



DM37x Linux BSP User Guide

BSP Documentation

Logic PD // Products
Published: July 2011
Last revised: December 2016

This document contains valuable proprietary and confidential information and the attached file contains source code, ideas, and techniques that are owned by Logic PD, Inc. (collectively "Logic PD's Proprietary Information"). Logic PD's Proprietary Information may not be used by or disclosed to any third party except under written license from Logic PD, Inc.

Logic PD, Inc. makes no representation or warranties of any nature or kind regarding Logic PD's Proprietary Information or any products offered by Logic PD, Inc. Logic PD's Proprietary Information is disclosed herein pursuant and subject to the terms and conditions of a duly executed license or agreement to purchase or lease equipment. The only warranties made by Logic PD, Inc., if any, with respect to any products described in this document are set forth in such license or agreement. Logic PD, Inc. shall have no liability of any kind, express or implied, arising out of the use of the Information in this document, including direct, indirect, special or consequential damages.

Logic PD, Inc. may have patents, patent applications, trademarks, copyrights, trade secrets, or other intellectual property rights pertaining to Logic PD's Proprietary Information and products described in this document (collectively "Logic PD's Intellectual Property"). Except as expressly provided in any written license or agreement from Logic PD, Inc., this document and the information contained therein does not create any license to Logic PD's Intellectual Property.

The Information contained herein is subject to change without notice. Revisions may be issued regarding changes and/or additions.

© Copyright 2016, Logic PD, Inc. All Rights Reserved.

Revision History

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	EN, SO	-Initial Release	JCA	07/28/11
B	EN, RAH	-Throughout: Added references to DM3730 Torpedo + Wireless SOM; Updated commands for BSP version 2.0; Reorganized flow of information; -Added Section 1.8; -Added Section 2.1 -Section 2.3.1.3: Corrected alternate command script included in notes section to install LWP: UserAgent package; -Added Section 1.1; -Removed Section 3.13.4 – 3.13.6: Combined information into Section 3.2.10 and updated; Changed kernel parameters; -Added Section 3.2; -Section 4.7.3: Updated backlight commands; -Section 0: Changed references of <i>mtdblock3</i> to <i>mtdblock4</i> throughout; -Added Section 4.13 -Section 4.17: Removed references to <i>musbd</i> images, as only standard sample images are provided; -Section 4.17.2: Corrected mount point to /mnt/mmcbk0p1. -Section 4.18 - 4.24: Additional peripheral information added: UARTs, I2C, SPI, Real Time Clock, ADCs, and Run/Idle/Suspend mode; -Added Section 4.28;	EN, DH	03/29/12
C	EN	-Section 1.5: Updated to clarify different development path options; -Section 2.4: Added <i>image.elf</i> file as one available after completion of build; -Added Section 2.5.5.1; -Added Section 2.7; -Section 3.2: Updated link to U-Boot command manual; -Added Section 3.2.6; -Section 3.2.10.4: Added note regarding the use of <i>nand write</i> and <i>nand write.l</i> ; -Added Section 3.2.11; -Added Section 4.30; -Added Section 7; -Added Section 8; -Added Section 9; -Added references to <i>README-setup</i> within tar ball to obtain the latest information regarding the tar ball	EN, BSB	05/18/12
D	EN, SO	-Added Section: 1.2, 1.3, and 1.5; -Section 1.8: Added link to <i>DM37x Linux SD Card Demo ReadMe</i> ; -Added Section 1.9; -Section 2: Added description of tasks performed by LTIB; -Section 2.2.1: Added additional information on patch files, patch command, etc.; -Section 2.3.1: Added note that following section only provides an example set of tasks; added note about script that can automate steps in the section; -Section 2.3.1.3: Updated initial command to <i>sudo su</i> ; added note that <i>sudo su</i> command may be required; -Section 2.4: Added note to ignore error message prompted by GNOME bug; -Added Section 2.4.1, 2.4.2, 2.5.6, and 2.5.7; -Section 2.7.3: Updated name of directory where CodeSourcery is installed; -Section 3: Removed NoLo as possible second-stage bootloader; removed LogicLoader as possible third-stage bootloader; added requirement that second and third-stage bootloaders must be stored in the same location; -Section 3.2: Removed I2C reads and writes from list of U-Boot tasks; -Section 3.2.3: Added note that U-Boot commands may differ based on version; -Section 3.2.5: Changed <i>set</i> command to <i>setenv</i> ; -Section 3.2.6: Added note to Step 3 that memory space can be	RAH, SO	11/21/12

		<p>reused or discarded as soon as bitmap shows on the display;</p> <ul style="list-style-type: none"> -Section 3.2.7: Changed <i>set</i> command to <i>setenv</i> throughout; -Section 3.2.10: Added note about possible need for different ECC algorithm depending on hardware configuration; -Section 3.2.10.4: Added reference to a script that can automate tasks described in the section; added note to Step 6 about need to use version-specific RAM disk image size; updated ECC type in Step 8; -Section 3.2.10.5: Added reference to a script that can automate tasks described in the section; added Step 3 to erase NAND flash; updated ECC type in Step 8; -Added Section 4.5.1.1; -Section 4.5.1.2: Added reminder to have Ethernet cable connected when booting; added note about how to save configuration changes across power cycles; -Section 0: Added reminder that quotation marks are required around field strings when using the <i>amixer</i> command; -Section 0: Added note that mounting location is arbitrary; -Section 4.11.1: Updated commands to erase flash partitions; -Section 4.12: Added note that section is only relevant to kernel version 2.6.x; -Section 4.13: Added note that section is only relevant to kernel version 3.x; -Added Section 4.13.1.1; -Section 4.17.2: Updated commands in Step 2 and Step 3; -Added Section 4.26; -Section 4.27: Added note about importance of using proper commands to shut down the development kit; -Section 4.30: Reorganized for clarity; -Added Section 6.2; -Section 8: Added instructions for camera use; added prerequisite that default build configuration must be completed before proceeding; -Section 8.1: Added Steps 44.b through 44.j; added note that YAFFS image type is required; added note about package dependencies; -Added Section 8.3 and 10 		
E	BSB, SO	<ul style="list-style-type: none"> -Throughout: Corrected dashes in commands to hyphens; updated template; updated links for new support site; -Section 2.5.1: Added command for X-Loader; -Added Section 2.5.4 about creating cross-compilation shell; -Added Section 2.5.7 about reviewing selected packages; -Section 3.2: Updated for new default display 28; -Section 3.2.2.1: Added vertical front porch (vfp) to display environment variables; -Section 3.2.9: Added reference to <i>DM3730/AM3703 U-Boot Labs</i> document; -Section 3.2.14: Added instructions to convert DocBook XML file to HTML file; -Added Section 4.2 about how to retrieve BSP version information; -Added Section 4.3 about how to display product ID system information; -Added Section 4.4 about how to display Linux system information; -Added Section 0 about how to display text messages; -Added Section 4.15 about using the USB controller; -Added Section 4.23 about the BQ27000 gas gauge; -Added Section 4.29 about adding CPU bookmarks; -Added Section 4.30 about using <i>peekpoke</i> command; -Added Section 4.31 about useful filesystem commands; -Added Section 10.3 about reporting problems 	RAH, PB, SO	10/11/13

F	BSB	<ul style="list-style-type: none"> -Added Section 3.2.11 regarding booting with X-Loader, U-Boot, the kernel, and root filesystem on an SD card; -Added Section 3.2.14 regarding how to move the debug console from UARTA to UARTB; -Section 4.7.3: Updated to include instructions about how to handle the console backlight blanking; Added Section 4.32 about booting the kernel from a TFTP server and booting the root filesystem from an NFS server; -Section 4.8: Added audio record, alsamixer and audio path information -Section 8.1: Added note to avoid setting C6x_C_DIR; -Added Section 10.3 regarding removing drivers; -Added Section 11 regarding basic driver/kernel debugging 	SO, AF, AM	08/26/14
G	BSB, AF	<ul style="list-style-type: none"> -Updated Section 2.4 Build -Added Section 2.7.1.1 Using <i>mkLogicFATcard.sh</i> to create an SD Card with a RAM Disk Image -Added 2.7.1.2 Using <i>mkLogicFATcard.sh</i> to create an SD Card with a YAFFS rootfs for NAND -Added Section 2.8 CodeSourcery 2011.09 Compiler -Added Section 3.2.7 Printing Text to the Display -Updated section 8.3.3 to change recommended USB mini-B plug to USB mini-A plug. -Added Section 3.2.7 Printing Text to the Display -Updated Section 4.2 Retrieve BSP Version -Updated Section 4.8 Audio -Updated Section 4.13 Wireless Networking with Linux 3.x Kernels -Added Section 4.13.3 Start Wireless Interface in Multi-Role -Added Section 4.13.4 Setting Regulatory Domains Using the CRDA (Central Regulatory Domain Agent) -Added Section 4.24 Smart Reflex -Updated Section 6.2.3 Run GPS Demo Application -Updated Section 8 Cameras and DSP (DVSDK) -Updated Section 8.1 Prepare Build Tools -Updated Section 8.3.3 USB Webcam Example -Added Section 13 Appendix A: Enable MCS0 and MCS7 	JMC, BSB, AF	8/28/15
H	AF, BSB	<ul style="list-style-type: none"> -Updated Section 3.2.2 Configure Boot -Updated Section 3.2.2.1: Add info on how to disable video -Added Section 3.2.12 Boot with Read-Only Root File system -Updated Section 4.11.2 Mount NOR Flash using JFFS2 -Added Section 4.11.3 Mount NAND Flash using YFFS2 -Added section 4.32 Using Linux Voltage and Current Regulator Framework -Added section 4.33 Reading Temperature Sensor -Delete the contents of section 4.12 and replace it with a note indicating that Kernel 2.6x support is no longer supported and users should upgrade to 3.0 kernels. 	AF, CT	12/09/2016

Table of Contents

1	Introduction	1
1.1	Nomenclature.....	1
1.2	Development Resources.....	1
1.3	The BSP Design	1
1.4	Prerequisites	2
1.5	Precautionary Statement	2
1.6	Paths to Linux Platform Development	2
1.7	Timesys Partnership	3
1.8	Recreating DM37x Linux Demo SD Card.....	3
1.9	Additional Information	3
2	Build Source	4
2.1	Virtual Machine.....	4
2.2	Obtain Build Environment and Source.....	5
2.2.1	Patches	5
2.3	Prepare Host PC.....	6
2.3.1	Required Host PC Packages	6
2.4	Build	8
2.4.1	Files Available after Build	8
2.4.2	Place Final Images on SOM.....	9
2.5	Common LTIB Commands	9
2.5.1	Source Code for Specific Packages.....	9
2.5.2	Build Changes and Create Output Files.....	9
2.5.3	Reset to Fresh Build Environment	9
2.5.4	Create Cross-Compilation Shell	9
2.5.5	Configure BSP.....	10
2.5.6	The <i>.config</i> File	14
2.5.7	Review Selected Packages	15
2.6	Review BSP Configuration without LTIB Installed	15
2.7	Other Tools	15
2.7.1	LTIB <i>/bin</i> Directory	15
2.7.2	<i>/opt/ltib/usr/bin</i> Directory	16
2.7.3	<i>/opt/CodeSourcery</i> Directory	16
2.8	CodeSourcery 2011.09 Compiler.....	16
3	Boot Configuration	18
3.1	Boot Sequence	18
3.2	U-Boot.....	18
3.2.1	Get Started	19
3.2.2	Configure Boot.....	19
3.2.3	U-Boot <i>help</i> Command	22
3.2.4	U-Boot Environment	23
3.2.5	Shell Variables	24
3.2.6	Display Splash Screen.....	25
3.2.7	Printing Text to the Display	26
3.2.8	Script with Variables.....	26
3.2.9	Script from Memory	28
3.2.10	Boot from NAND Flash	28
3.2.11	Boot with X-Loader, U-Boot, Kernel, and Root Filesystem on SD Card	37
3.2.12	Boot with Read-Only Root File system.....	38
3.2.13	Useful Scripts	40
3.2.14	Debug UART	41
4	Kernel.....	44
4.1	vi Editor.....	44
4.2	Retrieve BSP Version	45
4.3	Display Product ID System Information	45
4.4	Display Linux System Information.....	45
4.5	Wired Networking	46
4.5.1	Assign Development Kit IP Address	46

4.5.2	Set Speed, Duplex, and Auto-Negotiate	48
4.5.3	Test Network	49
4.6	Linux Processes	49
4.6.1	<i>ps</i> Command	49
4.6.2	<i>kill</i> Command.....	50
4.7	Video Display	51
4.7.1	Draw Test	51
4.7.2	DirectFB.....	51
4.7.3	Backlight.....	52
4.7.4	Display Message.....	53
4.8	Audio	53
4.9	External Memory Interface.....	55
4.10	Touch Screen	55
4.11	Built-in Flash Storage via MTD	55
4.11.1	Erase Flash Partitions.....	56
4.11.2	Mount NOR Flash using JFFS2	56
4.11.3	Mount NAND Flash using YFFS2.....	57
4.12	Wireless Networking with Linux 2.6x Kernels.....	57
4.13	Wireless Networking with Linux 3.x Kernels.....	57
4.13.1	Start Wireless Interface in Station Mode.....	57
4.13.2	Start Wireless Interface in AP Mode	58
4.13.3	Start Wireless Interface in Multi-Role	60
4.13.4	Setting Regulatory Domains Using the CRDA (Central Regulatory Domain Agent)	61
4.14	Bluetooth	61
4.14.1	Start or Stop Bluetooth Interface	62
4.14.2	Assign Hardware Name	63
4.14.3	View Bluetooth Device Configuration.....	63
4.14.4	Modify Bluetooth Device Configuration	63
4.14.5	Scan for Bluetooth Devices.....	63
4.14.6	Query Bluetooth Device.....	63
4.15	USB Controller.....	63
4.16	USB Host Controller	64
4.17	Processor OTG Controller	64
4.17.1	Use MUSB in Host Mode	64
4.17.2	Use MUSB in Device Mode	64
4.18	UART.....	65
4.19	I2C	66
4.20	SPI	66
4.21	Real Time Clock	66
4.22	Analog-to-digital Converters.....	67
4.23	BQ27000 Gas Gauge Support	68
4.24	Smart Reflex	69
4.25	Run/Idle/Suspend	70
4.26	Virtual Files.....	70
4.26.1	<i>echo</i> Command and ">" Operator.....	70
4.26.2	<i>/sys/kernel/debug</i> Directory	71
4.27	Shut Down Linux System	72
4.28	Additional Peripheral Test Information	72
4.29	CPU Benchmarks	72
4.30	Use <i>Peekpoke</i> to Examine/Modify Registers.....	75
4.31	Filesystem Commands.....	75
4.31.1	<i>df</i> Command.....	75
4.31.2	<i>cat /proc/mtd</i> Command	76
4.31.3	<i>flash_eraseall</i> Command	76
4.31.4	<i>badblocks</i> Command.....	76
4.32	Using Linux Voltage and Current Regulator Framework.....	76
4.33	Reading Temperature Sensor	78
5	Boot Kernel from TFTP Server and Root Filesystem from NFS Server	79
5.1	Set Up TFTP Server in Ubuntu 12.04	79
5.2	Setup NFS Server in Ubuntu 12.04.....	80

5.3	Set Up the DM3730/AM3703 Target Platform	81
6	Application Development.....	83
6.1	"Hello World" Application Example	83
6.1.1	Build "Hello World" Application	83
6.1.2	Transfer "Hello World" Application to Root Filesystem	83
6.1.3	Run "Hello World" Application	84
6.2	GPS Demo Application Example	85
6.2.1	Build GPS Demo Application.....	85
6.2.2	Transfer GPS Demo Application to Target	85
6.2.3	Run GPS Demo Application	85
6.3	Debug with GNU Debugger.....	87
6.3.1	Configure Build Options.....	87
6.3.2	Set Up GDB	87
7	Loadable Module Development	90
7.1	"Hello World" Module.....	90
7.1.1	Build "Hello World" Module	90
7.1.2	Run "Hello World" Module.....	90
8	Cameras and DSP (DVSDK)	92
8.1	Prepare Build Tools	92
8.2	Run Time Configuration	98
8.3	Example DSP and Camera Use.....	99
8.3.1	DSP Example	99
8.3.2	Parallel Camera Example.....	100
8.3.3	USB Webcam Example	102
9	GTK Demo.....	104
9.1	Prepare Build Tools	104
9.2	Build	104
9.3	Run Demo.....	104
10	Customize LTIB	106
10.1	Definitions	106
10.2	Integrate New Package.....	106
10.2.1	Integrate New Service	107
10.2.2	Build.....	109
10.2.3	Installing.....	109
10.2.4	Test.....	109
10.2.5	Configure	109
10.3	Removing Drivers	110
10.3.1	General Instructions	110
10.3.2	Remove Specific Interfaces.....	112
11	Basic Driver/Kernel Debugging Information	115
11.1	<i>dmesg</i>	115
11.2	Dynamic Debug	115
11.3	Enable Specific Debug Message	116
11.4	Debug Modules.....	117
12	Reporting Problems.....	118
13	Appendix A: Enable MCS0 and MCS7.....	119

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, LTIB, 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 custom code that Logic PD has implemented for the BSP. Linux, U-Boot, LTIB, 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](http://www.logicpd.com/support/)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](http://www.logicpd.com/support/support-packages/)² for additional information.

