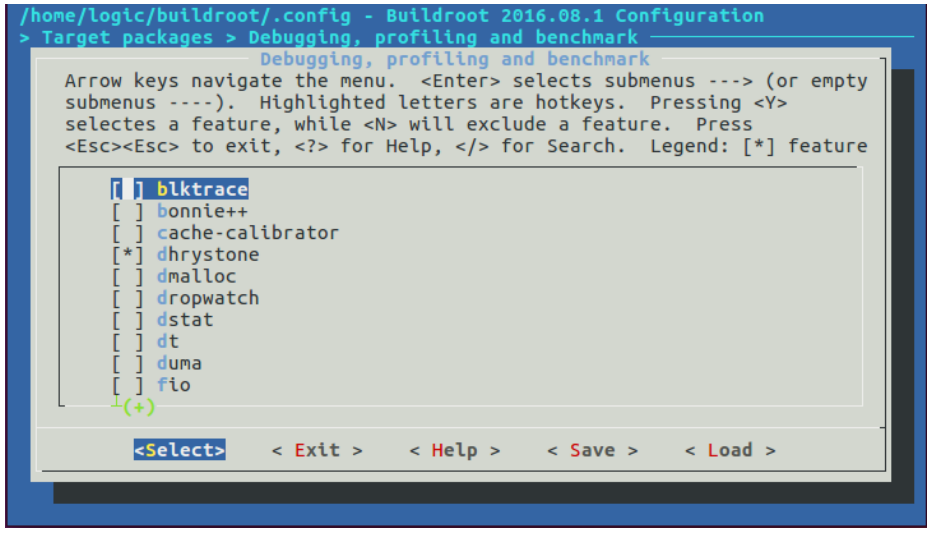
Benchmark packages have been included by default.

To locate the benchmark packages with Buildroot configuration menu, follow the steps below.

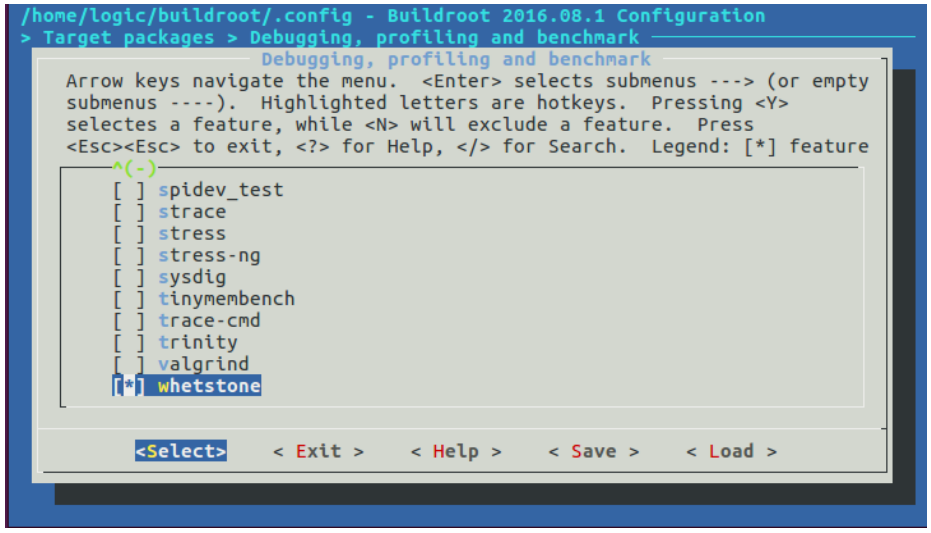
1. At the bash prompt, enter the following command to start the BUILDROOT menu system.

```
$ make menuconfig
```


2. In the main menu, select Target packages -> Debugging, profiling and benchmark (BR2_PACKAGE_DHRYSTONE)



3. In the *Debugging, profiling and benchmark* menu, select dhrystone and/or whetstone benchmarks (BR2_PACKAGE_WHETSTONE).



4. Save changes by exiting each sub menu.
5. Build the rootfs

```
$ make
```

6. Boot the SOM from the newly generated RootFS
7. Below are example commands for running the miscellaneous benchmarks.

```
# dhrystone

Dhrystone Benchmark, Version 2.1 (Language: C)
```



```
Program compiled without 'register' attribute

Please give the number of runs through the benchmark: 10000000

Execution starts, 10000000 runs through Dhrystone
Execution ends

Final values of the variables used in the benchmark:

Int Glob:            5
        should be:   5
Bool Glob:           1
        should be:   1
Ch 1 Glob:           A
        should be:   A
Ch 2 Glob:           B
        should be:   B
Arr 1 Glob[8]:       7
        should be:   7
Arr 2 Glob[8][7]:    10000010
        should be:   Number Of Runs + 10
Ptr Glob->
  Ptr Comp:          151560
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum Comp:         2
        should be:   2
  Int Comp:          17
        should be:   17
  Str Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next Ptr Glob->
  Ptr Comp:          151560
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum Comp:         1
        should be:   1
  Int Comp:          18
        should be:   18
  Str Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int 1 Loc:           5
        should be:   5
Int 2 Loc:           13
        should be:   13
Int 3 Loc:           7
        should be:   7
Enum Loc:            1
        should be:   1
Str 1 Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
```



```
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

Microseconds for one run through Dhrystone:    0.4
Dhrystones per Second:                      2824858.8
```

For Whetstone benchmark

```
# whetstone 60000
[  175.563568] random: crng init done

Loops: 60000, Iterations: 1, Duration: 13 sec.
C Converted Double
```

4.20 Use *devmem* to Examine/Modify Registers

Buildroot has the *devmem* package included. *Devmem* allows read/write access to physically addressable memory locations, which include not only DDR RAM, but also NOR flash, SRAM, and any memory-mapped I/O registers.

Below is an example of how to read the DM3730/AM3703 processor CONTROL.CONTROL_IDCODE[31:0] register.

```
# devmem 0x4830A204 32
0x2B89102F
```

The command below will cause a software reset by setting the global software reset control bit in the PRM_RSTCTRL register.

```
# devmem 0x48307250 32   0x02

U-Boot SPL 2018.03 (Mar 23 2018 - 14:03:53 -0500)
Trying to boot from NAND
The Expected Linux image was not found. Please check your NAND
configuration.
Trying to start u-boot now...


U-Boot 2018.03 (Mar 23 2018 - 14:03:53 -0500)

OMAP3630/3730-GP ES1.2, CPU-OPP2, L3-200MHz, Max CPU Clock 1 GHz
Model: LogicPD Zoom OMAP3 Development Kit
Logic DM37x/OMAP35x reference board + LPDDR/NAND…
```

4.21 Filesystem Commands

This section provides useful filesystem commands.

4.21.1 *df* Command

The *df* command reports how much free space is available for each mount.


```
# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root              3500416    327320   2991952  10% /
devtmpfs                111056         0    111056   0% /dev
tmpfs                   119468         0    119468   0% /dev/shm
tmpfs                   119468        64    119404   0% /tmp
tmpfs                   119468       124    119344   0% /run
/dev/mmcblk1p1          307016    123488    183528  40% /root/sd
```

4.21.2 *cat /proc/mtd* Command

The *cat /proc/mtd* command displays the flash partition table from the command line.

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00020000 "MLO"
mtd1: 001e0000 00020000 "U-Boot "
mtd2: 00020000 00020000 "U-Boot -env"
mtd3: 00400000 00020000 "kernel"
mtd4: 1f980000 00020000 "fs"
```

4.22 Using Linux Voltage and Current Regulator Framework

The TPS65950 regulates a variety of power domains which are made available through the Voltage and Current Regulator Framework.

The regulator used on the DM3730 modules can source different power levels and are addressed through the I2C1 bus. The Linux Voltage and Current Regulator Framework provide filesystem level access to read the settings of these regulators. The locations are found in /sys/class/regulator.

Table 4.32:AM/DM37xx Regulator Voltages

Regulator Path	Name	Min microvolts	Max microvolts	Voltage microvolts
regulator.0	regulator-dummy			
regulator.1	vwl1271			1800000
regulator.2	vddvario			
regulator.3	vdd33a			
regulator.4	pbias_mmc_omap2430	1800000	3000000	3000000
regulator.5	abb_mpu_iva	1012500	1375000	-22
regulator.6	fxed-supply			3300000
regulator.7	VAUX1	3000000	3000000	3000000
regulator.8	VAUX2_4030	0	0	1800000
regulator.9	VAUX3	0	0	2800000
regulator.10	VAUX4	1800000	1800000	1800000

regulator.11	VDD1	600000	1450000	1200000
regulator.12	VDAC	1800000	1800000	1800000
regulator.13	VIO	0	0	1800000
regulator.14	VINTANA1			1500000
regulator.15	VINTANA2	0	0	2750000
regulator.16	VINTDIG			1500000
regulator.17	VMMC1	1850000	3150000	3000000
regulator.18	VMMC2	1850000	3150000	2600000
regulator.19	VUSB1V5			1500000
regulator.20	VUSB1V8			1800000
regulator.21	VUSB3V1			3100000
regulator.22	VPLL1	0	0	1800000
regulator.23	VPLL2	1800000	1800000	1800000
regulator.24	VSIM	1800000	3000000	1800000

4.22.1 Read Regulator Names

```
# cat /sys/class/regulator/regulator.5/name
abb mpu iva
```

4.22.2 Read Current Regulator Voltage

```
# cat /sys/class/regulator/regulator.16/microvolts
1500000
```

4.22.3 Read Minimum Regulator Voltage

```
# cat /sys/class/regulator/regulator.17/min_microvolts
1850000
```

4.22.4 Read Maximum Regulator Voltage

```
# cat /sys/class/regulator/regulator.5/max_microvolts
1375000
```

4.22.5 Read Regulator State

```
# cat /sys/class/regulator/regulator.16/state
enabled
```


4.23 WL12xx WiFi

As of the publication of this document, the SOM-LV 802.11 device tree was incomplete. Updates will become available when the feature has been enabled.

Below are the requirements needed for both WiFi Station Mode and WiFi Access Point.

Follow these steps to configure the correct NVS file for your system.

1. Due to a bug in the WL12xx driver, the driver for WL1283 wrongly loads the WL127x driver. The following will point the wl127z driver to the wl128x nvs file and reboot.

NOTE: For FCC compliance, the NVS file that corresponds to the SOM should be used instead of the default one provided by the Open Source community. To do that, create a symbolic link to the NVS file named *wl127x-nvs.bin*. The example below is for the SOMDM3730-32-xxx, but it can be adapted for other versions.

```
# cp wl128x-nvs-tw32.bin /lib/firmware/ti-connectivity

# cd /lib/firmware/ti-connectivity
# mv wl127x-nvs.bin wl127x-nvs.bin.save
# ln -s wl128x-nvs-tw32.bin wl127x-nvs.bin
```

2. Reboot for the NVS file to be reloaded

```
# reboot
```

NOTE: Systems using a RAMDisk boot solution will need to update the RAMDisk rootfs to include the changes seen above. If additional development assistance is need consider contacting Logic PD support services[16].

4.23.1 WiFi MAC Address

The MAC address for the WiFi chip is programmed into an EEPROM on the SOM. This data is recorded a way proprietary to Logic PD, so the patch provided enables a special driver to decode the EEPROM and place the contents into text files located in /sys/class/product_id

To automatically set the MAC address create a file called /etc/init.d/S90wifimac and insert the following:

```
insmod /lib/modules/4.9.89/kernel/drivers/misc/eeprom/logicpd-new-
productid.ko
ifconfig wlan0 hw ether `cat /sys/class/product id/wifi macaddr`
```

Once the file is created change the permission of the file with the following

```
# chmod +x /etc/init.d/S90wifimac
```

[16] https://www.logicpd.com/contact/

The S90wifimac script must be run before continuing. Users can either execute the script now using the following command or reboot their system.