1.3 The BSP Design

Logic PD constantly reviews the latest Linux kernels to identify versions that support features required by our customers and chooses the kernels that are most stable. When a kernel version is chosen, a BSP is then built around it. Logic PD is always balancing the need to have

¹ <http://www.logicpd.com/support/>

² <http://www.logicpd.com/support/support-packages/>

the latest Linux kernel with the need for high stability. When those two ideas do not align, Logic PD will opt for the kernel with the greatest stability.

1.4 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.

Logic PD provides four different ways to modify and build the Linux BSP:

1. Build the latest BSP source using your preferred build tool. The DM37x Linux BSP download is available on Logic PD's website (see Section 2 for a link) and includes all source code for X-Loader, U-Boot, the Linux kernel, and the Linux Target Image Builder (LTIB) build tool. See the *README-setup* file within the tar ball for the specific location of the source. This option allows you to work in a familiar build environment, although Logic PD may not be able to provide assistance for build issues. Please see Section 2.2 for additional information on how to download the source. See Section 2.5.7 regarding BSP configuration.
2. Build the latest BSP using the Logic PD-recommended build tool. The DM37x Linux BSP download is available on Logic PD's website (see Section 2 for a link) and includes all source code for X-Loader, U-Boot, the Linux kernel, and the LTIB build tool. This option provides the fastest build time, but requires some setup on the host PC. Please see Section 2.3 for additional information on how to configure your Linux PC.
3. Build the latest BSP through a pre-configured build environment with source using a virtual machine (VM). The Virtual Machine SDK for the DM37x Linux BSP is available on Logic PD's website (see Section 2.1 for a link) and is configured to use VirtualBox as the VM manager. The VM includes source for X-Loader, U-Boot, the Linux kernel, the LTIB build tool, and a pre-configured Ubuntu operating system (OS). This option is the simplest way to get started building on nearly any OS. Because building occurs on a VM, the build is slightly slower than building on a native Linux-based PC. Please see Section 2.1 for additional information on how to build using a VM.

³ <http://support.logicpd.com/TechnicalSupport/RegisterProduct.aspx>

4. Download the latest BSP from the Timesys website. This includes source for X-Loader, U-Boot, the Linux kernel, and the Timesys build tools. This option provides the most versatile set of tools and optional Timesys support. Please see Section 1.7 for additional information about how to get started using the Linux Timesys tools.

Any of the methods noted above can be used to update, build, and debug the Linux BSP, although each method has different requirements. Please post a question to the Logic PD [Technical Discussion Group \(TDG\) forum](#)⁴ if you need help deciding which build method is right for you.

1.7 Timesys Partnership

Logic PD is very pleased to be partnered with Timesys. Logic PD supplies Timesys with our Linux BSP and works with them to integrate it into their excellent tools (LinuxLink and Factory). Logic PD and Timesys work together to ensure that the BSP and related Linux environment stays current and that mutual customers get the best support possible. In many cases, Logic PD customers can access a free trial of the Timesys tools. Please contact your Timesys or Logic PD sales person for additional information. Also, see the dedicated [Timesys/Logic PD landing page](#).⁵

1.8 Recreating DM37x Linux Demo SD Card

For information on how to recreate the DM37x Linux Demo SD card included in the DM3730 Development Kit, please see the [DM37x Linux SD Card Demo Image download](#).⁶ The download includes the necessary software files. The [DM37x Linux SD Card Demo Image ReadMe](#)⁷ provides instructions for completing the process.

1.9 Additional Information

There are several files in the DM37x Linux BSP download that will have additional information. Some of these documents are supplied by the build tool in the *doc* folder. Other *README* files are in the root of the BSP and contain the very latest version-specific information for the BSP.

- The *doc* folder in the BSP: This folder is provided by the LTIB build tool. There are several files here that contain a wealth of information about LTIB and how to use it. In particular, note the *doc/LtibFaq* file that contains a full description of all command line arguments supported by LTIB.
- *README*: LTIB introduction.
- *README-setup*: Location of source files and a full description of how to set up LTIB on a PC.
- *README-LTIB-shortcuts*: LTIB quick reference information.
- *RELEASE_INFO*: Release version information; this file is generated after the first build is complete.

⁴ <http://support.logicpd.com/TDGForum.aspx>

⁵ <http://www.timesys.com/supported/boards/logic>

⁶ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1405>

⁷ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1409>

2 Build Source

This section describes how to build X-Loader, U-Boot, the kernel, and the RAM-based Linux root filesystem using LTIB. The Linux image provided in Logic PD's [DM37x Linux BSP download](#)⁸ consists of a single .tar.bz2 file that contains X-Loader source, U-Boot source, kernel source, and LTIB.

Internally, Logic PD develops its Linux BSP and systems using an LTIB-based environment. LTIB is an open-source tool used to develop and deploy BSPs for various target platforms. See the [LTIB website](#)⁹ for more information. LTIB preforms the following tasks:

- Installs the cross compiler. The cross compiler, runs on a desktop 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 OS from source. LTIB 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. LTIB can be configured to add or remove a long list of included packages. Each package can have binaries and/or source. LTIB 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. LTIB 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. LTIB will create an image of the root filesystem that users can install on their hardware.
- Aids in development. LTIB 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. LTIB allows the developer to extract, modify, build, and deploy any single package.

For customers who do not wish to use LTIB, please see Section 2.2.

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, LTIB, build tools, and source. The

⁸ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=853>

⁹ <http://www.ltib.org>

[Virtual Machine SDK for the DM37x Linux BSP](#)¹⁰ has all building capabilities on the host PC as described in Section 2.4 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 manager. See the [Virtual Machine SDK for the DM37x Linux BSP ReadMe](#)¹¹ for installation instructions.

If you choose to use the VM, building occurs just like building on a native host. See Section 2.4 to continue building the BSP.

2.2 Obtain Build Environment and Source

Begin by downloading the DM37x Linux BSP tar ball to your Linux machine from the Logic PD website (see Section 2 for a link). Once downloaded, choose a directory to which you would like to extract the tar ball. Use the following command to extract the tar ball, where *filename* is the filename of the downloaded tar ball.

```
tar xvjf filename.tar.bz2
```

Be sure to review each of the *README* files within the tar ball for the latest information on the DM37x Linux BSP. The examples in this section are examples only and may not provide the most up-to-date information for your release.

For those customers who are familiar with Linux and wish to use a build tool other than LTIB, the source code within the .tar.bz2 file can be used with some other Linux build packages. See the *README-setup* file within the tar ball to see where the source files are located. Building the provided source with a build tool other than LTIB, however, is beyond the scope of this document.

2.2.1 Patches

A patch file is simply a delta between two text files that is created by running the Linux *diff* command. The output of the command is then saved as a patch file that can be used when a developer wishes to document only the changes made (since the patch file itself is a human readable text-based file). It can also be used when a developer wishes to disseminate a small file in place of the entire updated code base. Use the *diff -help* command in Linux for more information.

Applying the patch is achieved using the Linux *patch* command. The *patch* command will read the differences listed in the patch file and update the lesser (older) file to be the same as the newer file where the diff was created. Use the *patch -help* command in Linux for more information.

Logic PD will, at times, release patches to the original BSP. Those patches can be found in the DM37x Linux BSP tar ball *LPD-IP-package-pool/* directory. Typically patches Logic PD provides to the original BSP source will be named using the following format:

`<source><version>-BSP-<BSP version>-<patch name>.patch`

The `<source>` will often be X-Loader, U-Boot, or Linux. The `<version>` will be the version for the indicated source. The `<BSP version>` is the BSP version of the patch. The `<patch name>` is the name of the patch. For example, the environment patch for U-Boot BSP 1.0 is:

`u-boot-2009.11-BSP-1.0-environment.patch`

¹⁰ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=858>

¹¹ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1401>

When a patch file is included in the package spec file, LTIB will automatically apply the patch upon building, prepping, or deploying the package. See Section 10.2 for additional information about creating a spec file.

2.3 Prepare Host PC

LTIB is designed to be executed on a Linux host PC. The following steps should work for any Linux distribution; however, Logic PD can only confirm operation with the OS version indicated in the *README-setup* file included in the DM37x Linux BSP tar ball. Please note that the operations included below are examples only; see the *README-setup* file for the latest information regarding your version of the BSP and for additional information on using LTIB.

It is assumed that the user has properly configured the */etc/sudoers* file. If not, please refer to "Section 1" of the *README-setup* file for information on configuring the */etc/sudoers* file.

The tar ball now includes a script that can be run to install all the needed packages automatically. Please refer to the *README-setup* for more information about using the script in lieu of the manual package installation steps listed in this section.

2.3.1 Required Host PC Packages

The following packages are generally needed by LTIB and some of the standard components that Logic PD releases. 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 LTIB 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.

NOTE: The information presented below provides an example set of tasks needed to set up LTIB. For a complete detailed list of tasks for your specific version of the DM37x Linux BSP, see the *README-setup* file in the tar ball.

NOTE: Newer versions of the BSP now include a script named *ltib_setup.sh*. This script can be run in lieu of the steps below to automate your system setup. Run the script by issuing the commands below in the LTIB directory. Be sure to replace *<your LTIB directory>* with the location where you extracted the DM37x Linux BSP tar ball in Section 2.2.

```
bash$ cd <your LTIB directory>
bash$ ./ltib_setup.sh
```

Once the script is run, follow the prompts.

2.3.1.1 Switch to Bash Instead of Dash

At the Linux command prompt, enter the following command:

```
bash$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 2011-01-05 01:54 /bin/sh -> bash
```

If you see */bin/sh -> dash* instead of */bin/sh -> bash*, then execute the following command:

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

When prompted if you want dash as `/bin/sh`, select NO.

2.3.1.2 Install Packages

LTIB and the build tools require several packages on your host PC. The following list is an example of how to install the packages with example commands to install them. Please see the *README-setup* file in the downloaded DM37x Linux BSP tar ball for the latest list of packages and any additional information needed to install your specific BSP version.

```
bash$ sudo apt-get install build-essential
bash$ sudo apt-get install nfs-kernel-server
bash$ sudo apt-get install nfs-common
bash$ sudo apt-get install portmap
bash$ sudo apt-get install tftpd-hpa
bash$ sudo apt-get install tftp
bash$ sudo apt-get install rpm
bash$ sudo apt-get install wget
bash$ sudo apt-get install bison
bash$ sudo apt-get install flex
bash$ sudo apt-get install 'zlib*'
bash$ sudo apt-get install libncurses5
bash$ sudo apt-get install libncurses5-dev
bash$ sudo apt-get install libjpeg-dev
bash$ sudo apt-get install libx11-dev
bash$ sudo apt-get install xutils-dev
bash$ sudo apt-get install tcl
bash$ sudo apt-get install gettext
bash$ sudo apt-get install uuid-dev
bash$ sudo apt-get install libxext-dev
bash$ sudo apt-get install libtool
```

For 64-bit host PCs, run the following command in addition to those noted above:

```
bash$ sudo apt-get install ia32-libs
```

2.3.1.3 Perl Packages

Perl needs the LWP:UserAgent package. To install this, become root and enter a CPAN shell.

```
bash$ sudo sh
bash# perl -MCPAN -eshell
cpan[1]> install LWP::UserAgent
cpan[2]> exit
bash# exit
bash$ whoami
```

Verify that your user name is printed and that you are no longer root.

NOTE: The LWP:UserAgent package can also be installed via *sudo apt-get install libwww-perl* rather than *perl -MCPAN -eshell*, although either should work.

NOTE: In some Linux distributions, the *sudo su* command may be required in place of *su <root password>*, as indicated in the example above.

2.4 Build

The build is performed the same way, whether it is being built on a VM or the native host. The VM comes complete with the source tar ball already expanded. However, you may opt to download the latest tar ball from the [Logic PD support site](#).¹² The same tar ball available there can be used on the native host or the VM.

NOTE: When using LTIB, disregard the "error: failed to stat /home/<user name>/.gvfs: Permission denied" message. It is a message from the GNOME Virtual File System (GVFS) and is a known bug in GNOME. See this [GVFS wiki article](#)¹³ and this [DENX wiki article](#)¹⁴ for additional information.

With the tar ball expanded on your host PC and all the necessary packages installed (see the previous sections for requirements), the build can be started from within the expanded directory with the following command:

```
bash$ ./ltib -b --preconfig config/platform/omap_logic/defconfig
```

The initial build may take as long as twenty minutes on a 3.4 GHz Intel I7 quad with 16 GB of RAM. It may take more or less time, depending on the performance of your host PC. During the initial build, all the source code is expanded and all files are built. Successive builds will be much faster.

NOTE: If configured the kernel to use the omap3logic_defconfig-performance on the Torpedo + Wireless, MCS0 and MCS7 may not be available for Wifi operation. See Appendix A on how to enable MCS0 and MCS7 with omap3logic_defconfig-performance on the Torpedo + Wireless.

2.4.1 Files Available after Build

Upon completion of the build, the following files should be available:

- *rootfs.ext2.gz.uboot* – RAM-based root filesystem for loading from U-Boot
- *rootfs/boot/u-boot.bin* – U-Boot for use on SD/MMC
- *rootfs/boot/u-boot.bin.ift* – U-Boot for use on NAND flash
- *rootfs/boot/uImage* – Kernel image
- *rootfs/boot/MLO* – X-Loader

NOTE: If the build is configured for a NAND-based YAFFS filesystem, the *rootfs.yaffs2* file will be produced instead of *rootfs.ext2.gz.uboot*. See Section 2.5.5.1 for more information.

¹² <http://support.logicpd.com/Home.aspx>

¹³ <https://bugs.launchpad.net/gvfs/+bug/225361>

¹⁴ <http://www.denx.de/wiki/DULG/ELDKGvfsPermissionDenied>

2.4.2 Place Final Images on SOM

See *mkLogicFATcard.sh* in Section 2.7.1 for information on how to make a bootable SD card and copy images.

See Section 3.2.13.1 for the *makenandboot* utility to place images in raw NAND flash.

See section 3.2.13.2 for the *makeyaffsboot* utility to place images in a YAFFS filesystem on NAND flash.