```
# /etc/init.d/S90wifimac
```

This file will automatically install the kernel module for reading the EEPROM, then use *ifconfig* to set the WiFi MAC address with the contents of */sys/class/product_id/wifi_macaddr*.

4.23.2 WiFi Station Mode

To connect to an Access Point, WPA_SUPPLICANT is required.

1. Configure wpa_supplicant to connect to the router. The example below assumes a WPA2 with a shared key. For other configuration options, check out: WPA Supplicant README

   Edit /etc/wpa_supplicant.conf:

```
# cp /etc/wpa_supplicant.conf /etc/wpa_supplicant.conf.save
# vi /etc/wpa_supplicant.conf
```

   Add the following contents to wpa_supplicant.conf:

```
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1

network={
        ssid="Your SSID"
        psk="Your Secret Shared Key"
}
```

2. Save and exit the editor
3. Invoke wpa_supplicant in the background with the following:

```
# wpa_supplicant -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf -B
```

4. Once wpa_supplicant has authenticated to the access point, the IP address can be set using dhcpc

```
# udhcpc -iwlan0
```

5. Confirm the network connection with ifconfig:


```
# ifconfig
wlan0     Link encap:Ethernet  HWaddr DE:AD:BE:EF:00:00
          inet addr:10.1.23.21  Bcast:10.1.23.255  Mask:255.255.255.0
          inet6 addr: fe80::dcad:beff:feef:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1946 (1.9 KiB)  TX bytes:3210 (3.1 KiB)
```

4.23.3 WiFi Access Point

NOTE: The Torpedo + Wireless was certified only for use in the United States and Canada, but the regulatory database includes settings for the world. To be compliant with FCC and IC, the regulatory database should be modified to only permit the country codes that were tested, DFS should be disabled, and the NVS file should match the model of the Torpedo + Wireless SOM being used. Please contact Logic PD for assistance.

1. Log into development board and correct the NVS file as shown in 4.23.1 WiFi Station Mode

2. Enable IP forwarding

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

3. Bring up the Ethernet network. See section 4.4.1.4 or 4.4.1.5.

```
# ifup eth0
```

4. Edit /etc/hostapd.conf to enable the bridge by searching for the following code and un-commenting the line labeled 'bridge=0' and comment out the line labeled 'eap_server=0'

```
# For nl80211, this parameter can be used to request the AP interface to be
# added to the bridge automatically (brctl may refuse to do this before hostapd
# has been started to change the interface mode). If needed, the bridge
# interface is also created.
bridge=br0

…
# Use integrated EAP server instead of external RADIUS authentication
# server. This is also needed if hostapd is configured to act as a
RADIUS
# authentication server.
# eap_server=0
```

5. *Optionally:* edit /etc/hostapd.conf to setup the SSID (Default=test) and WPA2 settings


```
…
ssid=<your desired ssid>
…
wpa=1
…
wpa_passphrase=<your desired passphase>
…
wpa_key_mgmt=WPA-PSK WPA-EAP
…
wpa_pairwise=CCMP TKIP
…
rsn_pairwise=CCMP
…
country_code=US
```

6. *Optionally:* edit /etc/hostapd.conf to set the Country Code (Default = 00), Channel Number (Default=1), Operation Mode (default=802.11g 2.4GHz), WPA-PSK (default off), and more. See hostapd documentation for more details.

7. Bring up the Access Point and the bridge

```
# hostapd -B /etc/hostapd.conf -P /var/run/hostapd.pid
Configuration file: /etc/hostapd.conf
rfkill: Cannot open RFKILL control device
[  238.265991] wlcore: firmware booted (Rev 7.3.10.0.133)
[  238.302856] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[  238.357025] device wlan0 entered promiscuous mode
Using interface wlan0 with hwaddr de:ad:be:ef:00:00 and ssid "test"
[  238.386383] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

8. Add Ethernet (eth0) to bridge network (br0)

```
# brctl addif br0 eth0
```

9. Obtain an DHCP address on the bridge.

```
# udhcpc -i br0
```

10. Connect remote device to WiFi Network

4.23.4 Removing WIFI Support

Steps in this section will show users how to remove packages specific for wireless support.

1. Enter the buildroot menuconfig

   Make menuconfig


```
$ make menuconfig
```

2. Remove the Linux TI WL128X Firmware

```
Target Packages -> Hardware Handling -> Firmware -> WiFi Firmware -> WL128x
(BR2_PACKAGE_LINUX_FIRMWARE_TI_WL128X)
```

3. Remove the CRDA package.

```
Target Packages -> Networking Applications -> CRDA (BR2_PACKAGE_CRDA)
```

4. Remove hostapd used to act as an access point (BR2_PACKAGE_HOSTAPD, BR2_PACKAGE_HOSTAPD_ACS)

```
Target Packages->Networking Applications->hostapd (Enable ACS)
```

5. Remove bridge utils that connects the Ethernet to the Wifi (BR2_PACKAGE_BRIDGE_UTILS)

```
Target Package-> Networking Applications-> bridge-utils
```

6. Disable the Ethernet bridging in the Kernel Menuconfig (CONFIG_BRIDGE & CONFIG_BRIDGE_IGMP_SNOOPING)

```
$ make linux-menuconfig

Networking support-> Networking Options->802.1d Ethernet Bridging
```

7. Rebuild Linux

```
# make linux
```

8. Rebuilt rootfs with tools removed.

```
$ make
```

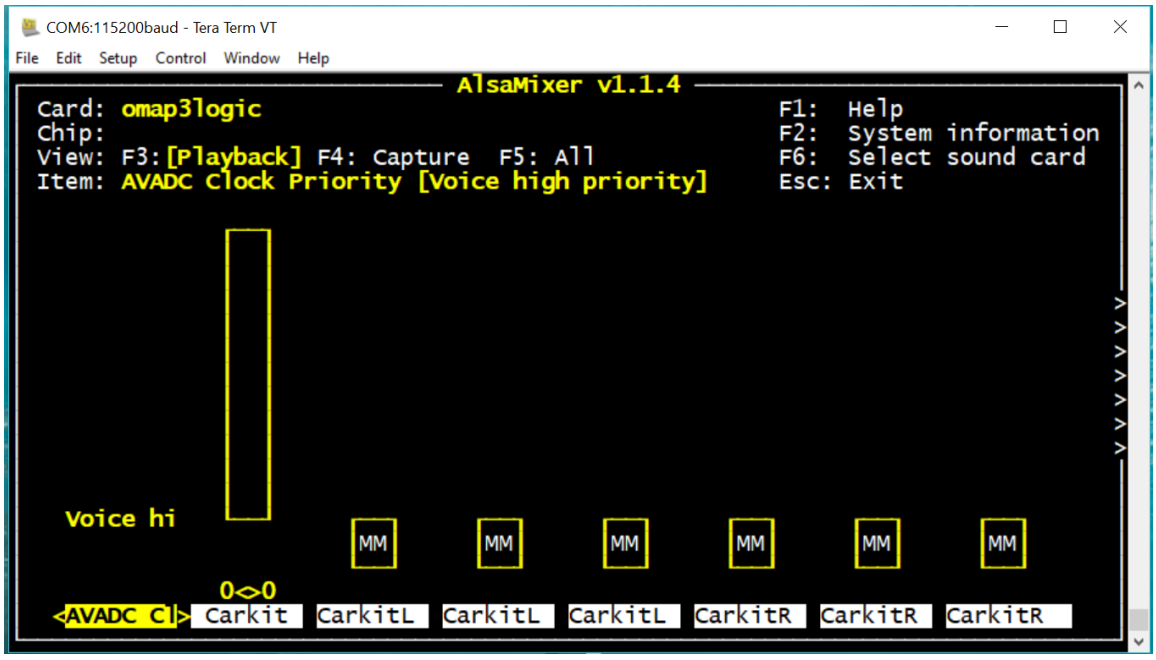
9. Boot the board using the new RootFS generated.

4.24 Audio Output

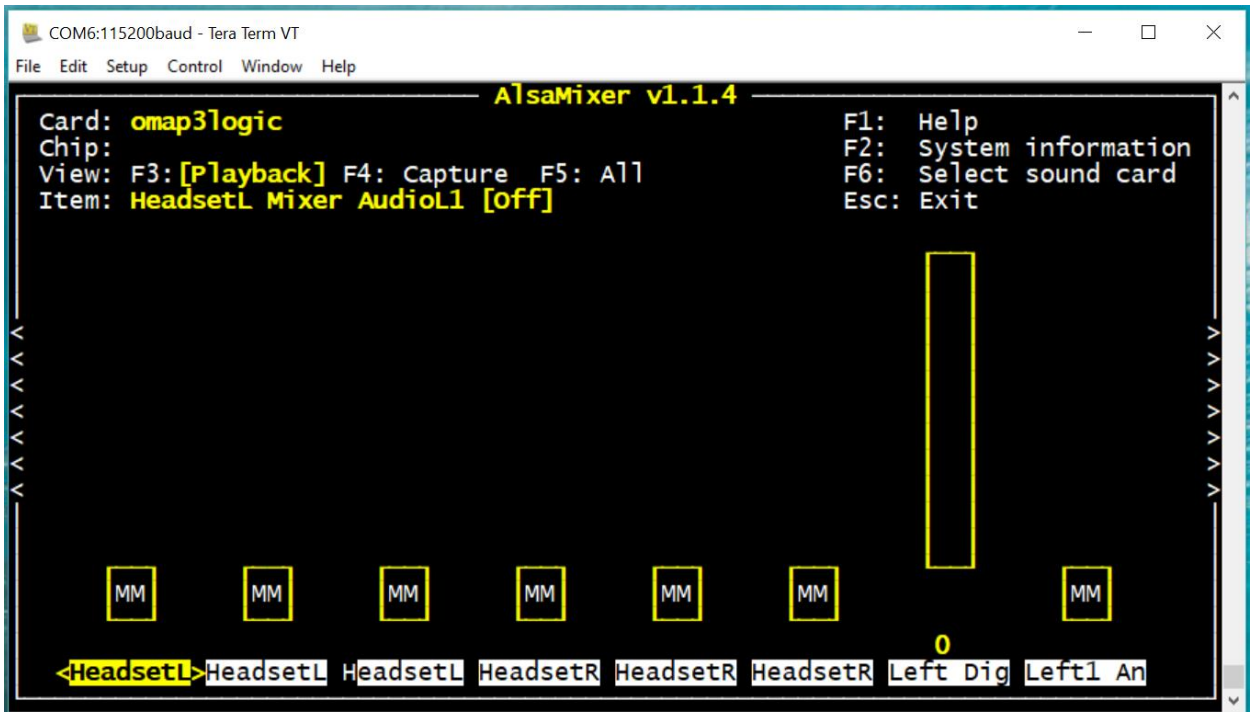
The audio on the baseboard is routed through the PMIC (twl4030) on the SOM. By default, the audio device is muted and needs to be unmuted before the audio can be heard. The audio control is provided by Advanced Linux Sound Architecture (ALSA). The BSP with kernel 4.9 has a few ALSA tools to facilitate the audio configuration.