2.5 Common LTIB Commands

2.5.1 Source Code for Specific Packages

These commands will extract the source of the package, apply any patches, and place it in *rpm/BUILD*. From here, the source can be modified as needed. Below are some examples.

```
bash$ ./ltib -p kernel -m prep
bash$ ./ltib -p x-loader -m prep
bash$ ./ltib -p u-boot -m prep
bash$ ./ltib -p spi-test -m prep
bash$ ./ltib -p draw-test -m prep
```

2.5.2 Build Changes and Create Output Files

Once you have modified the source code to your needs, you can use this command to compile it and deploy the appropriate files.

```
bash$ ./ltib -p kernel -m scbuild && ./ltib -p kernel -m scdeploy
```

2.5.3 Reset to Fresh Build Environment

This command will reset the LTIB environment to the default.

```
bash$ ./ltib -b -m distclean
```

2.5.4 Create Cross-Compilation Shell

The command below will create a shell with an environment equivalent to the environment used to build packages. This includes all the variables and spoofing necessary for cross-compilation. It is useful when you want to make changes to a package source and build it directly instead of using *./ltib*.

```
bash$ ./ltib -m shell
```

Once the changes are verified to build correctly, exit this shell and build using *./ltib* instead.

2.5.5 Configure BSP

The drivers, packages, X-Loader, U-Boot, build tools, and the output binaries for the BSP are configured with a file located in the LTIB source tree named *config*. This file is created the first time LTIB builds the source.

After your first build, the *config* file can be customized using a menu system started with the command below.

```
bash$ ./ltib -c
```

This *config* menu system is modeled after the existing Linux kernel configuration system. If you wish to make changes specific to the kernel, you can get to the kernel configuration system by selecting "Configure the kernel" from within the LTIB menu system.

You can use the following command to pull up the kernel configuration system directly and make your changes. LTIB will rebuild any packages dependent on the kernel (i.e., any that create modules).

```
bash$ ./ltib -p kernel -c
```

2.5.5.1 Example: Create YAFFS Root Filesystem Image

Booting with a RAMdisk-based root filesystem provides a repeatable system start at every boot. Since RAM is volatile, a RAM-based filesystem is lost and must be reloaded at every boot. This results in the same root filesystem every time the system is restarted. However, there may be cases in which you wish the root filesystem to exist in non-volatile memory so that any changes to the root filesystem are retained across power cycles.

LTIB can be configured to produce a NAND flash-based root filesystem, which will be non-volatile. To do this, start the LTIB menu configuration system and follow the steps below.

1. At the bash prompt, start the LTIB menu system.

```
bash$ ./ltib --c
```

2. Under the *Target Image Generation* menu heading, select Options.

```

eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

LTIB: Logic OMAP3530/03&DM3730/03 reference boards
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selectes a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded
^(-)
[ ] Produce cscope index
(omap3logic_defconfig) kernel preconfig
--- Include kernel headers
[*] use 'make headers_install' to install kernel headers
[ ] Configure the kernel
--- Leave the sources after building
(ericn) S.M username to replace '%s' in REPOSITORY variables
--- Package selection
    Package list --->
    Target System Configuration
    Options --->
    Target Image Generation
    Options --->
(BSP-dm37x-2.0-trunk) BSP Release Level (supplied by platform defconfig)
---
Load an Alternate Configuration File
v(+)

<Select> < Exit > < Help >

```

3. Under the *Choose your root filesystem image type* menu heading, choose Target image.

```

eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

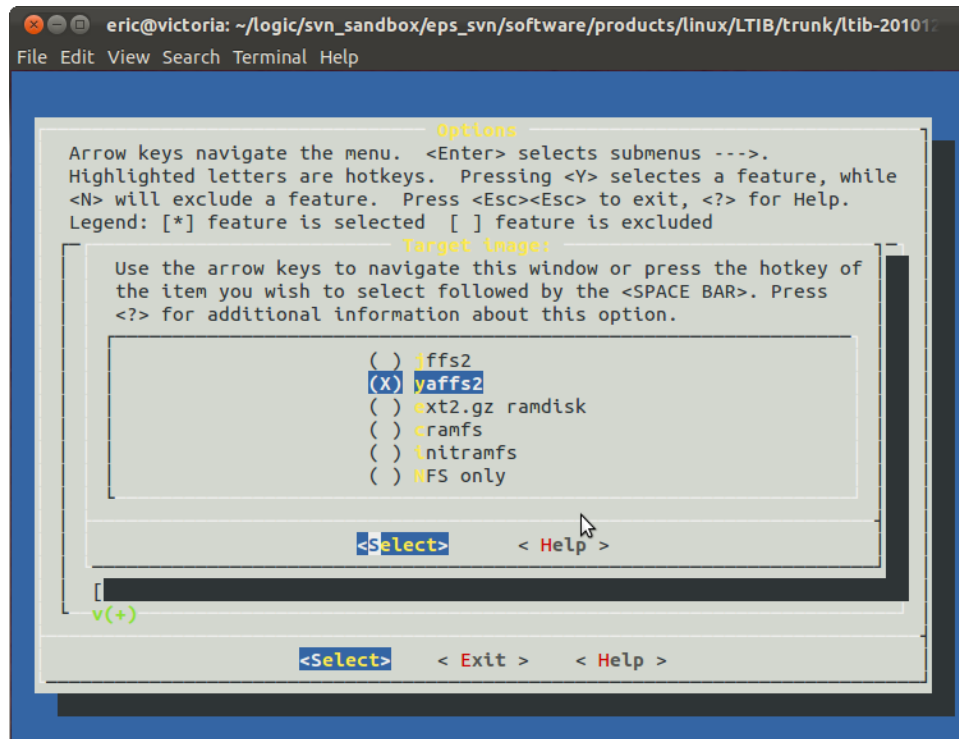
Options
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selectes a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded

--- Choose your root filesystem image type
Target image: (ext2.gz ramdisk) --->
[ ] Create a combined ELF image
[ ] Run a command after building the target image
[ ] read-only root filesystem
[*] mount debugfs on /debug during startup
[*] create a ramdisk or initramfs image that can be used by u-boot
() rootfs target directory
[*] keep temporary staging directory
[ ] Convert hard links to symbolic links
[*] remove man pages etc from the target image
[*] remove the /boot directory
[*] remove the /usr/src/ directory
[*] remove the /usr/include directory
[*] remove the /usr/share/locale directory
() remove these directories
v(+)

<Select> < Exit > < Help >

```

4. Next, select yaffs2.



5. Finally, save your changes by exiting each sub menu. Wait for LTIB to build your images.

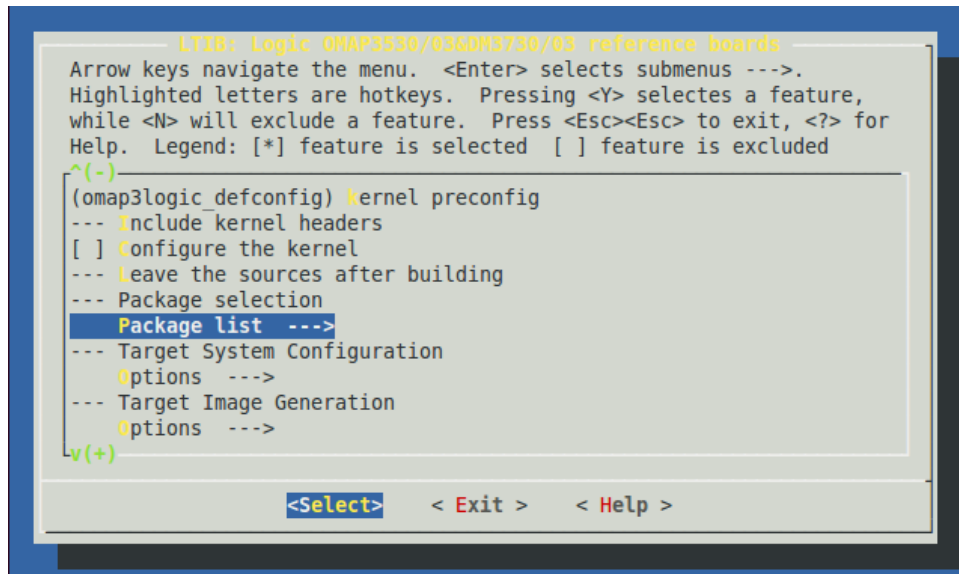
2.5.5.2 Example: Include GTK+ and Liberation Fonts Packages in Build

Additional configuration may be required to build and run GTK+ and the Liberation fonts. The example below merely shows how to include packages in your build if they are not present by default.

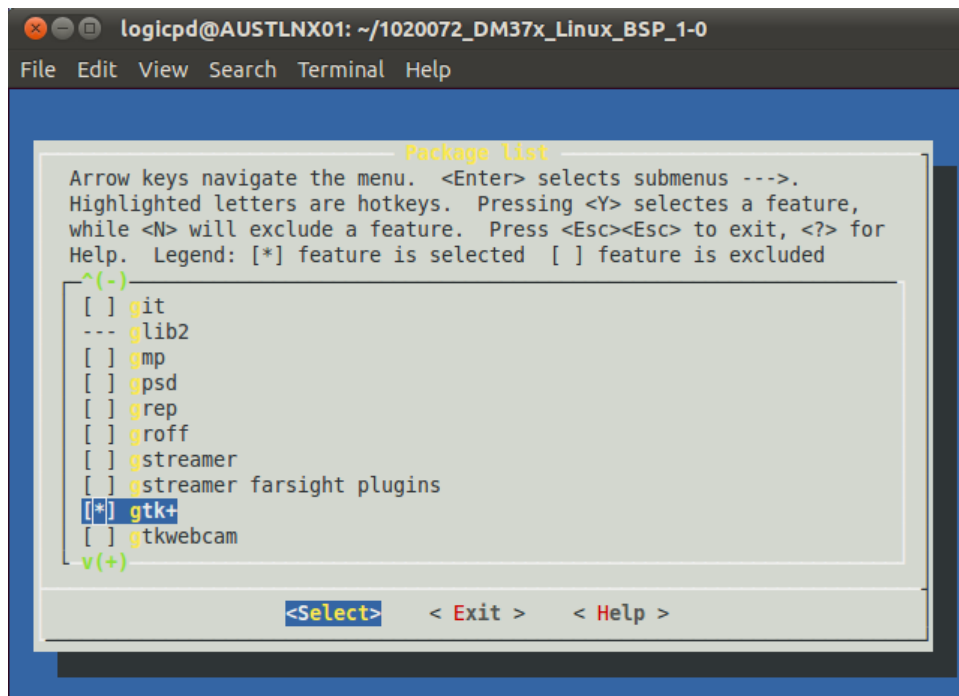
1. Start the LTIB menu system.

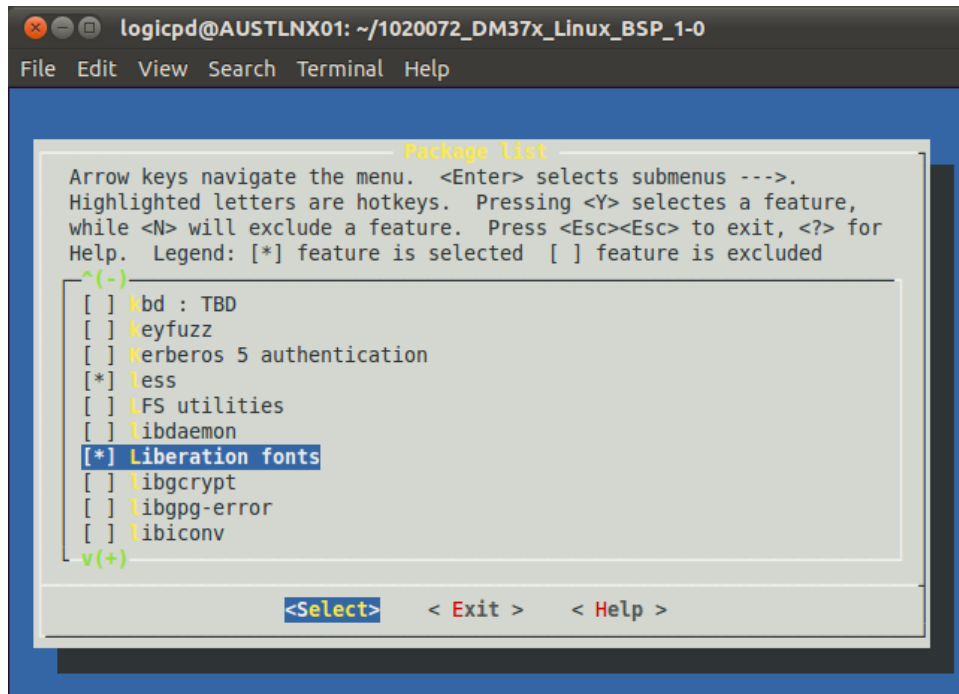
```
bash$ ./ltib -c
```

2. Select the *Package List* menu item to expand it.



3. Add the GTK+ and Liberation fonts packages by pressing the **Enter** key to navigate down the submenus. Press the **Y** key to select the item.





4. Exit the LTIB menu by pressing the **Esc** key to navigate back up the submenus.

After you exit the LTIB menu, the system will build the added packages into your newly created Linux image with GTK+ support.

2.5.6 The *.config* File

There may be cases where the developer doesn't want to use the graphical configuration tool (*./ltib --c*). In those cases, the text-based *.config* file contains the same configuration information and can be viewed and/or edited with a text editor.

Furthermore, when using the graphical editor, you can highlight an option and choose "help" at the bottom of the screen to display the *config* macro name and value used in the text-based *.config* file. In this way, a developer can correlate items updated in the graphical interface with items in the *.config* file. Note that when editing the *.config* file with a text editor, there is no check performed to ensure the configuration combinations are correct.

Upon building the BSP for the first time, a command similar to the one below will cause LTIB to create a file called *.config*. This file will contain first-level configuration information, such as the platform of the first build and the location of the full configuration file.

```
bash$ ./ltib -b --preconfig config/platform/omap_logic/defconfig
```

The above build command indicates that the *.config* file in the LTIB root contains the following:

```
CONFIG PLATFORM omap logic=y
CONFIG PLATFORM DIR="config/platform/omap_logic"
```

Here, we see the full configuration file is located in *config/platform/omap_logic/.config*. The file name in this location (*.config*) is implied.

When the default build command is used (see Section 2.4), the configuration file specified in the build command is copied to a file named *.config*. Thereafter, any changes to the configuration are maintained in the *.config* file.

2.5.7 Review Selected Packages

To list packages that are currently enabled, use the command below.

```
./ltib -m listpkgs | grep ' y '
```

2.6 Review BSP Configuration without LTIB Installed

There may be cases where a developer wants to know the default configuration of the BSP as configured with LTIB. For example, a developer may not wish to use LTIB, but needs to configure the preferred build tool. In these cases, the developer can look in the default configuration file with a text editor for all the configuration information.

2.7 Other Tools

In addition to LTIB, there are other tools included in the BSP to help make the developers life easier.

2.7.1 LTIB */bin* Directory

The */bin* directory in the LTIB install folder includes LTIB components and the *mkLogicFATcard.sh* file, which is used to format an SD card suitable for SOM booting. Use this file to partition, format, and copy boot files to your SD boot card. For additional information on creating a bootable SD card, see the [DM37x Linux SD Card Demo Image ReadMe](#)¹⁵ for a step-by-step process.

2.7.1.1 Using *mkLogicFATcard.sh* to create an SD Card with a RAM Disk Image

The *mklogicFATcard.sh* script with the *-c* option will allow developers to format a SD Card and copy the following files: MLO, u-boot.bin, uImage, and rootfs.ext2.gz.uboot to the sd card.

```
$ ./bin/mkLogicFATcard.sh -c [dev]
```

2.7.1.2 Using *mkLogicFATcard.sh* to create an SD Card with a YAFFS rootfs for NAND

The *mklogicFATcard.sh* script with the *-cy* option will allow developers to format a SD Card and copy the following files: MLO, u-boot.bin, uImage, and rootfs.yaffs to the sd card.

```
$ ./bin/mkLogicFATcard.sh -cy [dev]
```

¹⁵ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1409>

Note: Execute the *mklogicFATcard.sh* script from the LTIB root directory. In the command lines above replace [dev] with the assigned device you're your SD Card on your host PC. If no [dev] is specified a list of available devices will be presented.