1. Run *alsamixer* to unmute and set the audio levels.

```
$ alsamixer
```



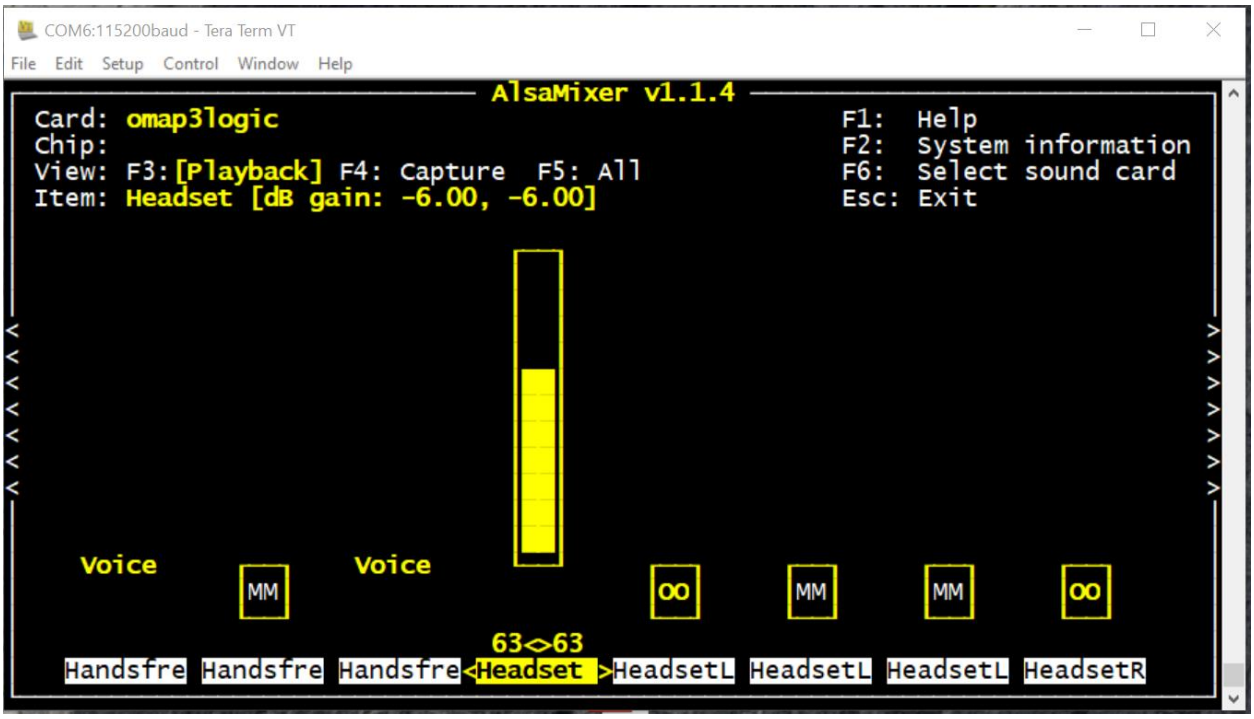
2. Locate Headset L1 and Headset R1 and unmute both by selecting each and pressing the 'm' button



Alternatively use the following command:

```
For "HeadsetL Mixer AudioL1"
$ amixer -c 0 cset numid=46 1
For "HeadsetL Mixer AudioR1"
$ amixer -c 0 cset numid=49 1
```

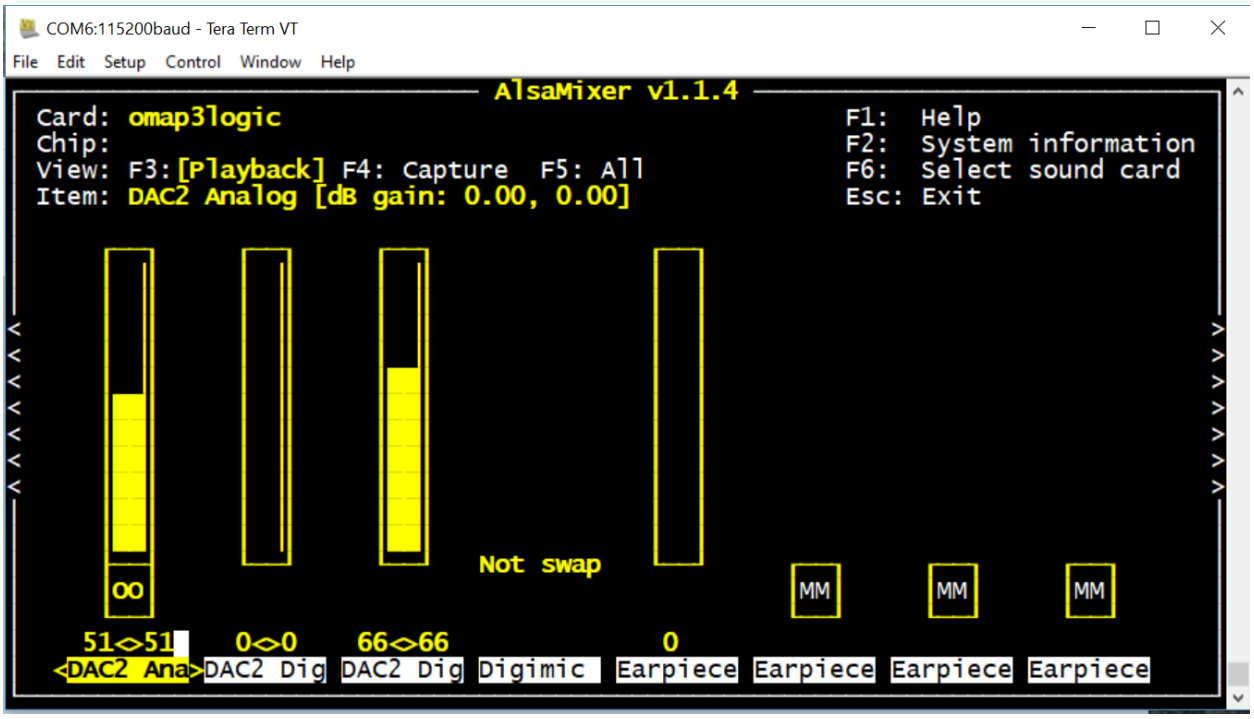
3. Select the headset and turn the volume to 63



Alternatively use the following command:

```
$ amixer -c 0 cset numid=14 1
```

4. Adjust DCA2 Analog to approximately 51 and DAC2 Digital Fine between 50-75 and Headset to about 66. These settings can be adjusted for personal preference.



Alternatively use the following command:

```
$ amixer set "DAC2 Analog" 18
$ amixer set "DAC2 Digital Fine" 63
$ amixer set "DAC2 Digital Coarse" 2
```

5. Exit the mixer by pressing Esc
6. Connect headphones to the Line Out jack on the baseboard
7. Run the speaker test to hear the audio hear a test sample on left and right audio channels.

```
$ speaker-test -t wave -c2
```

4.25 Camera Capture

Unlike the previous BSP's based on the 3.0.y kernel, the 4.4 kernel and beyond do not support DSP, so users who need DSP encoding or decoding should use the older BSP's. This section explains how to set up the parallel camera and provides examples of how to use it. The targeted parallel camera is the Leopard Imaging 5 megapixel LI-5M04 camera adapter board, which can be connected to J6 on the DM3730 Torpedo Development Kit. The parallel camera feature is not available on the DM3730 SOM-LV Development Kit.

Please connect the camera to the DM3730 Torpedo Development Kit before powering on the board. Note that JP5 needs a jumper across pins 1-2 to enable the 8-bit video data bus for standard SOM's. People with SOM's that support 12-bit video can place the jumper across pins 2-3. More information about Camera selection integration can be found on in the AN 596 Logic PD Development Kit Camera Module Selection and Integration application note. More information about the LI-5M04 camera adapter board is available on the Leopard Imaging website.[17]

1. Disable console blanking

   NOTE: The first command below is used to disable the LCD blanking timeout. This command can be used at any time and is not specific to the camera use.

```
DM-37x# echo -ne "\033[9;0]\033[?25l" > /dev/tty0
```

2. Prepare the system to use the LI-5M04 camera by using YAVTA to setup the Video For Linux Two (V4L2) system. Note the addresses below represent the various V4L2_CID_XXX controls exposed by the V4L2 subsystem. More information about the available controls can be found by running "yavta -l /dev/v4l-subdev8", or by referencing the Kernel Source.

   a. Gain (V4L2_CID_GAIN)

[17] http://shop.leopardimaging.com/product.sc?productId=24


```
DM-37x#  yavta -w '0x00980913 16' /dev/v4l-subdev8
Device /dev/v4l-subdev8 opened.
Control 0x00980913 set to 16, is 16
```

3. The tool, *media-ctl,* connects various hardware components together in the DM3730 Camera ISP so that video is piped to the correct place. The list of components can be found by running "media-ctl -p". The following command takes Bayer RGB from the camera, converts to UYVY in the Preview block, then scales with the resizer, and places the output in DDR via the V4L2 interface exposed by /dev/video6.

```
DM-37x# media-ctl -v -r -l '"mt9p031 1-0048":0->"OMAP3 ISP CCDC":0[1],
"OMAP3 ISP CCDC":2->"OMAP3 ISP preview":0[1], "OMAP3 ISP preview":1-
>"OMAP3 ISP resizer":0[1], "OMAP3 ISP resizer":1->"OMAP3 ISP resizer
output":0[1]'
Opening media device /dev/media0
Enumerating entities
looking up device: 81:7
looking up device: 81:0
looking up device: 81:8
looking up device: 81:1
looking up device: 81:9
looking up device: 81:2
looking up device: 81:10
looking up device: 81:3
looking up device: 81:4
looking up device: 81:11
looking up device: 81:5
looking up device: 81:6
looking up device: 81:12
looking up device: 81:13
looking up device: 81:14
looking up device: 81:15
Found 16 entities
Enumerating pads and links
Resetting all links to inactive
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Opening media device /dev/media0
Setting up link 81:0 -> 15:0 [1]
Opening media device /dev/media0
Setting up link 15:2 -> 23:0 [1]
Opening media device /dev/media0
Setting up link 23:1 -> 34:0 [1]
```



```
Opening media device /dev/media0
Setting up link 34:1 -> 41:0 [1]
Opening media device /dev/media0
```

4. Since the raw image is much larger than the processor can handle at 30 FPS, the image needs to be cropped. In the following example, the image is being copped to 1298x970 beginning at column 664 and row 541 on the sensor.

```
DM-37x# media-ctl -v -V '"mt9p031 1-0048":0 [SGRBG8 1298x970
(664,541)/1298x970], "OMAP3 ISP CCDC":2 [SGRBG10 1298x970], "OMAP3 ISP
preview":1 [UYVY 1298x970], "OMAP3 ISP resizer":1 [UYVY 480x272]'
Opening media device /dev/media0
Enumerating entities
looking up device: 81:7
looking up device: 81:0
looking up device: 81:8
looking up device: 81:1
looking up device: 81:9
looking up device: 81:2
looking up device: 81:10
looking up device: 81:3
looking up device: 81:4
looking up device: 81:11
looking up device: 81:5
looking up device: 81:6
looking up device: 81:12
looking up device: 81:13
looking up device: 81:14
looking up device: 81:15
Found 16 entities
Enumerating pads and links
Setting up selection target 0 rectangle (664,541)/1298x970 on pad
mt9p031 1-0048/0
Selection rectangle set: (664,542)/1298x970
Setting up format SGRBG8 1X8 1298x970 on pad mt9p031 1-0048/0
Format set: SGRBG8 1X8 1298x970
Setting up format SGRBG8 1X8 1298x970 on pad OMAP3 ISP CCDC/0
Format set: SGRBG8 1X8 1298x970
Setting up format SGRBG10 1X10 1298x970 on pad OMAP3 ISP CCDC/2
Format set: SGRBG8 1X8 1298x969
Setting up format SGRBG8 1X8 1298x969 on pad OMAP3 ISP preview/0
Format set: SGRBG8 1X8 1298x969
Setting up format SGRBG8 1X8 1298x969 on pad OMAP3 ISP AEWB/0
Unable to set format: Inappropriate ioctl for device (-25)
Setting up format SGRBG8 1X8 1298x969 on pad OMAP3 ISP AF/0
Unable to set format: Inappropriate ioctl for device (-25)
Setting up format SGRBG8 1X8 1298x969 on pad OMAP3 ISP histogram/0
Unable to set format: Inappropriate ioctl for device (-25)
Setting up format UYVY8 1X16 1298x970 on pad OMAP3 ISP preview/1
Format set: UYVY8 1X16 1280x961
Setting up format UYVY8 1X16 1280x961 on pad OMAP3 ISP resizer/0
Format set: UYVY8 1X16 1280x961
Setting up format UYVY8_1X16 480x272 on pad OMAP3 ISP resizer/1
```



```
Format set: UYVY8 1X16 480x272
```