2.7.2 /opt/ltib/usr/bin Directory

When LTIB performs the first build, it installs the appropriate cross compiler and an assortment of tools needed for building. The */opt/ltib/usr/bin* directory has an assortment of tools the developer may find handy. Information regarding most of the tools can be obtained by adding *--help* as the first argument when running the tool. The list of tools is quite large, so only a few key tools are noted in the list below.

- *mkfs.yaffs2* – A file used to create a YAFFS2 filesystem image. This image can be loaded into NAND flash using U-Boot's *write.yaffs* command.
- *ccache* – A compiler cache used to cache files when compiling to speed recompilation.
- *bmp_logo* – A tool to convert a BMP file to a header file. It is often used to link in a default splash screen in U-Boot.

2.7.3 /opt/CodeSourcery Directory

The CodeSourcery cross compiler is used to build the source code. This is a GNU-based tool set that includes a compiler, assembler, linker, debugger, and other standard GNU tools used for development. When LTIB performs its first build, the CodeSourcery tool set is installed in the directory */opt/CodeSourcery*.

2.8 CodeSourcery 2011.09 Compiler

The default cross compiler is the CodeSourcery-2009q1-203 gcc-4.3.3. Customers can change the default cross compiler in the LTIB menu to CodeSourcery-2011-09-70 gcc 4.6.1 as seen in the Figure 1.

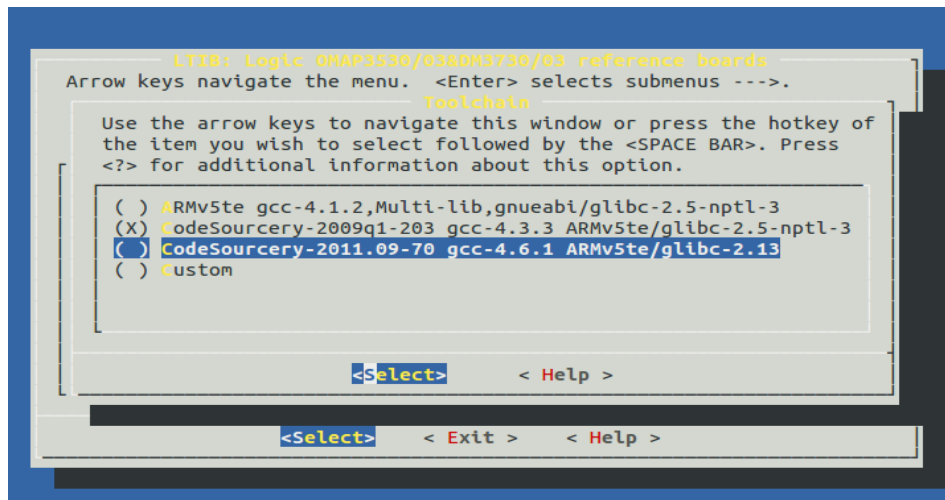


Figure 1: Compiler Options

Starting with release 2.3-0, the BSP added the newer compiler "CodeSourcery-2011.09-70 gcc-4.6.1 ARMv5te/glibc-2.13". This new compiler was found to be incompatible with packages such as Qtopia and DVSDK. In those cases, the build will fail. Since the 2.4-4 release build

issues with Qtopia and DVSDK has been resolved. The new compiler employs additional optimizations that produce faster, more efficient run-time code.

When changing between the 2009 to 2011 compilers in LTIB, build failures have been known to occur. The ".ltib -m distclean" instruction has not been sufficient to clean the build for use with a different compiler. The problem is some packages do not implement the "distclean", or the "clean" stanzas or don't implement them very well (openssl, perl, ...).

Developers should remove the "rootfs" and "rpm" directories from within the LTIB folder prior to changing the cross compiler and rebuilding. It will clear out any build data from the previous compile, making building with the new compiler succeed.

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

Bootting 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 X-Loader. 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.
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.

There is a choice where the second and third-stage bootloaders are stored, as long as they are both stored in the same location (e.g., SD card, NAND). The first-stage bootloader is always on the same substrate as the CPU. For the remainder of this document, we will focus 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 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](#).¹⁶

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 TFTP, SD/MMC, NFS, YAFFS, and UART
- Displaying splash screens on the display

¹⁶ <http://www.denx.de/wiki/DULG/Manual>

- Reading and writing SD/MMC
- Accessing a YAFFS NAND flash filesystem
- Performing scripting tasks
- Configuring a Linux kernel

3.2.1 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 (*uImage*) to RAM.
 - 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.
- If using the LCD, the LCD must be specified in the *bootargs* variable to be made available to Linux.
- At boot, U-Boot looks for a file on the SD card named *boot.scr*. If found, U-Boot will run that script using the *autoboot* environment variable script. The *autoboot* script is included as part of the default environment variables coded into the U-Boot source code. See *include/configs/omap3logic.h* in the U-Boot source code for more information.

3.2.2 Configure Boot

The demo binary images provided by Logic PD are to be used to boot from an SD card. Logic PD has provided some scripts in the default environment to help simplify the process of configuring the boot setup.

To configure your boot strategy, there are four environment variables that must be set:

- *\$kernel_location* – This specifies the kernel image (*uImage*) device. Valid values are *ram*, *nand*, *mmc*, or *tftp*.
- *\$rootfs_location* – This specifies the root filesystem device. Valid values are *ram*, *tftp*, */dev*, *nfs*, *mmc*, or *nand*.
- *\$rootfs_type* – This sets the type of root filesystem used. Valid values are *ramdisk*, *jffs*, *yaffs*, *ext3*, or *nfs*.
- *\$loadaddr* – This configures the location in RAM that the kernel must be loaded to for launch. The default value is *0x81000000*.

Additional settings are required for setting specific boot strategies:

- If booting from a */dev*-based file location, then *\$rootfs_device* must also be set with the device used. The device is specified using the Linux filesystem path */dev/mtdblock5*.

- If booting from a RAMdisk image, then *\$ramdisksize* and *\$ramdiskaddr* must also be set.
- If booting from an NFS filesystem, then *\$serverip*, *\$nfsrootpath*, and *\$nfsoptions* must be set where the parameters for *\$nfsoptions* are comma delimited (e.g., wsize=1500, rsize=1500).

There are additional settings that can be used, not all of which will be documented here. A detailed list of those environment variable settings can be found in the U-Boot source code in *include/configs/omap3logic.h*. A short list of several of those settings is included below.

- *\$display* - This variable stores the display configuration used. The default value is 28, identifying the display included with the DM3730 Development Kit.
- *\$ramdiskimage* - This is the file name of the RAMdisk image. The default value is rootfs.ext2.gz.uboot.
- *\$kernelimage* - This is the file name of the Linux kernel image (uImage).
- *\$serverip* - This is the default IP address used for TFTP operations.
- *\$setconsole* - This variable stores the console device settings. The default value is console ttyO0, 115200n8.
- *\$otherbootargs* - This variable has a set of static boot arguments that are passed to the kernel. The default value is ignore_loglevel early_printk no_console_suspend.

3.2.2.1 Configure Display

At boot time, if the display environment variable has not been set, U-Boot will display the following message:

```
===== NOTICE =====
The U-Boot environment was not found. If the display is not set
properly Linux will not have video support.

Valid display options are:
 2 == LQ121S1DG31      TFT SVGA      (12.1)  Sharp
 3 == LQ036Q1DA01      TFT QVGA      (3.6)   Sharp w/ASIC
 5 == LQ064D343         TFT VGA       (6.4)   Sharp
 7 == LQ10D368          TFT VGA       (10.4)  Sharp
15 == LQ043T1DG01       TFT WQVGA     (4.3)   Sharp
28 == LQ043T1DG28       TFT WQVGA     (4.3)   Sharp (DEFAULT) vga[-16 OR -
24]      LCD VGA       640x480
svga[-16 OR -24]      LCD SVGA      800x600
xga[-16 OR -24]       LCD XGA       1024x768
720p[-16 OR -24]      LCD 720P     1280x720
sxga[-16 OR -24]      LCD SXGA      1280x1024
uxga[-16 OR -24]      LCD UXGA      1600x1200

Default `display` environment variable is now being set to: 28
At the U-Boot prompt type commands: `setenv display <num>`, then type
`saveenv` to save the environment to NAND flash. This will avoid
seeing this notice on future boots
===== NOTICE =====
```

To set a different display or to simply avoid this boot-time message, assign the display of your choice to the *\$display* environment variable. The default display is 28. If using a Logic PD display, choose the number from the list above. If using a DVI adaptor on the LCD bus, choose one of the VESA display names followed by -16. If using the HDMI connector on the baseboard, select the VESA display name followed by -24.

The -16 and -24 set the number of bits per pixel. The LCD bus is 16 bits wide; as such, any device plugged into this bus (LCD or DVI adaptor) must be set to -16. HDMI by definition is 24 bits per pixel; therefore, setting the display to -24 is required. In order to configure the proper display, it is up to the user to know which display is connected and what the display capabilities are.

NOTE: Be sure to check your kit for the necessary jumper settings. On the DM3730 Torpedo Development Kit, JP2 must be moved when selecting the LCD parallel bus or HDMI. On the DM3730 SOM-LV Development Kit, JP5 must be moved when selecting LCD parallel bus or HDMI.

U-Boot does support custom displays. To configure a custom display, the *\$display* environment variable must be set to:

```
display=<xres>:<yres>:<hbp>:<hfp>:<vbp>:<vfp>:<hsw>:<vsw>:<pixel_clock>:
<config>:<data-lines>
```

Where:

- *xres* is the number of pixels in the x direction
- *yres* is the number of pixels in the y direction
- *hbp* is the horizontal back porch
- *hfp* is the horizontal front porch
- *vbp* is the vertical back porch
- *vfp* is the vertical front porch
- *hsw* is the horizontal synchronization pulse width
- *vsw* is the vertical synchronization pulse width
- *pixel_clock* is the bit clock
- *config* is the *DISPC_POL_FREQ* register of the DSS controller
- *data-lines* is the number of bits per pixel

As an example, XGA timing in HDMI output (24 data lines) would be:

```
display=1024:768:160:24:29:3:41:6:61714:0x100000:24
```

It is beyond the scope of this document to describe how to configure a custom display, beyond the parameters available in U-Boot. For additional information about configuring custom displays, please post a question to the Logic PD [TDG forum](#).

Once the display has been set in U-Boot, that same display configuration will be passed to the kernel in the *\$bootargs* variable. No further configuration of the Linux kernel is necessary.

NOTE: For users who do not wish to have a display, removing the display environmental variable will not initialize the screen and the driver will not initialize the frame buffer. Remove the environmental variable by setting without any parameters.

```
OMAP Logic # setenv display
```

3.2.3 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
bmp        - manipulate BMP image data
boot       - boot default, i.e., run 'bootcmd'
bootd      - boot default, i.e., run 'bootcmd'
bootm      - boot application image from memory
bootp      - boot image via network using BOOTP/TFTP protocol
chpart     - change active partition
cls        - clear screen
cmp        - memory compare
coninfo    - print console devices and information
cp         - memory copy
crc32      - checksum calculation
dhcp       - boot image via network using DHCP/TFTP protocol
dump_gpmc  - dump_gpmc - dump GPMC settings
dump_id_data - dump_id_data - dump product ID data
echo       - echo args to console
echo_lcd   - echo args to LCD
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 /)
fatinfo    - print information about filesystem
fatload    - load binary file from a dos filesystem
fatls     - list files in a directory (default /)
fsinfo     - print information about filesystems
fsload     - load binary file from a filesystem image
go         - start application at address 'addr'
help       - print online help
i2c        - I2C sub-system
imxtract   - extract a part of a multi-image
itest      - return true/false on integer compare
lcd percent - format for percentage output on LCD
loadb      - load binary file over serial line (kermit mode)
loads      - load S-Record file over serial line
loady      - load binary file over serial line (ymodem mode)
loop       - infinite loop on address range
```

```

ls          - list files in a directory (default /)
madc        - MADC subsystem
md          - memory display
mm          - memory modify (auto-incrementing address)
mmc         - MMC sub-system
mtdparts    - define flash/nand partitions
mtest       - simple RAM read/write test
mux config  - mux config - dump active mux registers
mw          - memory write (fill)
nand        - NAND sub-system
nandecce    - switch OMAP3 NAND ECC calculation algorithm
nboot       - boot from NAND device
nfs         - boot image via network using NFS protocol
nm          - memory modify (constant address)
ping        - send ICMP ECHO REQUEST to network host
poweroff    - Power down board
printenv    - print environment variables
rarpboot    - boot image via network using RARP/TFTP protocol
reset       - Perform RESET of the CPU
run         - run commands in an environment variable
saveenv     - save environment variables to persistent storage
sdram config - sdram config - dump SDRC registers
setenv      - set environment variables
showvar     - print local hushshell variables
sleep       - delay execution for some time
source      - run script from memory
test        - minimal test like /bin/sh
tftpboot    - boot image via network using TFTP protocol
time        - time execution of command
usb         - USB sub-system
usbboot     - boot from USB device
version     - print monitor version
ydebug      - YAFFS debug level
ydf         - YAFFS disk free
ydump       - YAFFS device struct
yls         - yaffs ls
ymkdir      - YAFFS mkdir
ymount      - YAFFS mount
ymv         - YAFFS mv
yrdm        - YAFFS read file to memory
yrm         - YAFFS rm
yrmdir      - YAFFS rmdir
yumount     - YAFFS unmount
ywr         - YAFFS write file from memory
OMAP Logic #

```

3.2.4 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
bootcmd=echo 'Catalyst boot command ...';run autoboot
bootdelay=3
baudrate=115200
bootfile=uImage
lcd anchor=30,0
disablecharging no
loadaddr=0x81000000
nandkerneloffset=0x280000
nandkernelsize=0x360000
scriptaddr=0x80FFB000
yupdatepart=cache
```

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... done
```

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

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

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 OSs 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.

3.2.5 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.

3.2.6 Display Splash Screen

When the power is first turned on, a user requires some feedback to acknowledge that something is happening with the system. To address this, it is a common desire to provide a splash screen on the display as soon as possible while the system boots. This section will provide an example on how to display a splash screen.

The splash screen should be formatted as a bitmap (.bmp) file and should match the screen type being used. In this example, we use the 4.3" display that is included with the DM3730 Development Kit. This display has 480 x 272 pixels with 16 bits per pixel. Each pixel is 5 bits for red, 6 bits for green, and 5 bits for blue (5:6:5 format). Many popular image editors can be used to create, edit, and convert images to provide this format. When writing this example, we used [GIMP](http://www.gimp.org/)¹⁷ to convert an image to 480 x 272 pixels. Using the advanced options, we then saved the image in bitmap format to 16 bits per pixel and R5 G6 B5 with the name *mysplash.bmp*.

1. Copy this file to an SD card. Boot the SOM to U-Boot and use the command below to initialize the SD/MMC card.

```
OMAP Logic # mmc init 1
mmc1 is available
```

2. List the directory of the SD card to ensure the splash screen file is present.

```
OMAP Logic # fatls mmc 1
 44772    mlo
 427320   u-boot.bin
3954200   uimage
14534102  rootfs.ext2.gz.uboot
42100608  rootfs.yaffs2
          .trash-1000/
 427332   u-boot.bin.ift
 261190   mysplash.bmp

7 file(s), 1 dir(s)
```

3. Next, load the splash screen to memory.

```
OMAP Logic # fatload mmc 1 0x81000000 mysplash.bmp
reading mysplash.bmp

261190 bytes read
```

The address *0x81000000* is the destination in RAM where the splash image will be stored. This address is somewhat arbitrary as long as the memory space is a valid memory space and does not impede on the U-Boot memory space. The U-Boot command *bdinfo* will display all the memory information needed to choose an address. Once the display shows the bitmap, this memory space can be reused or discarded.

4. Display the bitmap header information to verify the bitmap is loaded properly and has the proper format for U-Boot.

¹⁷ <http://www.gimp.org/>


```
OMAP Logic # bmp info 0x81000000
Image size      : 480 x 272
Bits per pixel: 16
Compression    : 3
```

5. Verify the LCD backlight is turned off, display the bitmap, and then turn the LCD backlight on to 100%.

```
OMAP Logic # backlight 0
OMAP Logic # bmp display 0x81000000
OMAP Logic # backlight 100
```

Commands similar to those above can be used to script the process of loading your own splash screen when U-Boot starts up. See the following section for additional information on how to write scripts.

3.2.7 Printing Text to the Display

U-boot provide the ability to show text messages on the display. Examples of printing text to the display are provided below.

Here is an example that will put up text at the current position (and increment the cursor position).

```
OMAP Logic # echo_lcd "This is a test"
```

The command below will go back to the beginning of the line and change "this" to "that".

```
OMAP Logic # echo_lcd "/rThat"
```

The command below will go back to the beginning of the line and clear it.

```
OMAP Logic # echo_lcd "/r/k"
```

Here is an example to display inverted text.

```
OMAP Logic # echo_lcd "/iIs it a test?/i"
```

3.2.8 Script with Variables

To script one or more U-Boot commands, the command list is assigned to a variable and the U-Boot *run* command is used to execute that list of commands, as shown below.

```
OMAP Logic # setenv color blue
OMAP Logic # setenv car_color echo $color
OMAP Logic # echo $car_color
```

```
echo blue
```

In the above example, we created the variable *color* and set it to blue. We also created the variable *car_color* and set it equal to *\$color*. Notice when we display the variable *\$car_color*, it does not contain the same text we set it to. The reason is the parser used the value of *\$color*, rather than the literal string *\$color*.

To correct this, we can use the back slash character *"\"* to delay the processing of the *"\$"* character.

```
OMAP Logic # setenv car_color echo \"$color"
OMAP Logic # echo $car_color
echo $color
```

Now, the variable *car_color* can be considered a simple script of one command. To run this script, we use the *run* command.

```
OMAP Logic # run car_color
blue
OMAP Logic #
```

The *"\"* character is crucial to writing scripts due to the delayed processing of a script. Processing of commands is delayed until script execution.

The next example shows how more than one command can be assigned to a variable. Multiple commands can be separated on the command line or in scripts using the semicolon character *";"*. Notice in the example below, we need to continue our use of the *"\"* to prevent the *";"* and the *"\$"* from being processed when we assign our script to a variable.

```
OMAP Logic # setenv car_color blue
OMAP Logic # setenv print_car_color echo The color of my car is\"; echo
\"$car_color\"; echo Done
OMAP Logic # echo $print_car_color
echo The color of my car is; echo $car_color; echo Done
OMAP Logic # run print_car_color
The color of my car is
blue
Done
OMAP Logic # setenv car_color red
OMAP Logic # run print_car_color
The color of my car is
red
Done
OMAP Logic #
```

In the above example, we created a shell variable *car_color* and assigned it a value of *blue*.

Next, we created a second shell variable *print_car_color* and assigned it a command list of three echo commands to print the color of the car. Notice each command was separated by a

"," character. Also notice we used the "\" character to escape the special characters "\$" and ";".

As with any other environment variable, our script can be saved in NAND flash for future use. Using the *saveenv* command will save all the variables defined in our environment, including our script(s). When the system is restarted, the environment is automatically reloaded, and we can again run our script with the command *run print_car_color*.

For more complex scripting, script variables can be defined that call other script variables using the same *run* command our example used above.

3.2.9 Script from Memory

An alternative method to saving and running scripts in the environment is to save the script as a separate file. This requires using the *source* command in the U-Boot shell and the *mkimage* tool included with the U-Boot source. The result of the *mkimage* tool is a file that can be loaded into RAM at the U-Boot prompt; the *source* command can then be used from the U-Boot shell to launch the script. For more details, please see the *source* command in the U-Boot manual available on the [DENX website](http://www.denx.de).¹⁸ Information about how to create the *boot.scr* script can be found in Lab 9 of the [DM3730/AM3703 U-Boot Labs](http://www.logicipd.com/Support/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1413).¹⁹

At boot time, the default environment includes a script that will search the SD card (if present) for a script file named *boot.scr*. If a file is found on the SD card with that file name, U-Boot will load that file into RAM and the *source* command is used to run it. This can be handy in cases where you want to override any boot strategy in NAND flash by simply booting from an SD card including this script file.

3.2.10 Boot from NAND Flash

When booting from NAND flash, the same X-Loader, U-Boot, and kernel images used in booting from an SD card are used.

NOTE: The references to the ECC algorithms in the following sections depend on the NAND device, the processor, and the NAND filesystem. If this hardware configuration differs from the one used in the writing of this document, a different ECC algorithm may be required. The U-Boot environment variable *defaultecc* will contain the default ECC algorithm used by U-Boot.

3.2.10.1 X-Loader

The CPU boot ROM expects to find X-Loader in one of the first four blocks of NAND flash. The reason for the four copies is that NAND flash is subject to bad blocks. If the CPU fails to load X-Loader in the first block, it will proceed to the next block and try again.

The CPU boot ROM also mandates a specific ECC algorithm be used when loading X-Loader. In the context of the U-Boot shell, this is the *hw* ECC algorithm.

3.2.10.2 U-Boot

X-Loader expects to find U-Boot starting from the fifth block of raw NAND flash. It also expects that U-Boot will use the chip ECC algorithm in the context of the U-Boot shell. Rather than having multiple copies of U-Boot to manage bad block situations in NAND flash, X-Loader will detect and skip any bad block it finds. So, if a bad block is encountered as X-Loader loads U-Boot, X-Loader will simply look to the next block for the remainder of U-Boot. For this reason, only one copy of U-Boot is required.

¹⁸ <http://www.denx.de>

¹⁹ <http://support.logicipd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1413>

3.2.10.3 Kernel

Once U-Boot is up and running, the rest of the boot process becomes much more flexible. U-Boot provides methods for reading from SD, YAFFS (filesystem based in NAND flash), TFTP, raw NAND flash, and other MTD devices. In addition, U-Boot is typically where the programming of a SOM's non-volatile memory (flash memory) takes place.

3.2.10.4 RAMdisk NAND Flash Boot Example

To program a bootable NAND flash, the following example uses U-Boot to do the programming. The example will assume a RAM-based filesystem and that X-Loader, U-Boot, and the kernel image are all stored in raw NAND flash. This example uses the binary images from Logic PD's DM37x Linux BSP and all of the commands below are performed in U-Boot. See Section 3.2.13.1 for a script that performs the same task.

NOTE: U-Boot and the environment are constantly undergoing improvement; the example below shows only one method. See the scripts in Section 3.2.11 for the latest implementation.

NOTE: In the following examples, you will see the use of the *nand write* and *nand write.i* commands. The distinction is subtle, but significant. It is beyond the scope of this document to describe all the workings of NAND flash; however, new and used NAND flash devices can have bad blocks. The *nand write* command will write data to the NAND flash device, skipping NAND flash bad blocks and the data that would be written to that block. The *nand write.i* command, on the other hand, will also skip NAND bad blocks, but will continue to write the data to the next good block. Therefore, *nand write* can result in data not written to the NAND flash device, while *nand write.i* will write all data to the NAND flash device.

1. Power on the SOM and press any key to interrupt the boot cycle to get to the U-Boot prompt.
2. Erase the NAND flash completely.

```
OMAP Logic # nand erase.chip
NAND erase.chip: device 0 whole chip
Erasing at 0x1ffe0000 -- 100% complete.
OK
```

3. Load the default environment.

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

4. Set up the U-Boot environment variables to indicate where the kernel and the RAMdisk should be loaded into RAM.

```
OMAP Logic # setenv loadaddr 0x81000000
OMAP Logic # setenv ramdiskaddr 0x82000000
```

5. Set the location and size of the kernel in NAND flash.

It is best to use the location in NAND flash that is consistent with the MTD partitions defined in U-Boot and passed into the kernel. Using the *mtdparts* command at the U-Boot prompt will display the NAND partitions defined. The value used for the

kernel_nand_size should be rounded up to the nearest NAND flash page size. The size set can be larger, but should not be larger than the partition defined in *mtdparts*.

```
OMAP Logic # setenv kernel_nand_offset 0x00280000
OMAP Logic # setenv kernel_nand_size 0x00400000
```

- Set similar offset and size parameters for the RAMdisk in NAND.

NOTE: The RAMdisk size may differ depending on which release you are using. Please use the size of your specific RAMdisk image for the *ramdisk_nand_size* variable. As with the kernel image, the *ramdisk_nand_size* should be rounded up to the nearest NAND flash page size. The size can be larger, but should not be larger than the partition defined in *mtdparts*.

```
OMAP Logic # setenv ramdisk_nand_offset 0x00680000
OMAP Logic # setenv ramdisk_nand_size 0x00dd8680
```

- Specify where the kernel image is located, where the root filesystem is located, and the type of root filesystem being used.

```
OMAP Logic # setenv kernel_location nand
OMAP Logic # setenv rootfs_location nand
OMAP Logic # setenv rootfs_type ramdisk
```

- Save the U-Boot environment.

```
OMAP Logic # saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
```

This concludes configuring U-Boot to boot from NAND flash. Follow the steps below to burn the boot images into NAND.

- Initialize the SD/MMC interface for reading. **NOTE:** This example assumes that all the boot images are on an SD card that is inserted into the bootable SD card slot on the baseboard.

```
OMAP Logic # mmc init
mmc1 is available
```

- Use a temporary space in RAM to load the image. Out of convenience, we simply use *\$loadaddr*. Before loading our image, we cleared the RAM to all 0xFFs because NAND flash forces the data to be aligned on page boundaries. Any extra bytes we write to NAND flash should be the erase byte 0xFF.

```
OMAP Logic # mw.l ${loadaddr} 0xFFFFFFFF 0x400000
```

3. Load X-Loader (named *MLO* on our bootable SD card) to a temporary space in RAM.

```
OMAP Logic # fatload mmc 1 ${loadaddr} mlo
reading mlo

34388 bytes read
```

4. Before we burn X-Loader to NAND flash, we need to choose the *hw* ECC algorithm U-Boot will use when writing to NAND flash.

```
OMAP Logic # nandecc hw
NAND: HW ECC selected
```

5. Now burn X-Loader to NAND flash. Remember that the boot ROM expects the second-stage bootloader (X-Loader) to be in the first four blocks of NAND flash. So, we burn the same copy of X-Loader to the first four blocks.

The burn size chosen here must be on a NAND flash page boundary. There is no harm in burning the entire block of NAND flash and, to keep this example simple, we will burn the entire block.

```
OMAP Logic # nand write ${loadaddr} 0x00000000 0x00020000

NAND write: device 0 offset 0x0, size 0x20000
131072
OMAP Logic # nand write ${loadaddr} 0x00020000 0x00020000

NAND write: device 0 offset 0x20000, size 0x20000
131072 bytes written: OK
OMAP Logic # nand write ${loadaddr} 0x00040000 0x00020000

NAND write: device 0 offset 0x40000, size 0x20000
131072 bytes written: OK
OMAP Logic # nand write ${loadaddr} 0x00060000 0x00020000

NAND write: device 0 offset 0x60000, size 0x20000
131072 bytes written: OK
```

X-Loader is now burned to NAND flash.

6. Next, prepare to burn U-Boot to NAND flash. Begin as we did in X-Loader by writing 0xFF to the temporary RAM space.

```
OMAP Logic # mw.l ${loadaddr} 0xFFFFFFFF 0x400000
```

7. Similar to what was done with X-Loader, load U-Boot to the temporary RAM space.

```
OMAP Logic # fatload mmc 1 ${loadaddr} u-boot.bin
reading u-boot.bin
```

```
439368 bytes read
```

8. For U-Boot and all remaining images, set the programming ECC algorithm to `${defaultecc}`.

```
OMAP Logic # nandecc ${defaultecc}
NAND: Internal to NAND ECC selected
```

9. Write U-Boot to the fifth NAND flash block. As is the case with X-Loader, the burn size must end on a page boundary and burning a few more bytes than what we need causes no harm.

```
OMAP Logic # nand write.i ${loadaddr} 0x00080000 0x00080000

NAND write: device 0 offset 0x80000, size 0x80000
524288 bytes written: OK
```

U-Boot is now burned into NAND flash.

10. Finally, burn the kernel and the root filesystem to NAND flash.

The process here is the same as above; however, for convenience we use the environment variables `$kernel_nand_offset`, `$kernel_nand_size`, `$ramdisk_nand_offset`, and `$ramdisk_nand_size` instead of specifying those parameters explicitly.

```
OMAP Logic # mw.1 ${loadaddr} 0xffffffff 0x400000
OMAP Logic # fatload mmc 1 ${loadaddr} ${kernelimage}
reading uImage

3937748 bytes read
OMAP Logic # nand write.i ${loadaddr} ${kernel_nand_offset}
${kernel_nand_size}

NAND write: device 0 offset 0x280000, size 0x400000
4194304 bytes written: OK
OMAP Logic # mw.1 ${loadaddr} 0xffffffff 0x400000
OMAP Logic # fatload mmc 1 ${loadaddr} ${ramdiskimage}
reading rootfs.ext2.gz.uboot

14500647 bytes read
OMAP Logic # nand write.i ${loadaddr} ${ramdisk_nand_offset}
${ramdisk_nand_size}

NAND write: device 0 offset 0x680000, size 0x00dd8680
14517888 bytes written: OK
```

11. The process of creating bootable NAND flash is now complete. Remove the SD card and cycle the power on the SOM; it will now boot from NAND flash.

3.2.10.5 YAFFS Root Filesystem Boot Example

The boot process is very flexible. The example below boots to the Linux kernel from NAND flash as did the previous example; however, rather than loading a RAMdisk root filesystem image, this example uses an existing root filesystem located on a NAND flash partition. See Section 3.2.13.2 for a script that performs a similar task.

1. Insert the DM37x Linux BSP Demo SD card and power on the SOM.
2. Let the SOM boot into the Linux kernel; enter the username and login as described in the Linux banner.
3. Erase the NAND flash completely.

```
OMAP Logic # nand erase.chip
NAND erase.chip: device 0 whole chip
Erasing at 0x1ffe0000 -- 100% complete.
OK
```

4. At the Linux prompt, mount a YAFFS partition on the NAND device.

```
DM-37x# mkdir /mnt/yaffs-nand
DM-37x# mount -t yaffs /dev/mtdblock5 /mnt/yaffs-nand
```

5. Copy the RAM-based root filesystem to the new mount point.

```
DM-37x# cp -a /bin /dev /etc /home /lib /mnt/yaffs-nand
DM-37x# cp -a /linuxrc /opt /root /sbin /tmp /usr /var /mnt/yaffs-nand
DM-37x# mkdir /mnt/yaffs-nand/{mnt,proc,sys}
```

6. Unmount the NAND partition. This ensures the filesystem cache is flushed to the device and a YAFFS checkpoint is created. A YAFFS checkpoint makes future mounts much faster.

```
DM-37x# umount /mnt/yaffs-nand
```

7. With the demo SD card still inserted, power off the SOM.
8. Power on the SOM and press any key to interrupt the boot cycle to get to the U-Boot prompt.
9. Erase the X-Loader, U-Boot, and kernel partitions.

```
OMAP Logic # nand erase.part x-loader
NAND erase.part: device 0 offset 0, size 0x80000
Erasing at 0x60000 - 100% complete.
OK
OMAP Logic # nand erase.part u-boot
NAND erase.part: device 0 offset 0x80000, size 0x1a0000
Erasing at 0x200000 - 100% complete.
OK
OMAP Logic # nand erase.part kernel
```



```
NAND erase.part: device 0 offset 280000, size 0x500000
Erasing at 0x760000 - 100% complete.
OK
```

10. Load the default environment.

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

11. Set up the U-Boot environment variable to indicate where the kernel should be loaded into RAM.