5. We need to set the location of the LD_PRELOAD so the tools where the libv4l libraries are located.

```
DM-37x# export LD_PRELOAD=/usr/lib/libv4l/v4l2convert.so
```

6. Use FFMPEG to display live video from the camera on the LCD. Because the DSP is absent, the video is all handled by the ARM processor, and the throughput is limited and may not achieve 30 frames per second.

```
DM-37x# ffmpeg -an -re -i /dev/video6 -f v4l2  -vcodec rawvideo -
pix_fmt rgb565le -f fbdev /dev/fb0
```


5 Boot Kernel from TFTP Server and Root Filesystem from NFS Server

This section explains how to configure both a Ubuntu 16.04 host PC and a DM3730/AM3703 target platform to boot the DM37x Linux BSP kernel from a TFTP server and the DM37x Linux BSP root filesystem from a network file system (NFS) server.

Having the kernel located on a TFTP server on the host PC allows developers to quickly change the kernel on their Linux host PC and test it on the DM3730/AM3703 target platform by simply rebooting the target platform. Other options for testing an updated the kernel would be to copy the updated kernel to an SD card or burn it into flash. These options work, but take more time.

NFS allows the host PC to share directories and files with the target platform over a network. Developers can use NFS to quickly change their application and test it on the target platform when the root filesystem is located in the host PC. Updates to an NFS can be seen by the target immediately and do not require the system to be reset.

This example will use the Virtual Machine SDK for the DM37x Linux BSP as the starting point for the Ubuntu 16.04 environment. These steps have been tested using the Virtual Machine SDK provided with the DM37x Mainstream Linux images. Other versions may require slight changes to the steps below.

5.1 Set Up TFTP Server in Ubuntu 16.04

This section describes the steps needed to install and run the TFTP server in Ubuntu 16.04. This assumes the steps for building the kernel and modules have been completed as described in section 2.5. For the example below, the device tree must be appended to the kernel. See section 2.6.

1. Install a TFTP server in Ubuntu 16.04.

```
$ sudo apt-get install tftpd-hpa
```

   Once the installation is complete, the tftp server will be running on the system listening on all active network interfaces.

   The default configuration file for tftpd-hpa is /etc/default/tftpd-hpa

   The default root directory where files will be stored is /var/lib/tftpboot

2. Create the TFTP configuration file.

```
$ sudo gedit /etc/default/tftpd-hpa
```

3. Edit the `TFTP_DIRECTORY` to change the TFTP directory.

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_ADDRESS="[::]:69"
TFTP_OPTIONS="--secure --create"
```


4. Create the tftp directory (*/var/lib/tftpboot)*. This must match the folder assigned to TFTP DIRECTORY in the TFTP configuration file in the step above.

```
$ sudo mkdir /var/lib/tftpboot
$ sudo chmod -R 777 /var/lib/tftpboot
$ sudo chown -R tftp /var/lib/tftpboot
```

5. Restart the tftpd-hpa service.

```
$ sudo service tftpd-hpa restart
```

Now the TFTP server is up and running.

6. Copy the kernel with appended device tree into the tftp directory (/var/lib*/tftpboot).* See section 2.6 for instructions on appending the device tree to the zImage.

```
$ cp ~/logic/buildroot/output/images/zImage_dtb
/var/lib/tftpboot/zImage
```

5.2 Setup NFS Server in Ubuntu 16.04

This section describes the steps needed to install and run the NFS server in Ubuntu 16.04. In addition, steps are provided that explain how to move the files from the build root filesystem to the NFS root filesystem.

7. NFS Server must be installed. Skip this step if the NFS server is already installed.

```
$ sudo apt install nfs-kernel-server
```

8. Set up a directory for the NFS server.

```
$ sudo mkdir -p /opt/nfs-exports/omap
```

9. Populate the */opt/nfs-exports/omap* directory with the content of the root filesystem generated by Buildroot.

```
$ sudo tar xvf /home/logic/logic/buildroot/output/images/rootfs.tar -C
/opt/nfs-exports/omap
```

10. Edit the*/etc/exports* configuration file.

```
$ sudo gedit /etc/exports
```

11. Add the following line to the end of the */etc/exports* configuration file. Both lines below must appear as a single line in the */etc/exports* configuration file.


```
/opt/nfs-exports/omap
192.168.120.0/24(rw,async,insecure,no root squash,no subtree check)
```

This tells the NFS server that anyone in 192.168.120.0, netmask 255.255.255.0 can mount */opt/nfs-export/omap*. You can change the IP range/netmask bits to whatever you need; the options for doing so are described in more detail here.[18]

12. Restart the nfs-kernel-server service.

```
$ service nfs-kernel-server restart
```

13. Also after making changes to */etc/exports* in a terminal, you must export all directories using the following command.

```
$ sudo exportfs -a
```

5.3 Set Up the DM3730/AM3703 Target Platform

This section explains how to set up the DM3730/AM3703 development platform to boot the DM37x Linux kernel from the TFTP server and use the DM37x Linux root filesystem on the NFS server located on the Ubuntu 16.04 Linux host PC.

1. Unlock NAND memory

```
OMAP Logic # nand unlock
```

2. Set up the U-Boot environment to an initial state.

```
OMAP Logic # env default -f -a; setenv preboot
```

3. Setup the nfs-boot script on U-boot.

The following example shows how to connect using *DHCP* to get an IP address.