```
OMAP Logic # setenv loadaddr 0x81000000
```

12. Set the location and size of the kernel in NAND flash.

It is best to use the location in NAND flash that is consistent with the MTD partitions defined in U-Boot and passed into the kernel. Entering the *mtdparts* command at the U-Boot prompt will display the NAND partitions defined. The value used for the *kernel_nand_size* should be rounded up to the nearest NAND flash page size. The size set can be larger, but should not be larger than the partition defined in *mtdparts*.

```
OMAP Logic # setenv kernel_nand_offset 0x00280000
OMAP Logic # setenv kernel_nand_size 0x00400000
```

13. Specify where the kernel image is located, where the root filesystem is located, and the type of root filesystem being used.

```
OMAP Logic # setenv kernel_location nand
OMAP Logic # setenv rootfs_location /dev
OMAP Logic # setenv rootfs_type yaffs
OMAP Logic # setenv rootfs_device /dev/mtdblock5
```

14. Save the U-Boot environment.

```
OMAP Logic # saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0x260000 -- 100% complete.
Writing to Nand... done
```

This concludes configuring U-Boot to boot from NAND flash. Follow the steps below to burn the boot images into NAND.

1. Initialize the SD/MMC interface for reading. **NOTE:** This example assumes that all the boot images are on an SD card that is inserted into the bootable SD card slot on the baseboard.

```
OMAP Logic # mmc init
mmc1 is available
```

2. Use a temporary space in RAM to load the image. Out of convenience, we simply use `$loadaddr`. Before loading our image, we clear the RAM to all 0xFFs because NAND flash forces our data to be aligned on page boundaries. Any extra bytes we write to NAND flash should be the erase byte 0xFF.

```
OMAP Logic # mw.1 ${loadaddr} 0xFFFFFFFF 0x400000
```

3. Load X-Loader (named *MLO* on our bootable SD card) to a temporary space in RAM.

```
OMAP Logic # fatload mmc 1 ${loadaddr} mlo
reading mlo

34388 bytes read
```

4. Before we burn X-Loader to NAND flash, we need to choose the *hw* ECC algorithm U-Boot will use when writing to NAND flash.

```
OMAP Logic # nandecc hw
NAND: HW ECC selected
```

5. Now burn X-Loader to NAND flash. Remember that the boot ROM expects the second-stage bootloader (X-Loader) to be in the first four blocks of NAND flash. So, we burn the same copy of X-Loader to the first four blocks.

The burn size chosen here must be on a NAND flash page boundary. There is no harm in burning the entire block of NAND flash and, to keep this example simple, we will burn the entire block.

```
OMAP Logic # nand write ${loadaddr} 0x00000000 0x00020000

NAND write: device 0 offset 0x0, size 0x20000
131072
OMAP Logic # nand write ${loadaddr} 0x00020000 0x00020000

NAND write: device 0 offset 0x20000, size 0x20000
131072 bytes written: OK
OMAP Logic # nand write ${loadaddr} 0x00040000 0x00020000

NAND write: device 0 offset 0x40000, size 0x20000
131072 bytes written: OK
OMAP Logic # nand write ${loadaddr} 0x00060000 0x00020000

NAND write: device 0 offset 0x60000, size 0x20000
131072 bytes written: OK
```

X-Loader is now burned to NAND flash.

6. Next, prepare to burn U-Boot to NAND flash. Begin as we did in X-Loader by writing 0xFF to the temporary RAM space.

```
OMAP Logic # mw.1 ${loadaddr} 0xFFFFFFFF 0x400000
```

7. Similar to what was done with X-Loader, load U-Boot to the temporary RAM space.

```
OMAP Logic # fatload mmc 1 ${loadaddr} u-boot.bin
reading u-boot.bin

439368 bytes read
```

8. For U-Boot and all remaining images, set the programming ECC algorithm to `${defaultecc}`.

```
OMAP Logic # nandecc ${defaultecc}
NAND: Internal to NAND ECC selected
```

9. Write U-Boot to the fifth NAND flash block. As is the case with X-Loader, the burn size must end on a page boundary and burning a few more bytes than what we need causes no harm.

```
OMAP Logic # nand write.i ${loadaddr} 0x00080000 0x00080000

NAND write: device 0 offset 0x80000, size 0x80000
524288 bytes written: OK
```

U-Boot is now burned into NAND flash.

10. Finally, burn the kernel to NAND flash.

The process here is the same as above; however, for convenience we use the environment variables `$kernel_nand_offset` and `$kernel_nand_size`, instead of specifying those parameters explicitly.

```
OMAP Logic # mw.1 ${loadaddr} 0xffffffff 0x400000
OMAP Logic # fatload mmc 1 ${loadaddr} ${kernelimage}
reading uImage

3937748 bytes read
OMAP Logic # nand write.i ${loadaddr} ${kernel_nand_offset}
${kernel_nand_size}

NAND write: device 0 offset 0x280000, size 0x400000
4194304 bytes written: OK
```

11. The process of creating bootable NAND flash is now complete. Remove the SD card and cycle the power on the SOM; it will now boot from NAND flash.

3.2.11 Boot with X-Loader, U-Boot, Kernel, and Root Filesystem on SD Card

The Linux Demo image has the X-loader, U-Boot, and kernel booting from the SD card with a RAM-based root filesystem loaded into RAM. Any changes to the RAM-based root filesystem are lost following a reset.

The example in this section provide a way to create a two partition SD card with a FAT partition holding X-loader, U-Boot, and the kernel and an ext3 partition containing the root filesystem. These instructions have been verified using the Virtual Machine SDK for the DM37x Linux BSP v2.4-2.

A script is provided to simplify the steps required to create an SD card with root filesystem.

12. Download the *create_sdcard* script from [here](#).²⁰
13. Place the *create_sdcard* script into the LTIB root directory (i.e., *~/logic/Logic_BSPs/Linux_3.0/REL-Itib-DM3730-2.4-2*).
14. Give yourself permission to run the script.

```
bash$ chmod 777 create_sdcard.sh
```

15. Insert an SD card into your Linux host PC.
16. Run the script.

```
bash$ ./create_sdcard.sh
Devices available:
  sdb is 1.8GB - Card Reader
Enter device: sdb
Setting up sdb

Do you wish to continue? (y/N) y
Partitioning sdb.
[sudo] password for logic:
mke2fs 1.42 (29-Nov-2011)
Mounting bootloader partition
Mounting root partition
Flushing data to SD card
Unmounting bootloader partition
Unmounting root partition
```

17. After the script has completed, remove the SD card and insert it into your DM3730/AM3703 SOM system.
18. Boot the system and press any key to pause at the U-Boot prompt
19. Update the following U-Boot variables.

```
OMAP Logic # nand erase.chip
OMAP Logic # env default -f
OMAP Logic # setenv rootfs_location /dev
OMAP Logic # setenv rootfs_type ext3
```

²⁰ http://support.logipd.com/Portals/0/Users/049/05/305/create_sdcard.zip

```
OMAP Logic # setenv rootfs_device /dev/mmcblk0p2
OMAP Logic # setenv kernel_location mmc
OMAP Logic # saveenv
OMAP Logic # reset
```

20. Verify your system root filesystem is running from `/dev/mmcblk0p2`.

```
DM-37x# df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/root              1631756    203676   1345188  13% /
tmpfs                  120400         52    120348   0% /dev
/dev/mmcblk0p2         1631756    203676   1345188  13% /mnt/mmcblk0p2
/dev/mmcblk0p1         307016      4384    302632   1% /mnt/mmcblk0p1
shm                    120400         0    120400   0% /dev/shm
rwfs                     512         0         512   0% /mnt/rwfs
DM-37x#

DM-37x# cat /proc/cmdline

nand-ecc=chip console=ttyO0,115200n8 display=28 ignore loglevel
early printk no console suspend mtdparts=omap2-nand.0:512k(x-
loader),1664k(u-boot),384k(u-boot-env),5m(kernel),20m(ramdisk),-(fs)
root=/dev/mmcblk0p2 rw rootfstype=ext3 rootwait DM-37x#
```

3.2.12 Boot with Read-Only Root File system

A read-only root file system can be desired when developers want to prevent possible corruption to NAND flash devices due to unexpected system power down events. While YAFFS file systems protects against sudden loss of power developers may also consider using a read-only root file system to limit the number of writes to NAND memory has a limited number of P/E cycles.

Read-only partitions also provide a more stable system. System behavior is easily reproducible when a system is design using read-only root file systems.

Here are the steps for changing the default read/write partition to a read-open partitioning in YAFFS.

1. Load LTIB configuration window

```
$ ./ltib -c
```

2. In the LTIB Configuration Menu select '*Options --->*' under '*Target Image Generation*'

```

LTIB: Logic OMAP3530/03&DM3730/03 reference boards
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters
are hotkeys. Pressing <Y> selects a feature, while <N> will exclude a feature.
Press <Esc><Esc> to exit, <?> for Help. Legend: [*] feature is selected [ ]
feature is excluded
^(-)
--- Choose your Kernel
  kernel (Linux 3.0-omap-logic) --->
(0) Kernel build verbosity
[ ] Always rebuild the kernel
[ ] Produce cscope index
(omap3logic_defconfig) kernel preconfig
--- Include kernel headers
[*] use 'make headers_install' to install kernel headers
[*] Configure the kernel
--- Leave the sources after building
--- Package selection
  Package list --->
--- Target System Configuration
  Options --->
--- Target Image Generation
  Options --->
(BSP-dm37x-2.4-4) BSP Release Level (supplied by platform defconfig)
v(+)

<Select>  < Exit >  < Help >

```

3. In the Option menu make the following changes.

- [*] read-only root filesystem
- (4M) tmpfs size
- (/tmp /etc /var) Place these dirs in writable RAM

```

Options
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters
are hotkeys. Pressing <Y> selects a feature, while <N> will exclude a feature.
Press <Esc><Esc> to exit, <?> for Help. Legend: [*] feature is selected [ ]
feature is excluded
--- Choose your root filesystem image type
  Target image: (yaffs2) --->
[ ] Create a combined ELF image
[ ] Run a command after building the target image
[*] read-only root filesystem
[*] mount debugfs on /debug during startup
(4M) tmpfs size
(/tmp /etc /var) Place these dirs in writable RAM
() rootfs target directory
[*] Keep temporary staging directory
[ ] Convert hard links to symbolic links
[*] remove man pages etc from the target image
[*] remove the /boot directory
[*] remove the /usr/src/ directory
[*] remove the /usr/include directory
[*] remove the /usr/share/locale directory
() remove these directories
v(+)

<Select>  < Exit >  < Help >

```

4. Exit and save all changes to rebuild the Linux OS image.

5. Create a SD card using the following option from the LTIB root directory

```
$ ./bin/mkLogicFATcard.sh -cy
```

6. Move the newly created SD card to the embedded system.

7. Power on the embedded system
8. Press any key to stop at the u-boot prompt
9. Program NAND memory with YAFFS partition

```
OMAP Logic # run makeyaffsboot
```

10. Update the u-boot environment variables using the following commands.

```
OMAP Logic # setenv otherbootargs no_console_suspend ignore_loglevel ro
OMAP Logic # setenv _set_rootfs_type_yaffs 'setenv bootargs ${bootargs} ro rootfstype=yaffs2'
OMAP Logic # saveenv
```

11. Boot the Linux kernel

```
OMAP Logic # boot
```

12. Verify the YAFFS2 partition is Read-Only. Any attempts to write to the root file system should return a message indicating the file system is a Read-only. Any rights to the RAM-based partitions are no persistence between resets.

```
DM-37x# mount
rootfs on / type rootfs (rw)
/dev/root on / type yaffs2 (ro,relatime)
proc on /proc type proc (rw,relatime)
sys on /sys type sysfs (rw,relatime)
tmpfs on /dev type tmpfs (rw,relatime,mode=755)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
shm on /dev/shm type tmpfs (rw,relatime)
rwfs on /mnt/rwfs type tmpfs (rw,relatime,size=4096k)
rwfs on /tmp type tmpfs (rw,relatime,size=4096k)
rwfs on /etc type tmpfs (rw,relatime,size=4096k)
rwfs on /var type tmpfs (rw,relatime,size=4096k)
usbfs on /proc/bus/usb type usbfs (rw,relatime)
debug on /sys/kernel/debug type debugfs (rw,relatime)
```

NOTE: Developers can ignore the first line '*rootfs on / type rootfs (rw)*' seen above. This is supposed to be hidden since it is a virtual fs that only the kernel users.

3.2.13 Useful Scripts

To help simplify some actions common to many developers, Logic PD includes several scripts in the U-Boot default environment. This section describes these scripts and provides information on how to use them.

NOTE: Use the *printenv* command followed by the script name to display the script. These scripts can provide useful examples for users to modify or write their own scripts.

3.2.13.1 *makenandboot* Script

The *makenandboot* script is used to perform all steps in Section 3.2.10.4. This script assumes the following files are saved on the SD card:

- X-Loader (file name *MLO*)
- U-Boot (file name *u-boot.bin*)
- Signed U-Boot (file name *u-boot.bin.ift*). This file includes the size of the image in the file header and is used when booting from NAND flash to speed up boot by preventing unused NAND flash from being read.
- Linux kernel (file name *uImage*)
- RAM-based root filesystem (file name *rootfs.ext2.gz.uboot*)

With these files on an SD card that is inserted into the bootable SD card slot on the baseboard, the *makenandboot* script can be run using the command below.

```
OMAP Logic # run makenandboot
```

After entering the command, review the output for any errors. Once complete, remove the SD card and cycle the power on the SOM.

3.2.13.2 *makeyaffsboot* Script

The *makeyaffsboot* script is used to perform all steps in Section 3.2.10.5. This script assumes the following files are saved on the SD card:

- X-Loader (file name *MLO*)
- U-Boot (file name *u-boot.bin*)
- Signed U-Boot (file name *u-boot.bin.ift*). This file includes the size of the image in the file header and is used when booting from NAND flash to speed up boot by preventing unused NAND flash from being read.
- Linux kernel (file name *uImage*)
- YAFFS-based root filesystem (file name *rootfs.yaffs2*). See Section 2.5.5.1 for additional information about how to create a *rootfs.yaffs2* image.

With these files on an SD card that is inserted into the bootable SD card slot on the baseboard, the *makeyaffsboot* script can be run using the command below.

```
OMAP Logic # run makeyaffsboot
```

After entering the command, review the output for any errors. Once complete, remove the SD card and cycle the power on the SOM.

3.2.14 Debug UART

The default debug console is UARTA. The information below provides an example for developers looking to move the debug console from UARTA to UARTB. Similar steps can be used by developers looking to move the debug console from UARTA to UARTC.

1. Build default system using *defconfig* in Section 2.4.
2. Modify the *omap3logic.h* file located at `~\logic\logic\Logic_BSPs\Linux_3.0\REL-Itib-DM3730-2.4-2\rpm\BUILD\u-boot-2011.06\include\configs\omap3logic.h`.

Change line 108 from:

```
/*
 * select serial console configuration
 */
//#define CONFIG CONS INDEX          1
//#define CONFIG SYS NS16550 COM1      OMAP34XX UART1
//#define CONFIG SERIAL1              1      /* UART1 on OMAP3
EVMOMAP Logic boards */
```

To:

```
/*
 * select serial console configuration
 */
#define CONFIG CONS INDEX          3
#define CONFIG SYS NS16550 COM3      OMAP34XX UART3
#define CONFIG SERIAL3              3      /* UART3 on OMAP3 EVMOMAP
Logic boards */
```

3. Modify the *logic.c* file located at `~\logic\logic\Logic_BSPs\Linux_3.0\REL-Itib-DM3730-2.4-2\rpm\BUILD\u-boot-2011.06\board\ti\logic\logic.c`.

Change line 1198 from:

```
    MUX_VAL(CP(UART3_RX_IRRX), (IEN | PTD | EN | M7));
/*UART3_RX_IRRX*/
    MUX_VAL(CP(UART3_TX_IRTX), (IEN | PTD | EN | M7));
/*UART3_TX_IRTX*/
```

To:

```
    MUX_VAL(CP(UART3_RX_IRRX), (IEN | PTD | DIS | M0));
/*UART3_RX_IRRX*/
    MUX_VAL(CP(UART3_TX_IRTX), (IDIS | PTD | DIS | M0));
/*UART3_TX_IRTX*/
```

4. The default UART debug console in the kernel is also UARTA. To change the default debug console in the kernel change the *consoledevice* to *ttyO2* for UARTB in the U-Boot variable. Use *ttyO1* for UARTC.

```
OMAP Logic # setenv consoledevice ttyO2
```

5. Save the environment to NAND so that the variable is available on the next boot.

```
OMAP Logic # saveenv  
Saving Environment to NAND...  
Erasing Nand...  
Erasing at 0x260000 -- 100% complete.  
Writing to Nand... done
```

6. Modify the *defaults.lkc* file located at `~\logic\logic\Logic_BSPs\Linux_3.0\REL-Itib-DM3730-2.4-2\config\userspace\defaults.lkc`.

Change line 124 from:

```
config SYSCFG CONSOLEDEV  
    string  
    default "ttyS0"
```

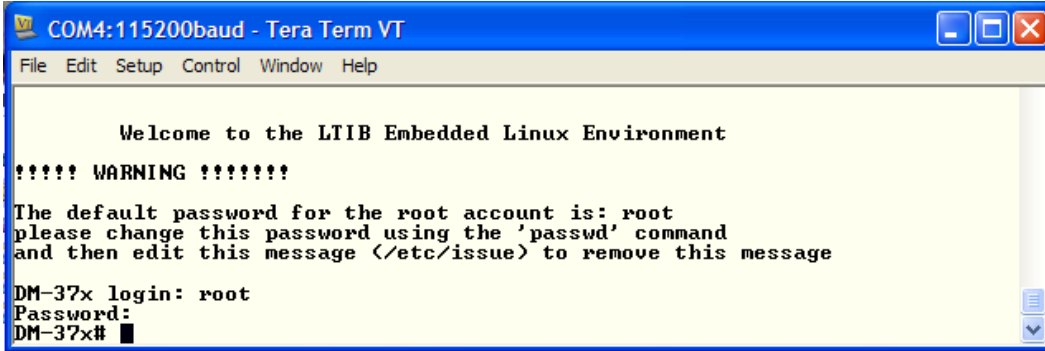
To:

```
config SYSCFG CONSOLEDEV  
    string  
    default "ttyO2"
```

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](http://www.kernel.org/)²¹ 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.

Follow the steps below to convert the DocBook XML files in that directory to HTML files so you can view them with any web browser. Be sure to replace *<your LTIB directory>* with a value appropriate for your system.

```
bash$ sudo apt-get install xmlto xsltex
bash$ cd <your LTIB directory>
bash$ cd rpm/BUILD/linux
bash$ make htmldocs
```

Then, use your favorite browser to access the DocBook *rpm/BUILD/linux-3.0/Documentation/DocBook/index.html* page.

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.

²¹ <http://www.kernel.org/>

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.

```
DM-37x# vi /etc/network/interfaces
```

4.2 Retrieve BSP Version

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

```
DM-37x# cat /proc/version
Linux version 3.0.101-BSP-dm37x-2.4-4 (logic@logic-Virtualbox) (gcc
version 4.3.3 (Sourcery G++ Lite 2009q1-203) ) #8 Thu Aug 27 08:40:27
CDT 2015
```

4.3 Display Product ID System Information

Information specific to the SOM is stored in the Product ID chip on the module. This information is stored in files when booting the Linux OS. The commands below allow the user to access this information at the command line.

```
# Logic PD (LPD) part number
DM-37x# cat /sys/class/product_id/part_number

# Logic PD (LPD) model number
DM-37x# cat /sys/class/product_id/model_name

# Logic PD (LPD) serial number
DM-37x# cat /sys/class/product_id/serial_number

# LAN MAC address for on-board Ethernet
DM-37x# cat /sys/class/product_id/lan_macaddr

# Wi-Fi MAC address for on-board Wi-Fi
DM-37x# cat /sys/class/product_id/wifi_macaddr
```

4.4 Display Linux System Information

The Linux commands below provide additional system information.

```
# provided cmdline passed in by U-Boot to the kernel
DM-37x# cat /proc/cmdline

# memory space
DM-37x# cat /proc/meminfo

# partition information on NAND and NOR flash (does not display SD card
or USB memory stick partitions)
DM-37x# cat /proc/mtd
```

```
# partition size availability
DM-37x# df -k
```

4.5 Wired Networking

4.5.1 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 *otherbootargs* environment variable. See Section 3.2.5 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.

4.5.1.1 Use *ifconfig* Command to Report Ethernet Status

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

```
DM-37x# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:08:EE:05:88:F1
          inet addr:10.0.5.202  Bcast:0.0.0.0  Mask:255.255.252.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4398 errors:0 dropped:15 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:459964 (449.1 KiB)  TX bytes:378 (378.0 B)
          Interrupt:33

DM-37x#
```

4.5.1.2 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 *otherbootargs* environment variable.

You can then execute the boot command or save the environment for future boot from power up. See Section 3.2.5 for more information on setting U-Boot environment variables.

```
OMAP Logic # echo $otherbootargs
ignore loglevel early printk no console suspend
OMAP Logic # setenv otherbootargs ${otherbootargs} ip=:::::dhcp
OMAP Logic # echo $otherbootargs
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 3.2.4 for more information.

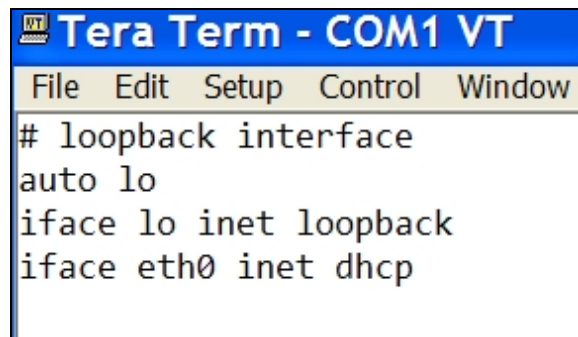
4.5.1.3 Add Static IP Address from Kernel Command Line

To add a static IP address to the kernel command line, follow the example in Section 4.5.1.2, 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.

4.5.1.4 Using DHCP

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

```
iface eth0 inet dhcp
```



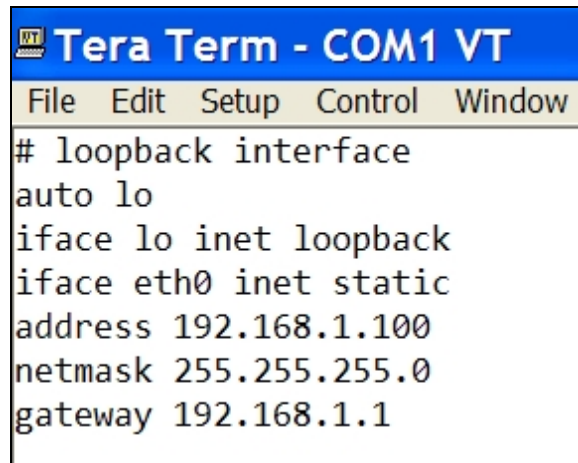
Next, use the *ifup* command to bring up the interface.

```
DM-37x# ifup eth0
DM-37x#
```

4.5.1.5 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 <>
```



```

Tera Term - COM1 VT
File Edit Setup Control Window
# loopback interface
auto lo
iface lo inet loopback
iface eth0 inet static
address 192.168.1.100
netmask 255.255.255.0
gateway 192.168.1.1

```

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

```

DM-37x# ifup eth0
DM-37x#

```

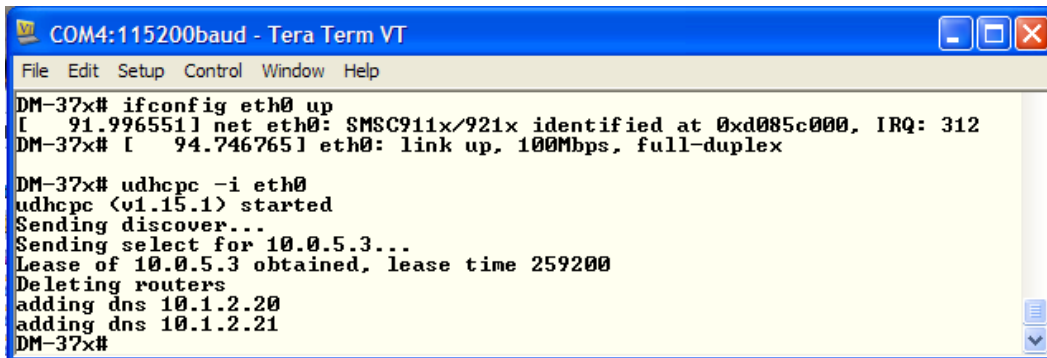
4.5.1.6 Use *ifconfig* Command with DHCP

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

```

DM-37x# ifconfig eth0 up
DM-37x# udhcpc -i eth0

```



```

COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help
DM-37x# ifconfig eth0 up
[ 91.996551] net eth0: SMSC911x/921x identified at 0xd085c000, IRQ: 312
DM-37x# [ 94.746765] eth0: link up, 100Mbps, full-duplex

DM-37x# udhcpc -i eth0
udhcpc (v1.15.1) started
Sending discover...
Sending select for 10.0.5.3...
Lease of 10.0.5.3 obtained, lease time 259200
Deleting routers
adding dns 10.1.2.20
adding dns 10.1.2.21
DM-37x#

```

4.5.2 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.

```
DM-37x# ethtool -s eth0 autoneg off speed 10 duplex half
DM-37x# ethtool -s eth0 autoneg on speed 100 duplex
```

4.5.3 Test Network

The demo image includes the *inetd* program. To enable services, uncomment the appropriate line in the *inetd* configuration file found in */etc/inetd.conf*.

1. Start the *inetd* server.

```
DM-37x# /etc/rc.d/init.d/inetd start
```

2. If you modify the configuration file while *inetd* is running and would like *inetd* to reconfigure itself, send it the HUP signal.

```
DM-37x# killall -HUP inetd
```

3. You may also start the Dropbear SSH server on your DM3730 Development Kit using the command below.

```
DM-37x# /etc/rc.d/init.d/dropbear start
```

4. Once the Dropbear server has been started on the SOM, you may SSH to the SOM using the command below on your Linux host PC, where *ww.xx.yy.zz* is the SOM's IP address:

```
ssh root@ww.xx.yy.zz
```

4.6 Linux Processes

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

4.6.1 *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.

```
DM-37x# ps agux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.3	0.2	2144	624	?	Ss	22:23	0:06	init
root	2	0.0	0.0	0	0	?	S	22:23	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	22:23	0:00	[ksoftirqd/0]
root	4	0.0	0.0	0	0	?	S	22:23	0:00	[kworker/0:0]
root	5	0.0	0.0	0	0	?	S	22:23	0:00	[kworker/u:0]
root	6	0.1	0.0	0	0	?	S	22:23	0:01	[rcu_kthread]
root	7	0.0	0.0	0	0	?	S<	22:23	0:00	[khelper]
root	11	0.0	0.0	0	0	?	S	22:23	0:00	[sync_supers]
root	12	0.0	0.0	0	0	?	S	22:23	0:00	[bdi-default]
root	13	0.0	0.0	0	0	?	S<	22:23	0:00	[kblockd]
root	14	0.0	0.0	0	0	?	S<	22:23	0:00	[omap2_mcspi]
root	15	0.0	0.0	0	0	?	S	22:23	0:00	[khubd]
root	17	0.0	0.0	0	0	?	S	22:23	0:00	[twl4030-irq]
root	19	0.0	0.0	0	0	?	S	22:23	0:00	[kworker/u:1]
root	20	0.0	0.0	0	0	?	S<	22:23	0:00	[l2cap]
root	21	0.0	0.0	0	0	?	S<	22:23	0:00	[cfg80211]
root	22	0.0	0.0	0	0	?	S	22:23	0:01	[kworker/0:1]
root	23	0.0	0.0	0	0	?	S<	22:23	0:00	[rpciod]
root	24	0.0	0.0	0	0	?	S	22:23	0:00	[kswapd0]
root	26	0.0	0.0	0	0	?	S<	22:23	0:00	[nfsiod]
root	33	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock0]
root	34	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock1]
root	35	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock2]
root	36	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock3]
root	37	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock4]
root	38	0.0	0.0	0	0	?	S	22:23	0:00	[mtdblock5]
root	41	0.0	0.0	0	0	?	S<	22:23	0:00	[kpsmouse]
root	44	0.0	0.0	0	0	?	S<	22:23	0:00	[krfcomm]
root	45	0.0	0.0	0	0	?	S	22:23	0:00	[mmcqd/0]
root	56	0.1	0.2	1828	560	?	S<s	22:23	0:02	udevd --daemon
root	651	0.0	0.1	2144	448	?	Ss	22:23	0:00	/sbin/syslogd
root	653	0.0	0.1	2144	464	?	Ss	22:23	0:00	/sbin/klogd
root	681	0.0	0.5	2428	1220	tty00	Ss	22:23	0:00	-sh
root	705	0.0	0.0	0	0	?	S	22:49	0:00	[flush-1:0]
root	707	0.0	0.3	2248	848	tty00	R+	22:49	0:00	ps agux

4.6.2 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.

```
DM-37x# kill -9 688
DM-37x#
```

4.7 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.

4.7.1 Draw Test

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

```
DM-37x# draw-test
```

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

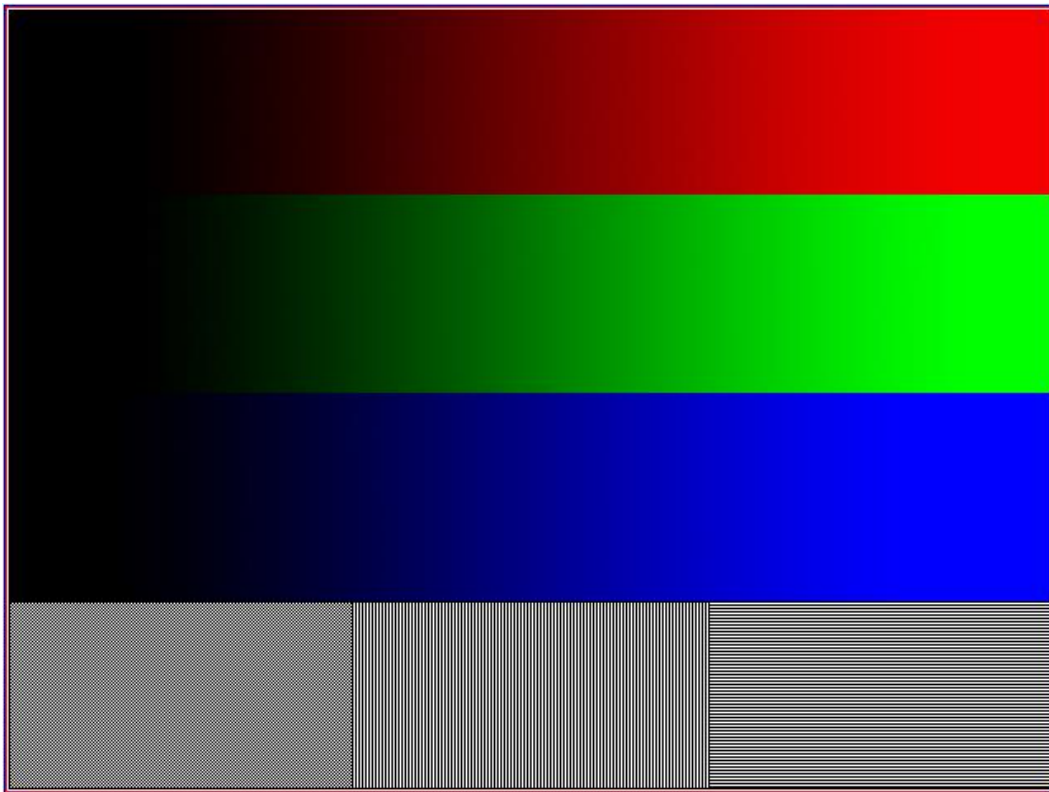


Figure 4.1: LCD Draw Test Display Pattern

4.7.2 DirectFB

The demo image comes with several DirectFB sample programs. You should launch these test programs in the background using the "&" character to avoid losing your command prompt, as the programs do not take input over `ttyS0` once they have started.