```
OMAP Logic # setenv nfsargs 'run setconsole; setenv serverip
${tftpserver}; setenv bootargs console=${console} root=/dev/nfs
nfsroot=${nfsrootpath}
ip=${ipaddr}:${tftpserver}:${gatewayip}:${netmask}::eth0:off'

OMAP Logic # setenv nfsboot 'dhcp; run nfsargs; run common_bootargs;
run dump_bootargs; tftpboot $loadaddr zImage;bootz $loadaddr'

OMAP Logic # setenv autoload no
```

The following example shows to connect using a *static IP* address. Where ipaddr is the static IP address of the device. Please note the netmask and gateway ip may vary depending on

[18] http://linux.die.net/man/5/exports

individual network configuration. Set the IP address of the DM3730/AM3703 target system. The IP address in this example was set to 192.168.120.100

```
OMAP Logic # setenv ipaddr 192.168.120.100
OMAP Logic # setenv gatewayip 192.168.120.100
OMAP Logic # setenv netmask 255.255.255.0

OMAP Logic # setenv nfsargs 'run setconsole; setenv serverip
${tftpserver}; setenv bootargs console=${console} root=/dev/nfs
nfsroot=${nfsrootpath}
ip=${ipaddr}:${tftpserver}:${gatewayip}:${netmask}::eth0:off'

OMAP Logic # setenv nfsboot 'run nfsargs; run common_bootargs; run
dump_bootargs; tftpboot $loadaddr zImage;bootz $loadaddr'
```

4. Set the *tftpserver* to match the correct server location. In this example the *tftpserver* is at 192.168.120.53.

```
OMAP Logic # setenv tftpserver 192.168.120.53
```

5. Set the directory in the NFS server to use the root filesystem.

   NOTE: This directory must match the directory used in Section 5.2.

```
OMAP Logic # setenv nfsrootpath /opt/nfs-exports/omap
```

6. Set the default boot sequence to load from network.

```
OMAP Logic # setenv defaultboot 'run nfsboot'
```

7. Save the U-Boot environment variables for future boots.

```
OMAP Logic # saveenv
```

8. Execute the following command to reboot the system and load the kernel from the TFTP server. The root filesystem will now point to the one located on the Linux host PC at the location defined at *nfsrootpath*.

```
OMAP Logic # reset
```


6 Application Development

This section describes some application development fundamentals for the Linux BSP. There are an infinite number of approaches to developing applications for embedded Linux. However, the following fundamental constraints on development must be taken into consideration:

- The application must be linked against the current kernel glibc runtime library.
- Building the application should be cross compiled on a host PC and then transferred to the target device. Theoretically, compiling and linking can take place on the target device. However, there are so many more resources available on a host PC (namely disk space and speed) that make cross compiling using the desktop the preferred method.
- A method for debugging must be established. At times, this could simply be *printf()* statements sprinkled throughout the code. However, faster and more convenient methods consist of a debugger application capable of displaying source and setting break points.

7 Basic Driver/Kernel Debugging Information

This section provides basic debug tools to debug the kernel and custom driver.

7.1 Logs

The command *dmesg* (for display message) is available on some Unix-like operating systems and it prints the internal message buffer ring of the kernel. To get the complete log, see */var/log/messages*.

The log levels allow you to see kernel messages specific to the level set. A level of 7 will display all possible messages. A level of 0 will only display kernel emergency messages indicating the system is about to crash or is unstable.

To view the current console_loglevel, enter the command below at the kernel prompt.

```
# cat /proc/sys/kernel/printk
7       4       1       7
```

The first integer shows the current console_loglevel; the second shows the default log level, the third shows the minimum level, and the last integer shows the boot time default log level.

By default, the optargs parameter is set to ignore_loglevel, which means the console log messages cannot be turned off. Remove that from optargs in the U-Boot environment, and use the command below from Linux shell prompt to set the log level that will be output to the console. Once running Linux, all messages are still saved in */var/log/messages*

```
# dmesg -n <0-7>
```

More information on kernel log levels can be found in the Linux *Debugging by print* wiki article.[19]

7.2 Debug Modules

Information for specific modules is readily available using the commands below.

The lsmod command provides developers a way to list modules loaded in the kernel.

```
# lsmod

lsmod
Module                  Size  Used by
omapfb                 39469  1
cfbfillrect             3614  1 omapfb
cfbimgblt               2416  1 omapfb
cfbcopyarea             3187  1 omapfb
evdev                  13085  0
cpufreq_dt              5617  0
joydev                 10008  0
```

[19] http://elinux.org/Debugging_by_printing#Log_Levels


```
thermal sys            59384  1 cpufreq dt
leds gpio               3826  0
hwmon                   4213  1 thermal sys
led class               5482  1 leds gpio
gpio keys               8915  0
panel dpi               3708  1
pwm bl                  4349  0
pwm omap dmtimer        4158  1
omap wdt                5293  0
omap hdq                6609  0
wire                   29347  1 omap hdq
cn                      5385  1 wire
twl4030 wdt             2711  0
pwm twl led             4520  0
twl4030 charger        10273  0
twl4030 pwrbutton       2397  0
pwm twl                 4395  0
industrialio           39947  1 twl4030 charger
ohci omap3              2781  0
tsc2004                 2312  0
omapdss               251496  3 panel dpi,omapfb
tsc200x core            7465  1 tsc2004
ohci_hcd               30355  1 ohci_omap3
```

The *modinfo* command provides information about the Linux kernel module.

```
# modinfo <module_name>
```

The *modprobe -c* | *less* command lists the comprehensive configuration for all modules.

```
# modprobe -c | less
```

The *grep* command displays the configuration of a particular module.

```
# modprobe -c | grep <module_name>
```

The *modprobe* command lists the dependencies of a module (or alias), including the module itself.

```
# modprobe --show-depends <module_name>
```

For example:

```
# modprobe --show-depends omapfb
builtin omapfb #
```

The *ls* command lists available module parameters.


```
# ls /sys/module/<module_name>/parameters
```

For example:

```
# ls /sys/module/omapfb/parameters/
auto update freq  debug             test
```