For example, try the following programs one at a time, killing each one using *kill -9 pid* after it completes.

```
DM-37x# df_dok &
DM-37x# df_andi &
DM-37x# df_knuckles &
```

4.7.3 Backlight

The backlight can be controlled using the commands below. The range can be anywhere from 0 (off) to 255 (full brightness).

```
DM-37x# echo 0 > /sys/class/backlight/omap3logic/brightness
DM-37x# echo 255 > /sys/class/backlight/omap3logic/brightness
```

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 otherbootargs "$otherbootargs 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.

```
DM-37x# echo -e '\033[13]' > /dev/tty0
```

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

```
DM-37x# echo -e '\033[9;<n>]' > /dev/tty0
```

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

```
DM-37x# echo -e '\033[9;5]' > /dev/tty0
```

The following command cancels screen blanking.

```
DM-37x# echo -e '\033[9]' > /dev/tty0
```

4.7.4 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`.

```
DM-37x# echo "Hello World" > /dev/tty0
```

4.8 Audio

Support for audio-out is available in the demo image via the following programs:

- `aplay <sound file>`
- `mp3play <MP3 file>`

Volume can be controlled using the command below; the `<field>` and `<limit>` variables in the command are defined in Table 4.1. **NOTE:** The quotes around the `<field>` strings in Table 4.1 are required when using the `amixer` command.

```
DM-37x# amixer set <field> <limit>
```

Table 4.1: Field and Limit Definitions

<code><field></code>	<code><limit></code>
"DAC1 Digital Fine"	0-63
"DAC1 Analog"	0-18
"DAC1 Digital Coarse"	0-2

The following command can be used to mute:

```
DM-37x# amixer set Master mute
Simple mixer control 'Master',0
  Capabilities: pswitch pswitch-joined
  Playback channels: Mono
  Mono: Playback [off]
```

The following command can be used to unmute:

```
DM-37x# amixer set Master unmute
Simple mixer control 'Master',0
  Capabilities: pswitch pswitch-joined
  Playback channels: Mono
  Mono: Playback [on]
```

To test microphone record from line-in:

```
DM-37x# arecord -f cd --duration=10 > ~/output.wav
DM-37x# aplay ~/output.wav
```

To test the microphone to record from audio-in follow these steps:

1. Default is line-in. If it is desired, skip to step 2. If a microphone is to be used, enable headset mic bias:

```
DM-37x# i2cset -f -y 1 0x49 0x04 0x04
```

2. Launch the `alsamixer` application.

```
DM-37x# alsamixer
```

3. Hit tab - this will switch it from Playback mode to Capture mode
4. Arrow over to Analog Left AUXL and disable it using the space bar
5. Arrow over to Analog Left Headset Mic and enable it
6. Arrow over to Analog Right AUXR and disable it
7. Connect the tip of the mic to the MIC_IN signal and then ground the other mic signal. If the audio is faint increase the gain using AlsaMixer (first item on the left) to 30 Db as seen in the figure below.

Note: The audio driver is located in the `~/logic/Logic_BSPs/Linux_3.0/REL-Itib-DM3730-2.x-x/rpm/BUILD/linux/sound` directory of the Linux kernel. The `OPT_MODE=0` is set by default and must not be changed. Caution should be taken when making changes to the audio configuration as it is complex.

This table below can be used when cross referencing Figure 14-7 (Voice/Audio Option 2 (OPT_MODE = 0) Block Diagram) in the [TPS65950 Technical Reference Manual Rev G](#)²².

amixer name	TPS65950 register	Default amixer Setting (range)	Register Setting
Analog	ANAMIC_GAIN	5 (0-5)	0x5 (30dB)
TX1 Digital	ATXL1PGA/ATXR1PGA	15 (0-31)	0xf (15dB)
DAC1 Digital Coarse	ARXL1PGA/ARXR1PGA coarse	1 (0-2)	0x1 (6dB)
DAC1 Digital Fine	ARXL1PGA/ARXR1PGA fine	63 (0-63)	0x3f (0dB)
DAC1 Analog	ARXL1_APGA_CTL/ARXR1_APGA_CTL	12 (0-18)	0x6 (0dB)
Headset	HS_GAIN_SET	1 (0-3)	0x3 (-6dB)

²² <http://www.ti.com/lit/pdf/swcu050>

4.9 External Memory Interface

The demo image includes support for SD/MMC memory cards. If the card is present at boot time, it should be automatically detected, mounted, and made available at `/mnt/mmcblk0p1`.

The SD/MMC interface is found at `/dev/mmcblk0` and the first partition is located at `/dev/mmcblk0p1`. If the card is not mounted automatically, 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`.

```
DM-37x# mkdir /mnt/sdcard
DM-37x# mount -t vfat /dev/mmcblk0p1 /mnt/sdcard
```

4.10 Touch Screen

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

To view raw touch data, try the octal dump command and then touch the screen. Press **Ctrl+C** to exit raw data view.

To calibrate the touch screen, use the commands below.

```
DM-37x# export TSLIB_TSDEVICE=/dev/input/event0
DM-37x# export TSLIB_CALIBFILE=/etc/pointercal
DM-37x# export TSLIB_CONSOLEDEVICE=none
DM-37x# export TSLIB_FBDEVICE=/dev/fb0
DM-37x# ts_test
```

Verify touches respond appropriately and then enter the command below.

```
DM-37x# ts_calibrate
```

4.11 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.

```
DM-37x# cat /proc/mtd
```

```

COM201:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
DM-37x# cat /proc/mtd
dev:   size      erasesize  name
mtd0: 00080000 00020000 "x-loader"
mtd1: 001a0000 00020000 "u-boot"
mtd2: 00060000 00020000 "u-boot-env"
mtd3: 00500000 00020000 "kernel"
mtd4: 01400000 00020000 "ramdisk"
mtd5: 1e480000 00020000 "fs"
mtd6: 00800000 00020000 "nor-flash"
DM-37x#

```

NOTE: The above partition map may differ from that on your DM3730 Development Kit depending on the Linux BSP version and the hardware you have installed.

4.11.1 Erase Flash Partitions

WARNING: Do not erase any partitions unless you are certain you know what they are and that your system will be able to recover. Please understand how your system boots before erasing any partitions.

Before you begin to use a flash partition, you should completely erase it. Use the command below to do so, where *x* is the number of the partition you wish to erase as identified in the output from the `/proc/mtd` command.

```
DM-37x# flash_eraseall /dev/mtdx
```

For example, to completely erase the NOR flash partition identified in Section 4.11 above, use the command below.

```
DM-37x# flash_eraseall /dev/mtd6
```

4.11.2 Mount NOR Flash using JFFS2

NOTE: Support for JFFS2 may be limited. Starting with the Linux kernel 3.0, there have been some fundamental changes to the kernel prohibiting the use of JFFS2. Development of JFFS2 by the open source community has also ceased.

You can use the `mount` command to access on-board NOR flash. In the example below, the NOR flash filesystem partitions previously mentioned are mounted using the JFFS2 flash filesystem.

```

DM-37x# mkdir /mnt/jffs2-nor (make a mount point)
DM-37x# mount -t jffs2 /dev/mtdblock6 /mnt/jffs2-nor (mount the file
system)

```

If errors occur format the NOR flash using one of these two methods.

1. Restart the system using LogicLoader and erase the flash using `'erase /dev/flash0 B0 B64'`.

2. From within Linux run 'flash_eraseall /dev/mtd6'.

4.11.3 Mount NAND Flash using YFFS2

You can use the *mount* command to access on-board NAND flash. In the example below, the NAND flash filesystem partitions are mounted using the YFFS2 flash filesystem.

```
DM-37x# mkdir /mnt/yffs2-nand
DM-37x# mount -t yffs2 /dev/mtdblock5 /mnt/yffs2-nand
```

4.12 Wireless Networking with Linux 2.6x Kernels

IMPORTANT NOTE: Kernel 2.6x is considered deprecated in terms of Logic PD support. Users are encouraged to update to the latest BSP which uses a 3.0 based Linux kernel.

4.13 Wireless Networking with Linux 3.x Kernels

Wireless networking with Linux 3.x kernels uses open source drivers and an open source WPA supplicant. To simplify starting the wireless network, Logic PD includes two scripts in the Linux filesystem. One script is for station mode (client-side connection) and the other script is for access point (AP) mode (host Wi-Fi connections). These scripts will start up all the necessary services for each mode. By reviewing these scripts, customers can create their own connections for their application.

NOTE: If you are using a Logic PD Linux BSP with a kernel version prior to 3.x, please see Section 4.12.

NOTE: Customers who have critical data to send over WiFi should ensure the SOM remains within the range of other devices on the WiFi network. Due the volatile nature of WiFi and the potential for external influences, customers are also suggested to implement a retry mechanism in the event the SOM is unable to transmit data.

4.13.1 Start Wireless Interface in Station Mode

As mentioned above, a script is used to start the wireless interface and additional details about the script can be reviewed using the *cat* command at the Linux shell prompt.

Below is an example of how to start the Wi-Fi network in station mode. Run the script and answer the three subsequent questions using the following information:

- SSID: *myssid*
- Encryption: *WPA2*
- Passphrase: *myssidpassphrase*

```
DM-37x# /etc/rc.d/init.d/network-wifi-station init
Importing configuration variables from /etc/rc.d/rc.conf
Bring down wlan0
ifdown: interface wlan0 not configured
Stopping wpa_supplicant:
```



```

killall: wpa supplicant: no process killed
Removing /etc/wpa supplicant.conf
wpa supplicant: Missing /etc/wpa supplicant.conf; recreating
wpa supplicant: Missing SSID/Passphrase
wpa supplicant: Enter WiFi SSID to connect to: myssid
wpa supplicant: Enter Encryption mode (NONE, WEP40, WEP128, WPA, WPA2):
WPA2
wpa supplicant: Enter WiFi WPA2 passphrase: myssidpassphrase
loading wl12xx sdio
[ 62.964935] wl12xx: loaded
[ 62.967803] wl12xx: initialized
Starting wpa supplicant:
[ 64.270355] wl1283: firmware booted (Rev 7.1.3.50.58)
Bouncing WiFi interface (workaround).
[ 64.398529] wl1283: down
[ 65.253967] wl1283: firmware booted (Rev 7.1.3.50.58)
udhcpc (v1.15.1) started
Sending discover...
[ 67.363342] wlan0: authenticate with 00:18:e7:db:a9:7d (try 1)
[ 67.378723] wlan0: authenticated
[ 67.464904] wlan0: associate with 00:18:e7:db:a9:7d (try 1)
[ 67.476470] wlan0: RX AssocResp from 00:18:e7:db:a9:7d (capab=0x431
status=0 aid=2)
[ 67.484680] wlan0: associated
[ 67.570617] wl1283: Association completed.
Sending discover...
Sending select for 192.168.0.195...
Lease of 192.168.0.195 obtained, lease time 86400
Deleting routers
adding dns 192.168.0.1
DM-37x#

```

Now the network is up and running. From this point on, all networking features (e.g., *ping*, *ifconfig*) are available just as in the wired case.

4.13.1.1 Scan for Available Wireless Networks

Once the wireless network has been initialized in station mode, you can scan for available networks using the command below.

```
DM-37x# iw dev wlan0 scan
```

4.13.2 Start Wireless Interface in AP Mode

As mentioned above, a script is used to start the wireless interface and additional details about the script can be reviewed using the *cat* command at the Linux shell prompt.

Below is an example of how to start the wireless network in AP mode. Run the script and answer the three subsequent questions using the information provided below. The script will start the wireless in AP mode, the wired network, the *UDHCPC* for the wireless network, and *DNSmasq*.

- SSID: *myssid*
- Encryption: *WPA2*
- Passphrase: *myssidpassphrase*

```

DM-37x# /etc/rc.d/init.d/network-wifi-ap init
Importing configuration variables from /etc/rc.d/rc.conf
Stopping hostapd
killall: hostapd: no process killed
Stopping dhcpd
killall: dhcpd: no process killed
Stopping dnsmasq
killall: dnsmasq: no process killed
Deleting iptable/chains from kernel:
Bring down eth0
ifdown: interface eth0 not configured
loading wl12xx sdio
[ 76.238311] wl12xx: loaded
[ 76.241149] wl12xx: initialized
Bring up eth0
[ 76.526428] smsc911x smsc911x.0: eth0: SMSC911x/921x identified at
0xd0876000, IRQ: 289
udhcpc (v1.15.1) started
Sending discover...
Sending discover...
Sending discover...
[ 84.254150] eth0: link up, 100Mbps, full-duplex
No lease, forking to background
Setup iptable forwarding between eth0 and wlan0:
Starting dnsmasq:
Assign wlan0 IP address 172.31.1.1
[ 87.219543] wl1283: firmware booted (Rev 7.1.3.50.58)
Starting the dhcp server on wlan0:
Internet Systems Consortium DHCP Server V3.0.3b1
Copyright 2004-2005 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/
Wrote 0 leases to leases file.
Listening on Socket/wlan0/172.31.1/24
Sending on Socket/wlan0/172.31.1/24
Sending on Socket/fallback/fallback-net
Stopping hostapd:
killall: hostapd: no process killed
hostapd: Missing /etc/hostapd.conf; re-creating
hostapd: Enter WiFi SSID to advertise: myssid
hostapd: Enter Encryption mode (NONE, WEP40, WEP128, WPA, WPA2): WPA2
hostapd: Enter WiFi WPA2 passphrase: myssidpassphrase
Starting hostapd:
Configuration file: /etc/hostapd.conf
[ 106.121765] wl1283: down
[ 107.089019] wl1283: firmware booted (Rev 7.2.0.0.47)
Using interface wlan0 with hwaddr 00:08:ee:05:7d:fc and ssid 'myssid'
DM-37x# [ 128.728332] NOHZ: local softirq pending 08
[ 129.435272] NOHZ: local_softirq_pending 08

```

```
[ 135.453948] NOHZ: local softirq pending 08
[ 159.473907] NOHZ: local softirq pending 08
[ 165.476837] NOHZ: local softirq pending 08
[ 165.493408] NOHZ: local softirq pending 08
[ 171.555816] NOHZ: local softirq pending 08
[ 171.635589] NOHZ: local softirq pending 08
[ 195.508026] NOHZ: local softirq pending 08
[ 201.517669] NOHZ: local_softirq_pending 08
```

4.13.3 Start Wireless Interface in Multi-Role

Note: This is only available with the Wireless Backports installed. The stock drivers are not capable of supporting multi-role. Make sure the `/etc/rc.d/rc.conf` has been modified per the backport instructions to change the `wl12xx_sdio` driver to `wlcore_sdio` instead.

As a suggestion, doing the following examples using `yaffs` instead of a RAMdisk will retain the generated files which make the configuration go by more quickly.

1. Go through the exercises for Access Point in 4.13.2 with NAT/Masquerade enabled to generate `hostapd.conf` file.
2. Go through the exercises for Station Mode found in 4.13.1 to generate the `/etc/wpa_supplicant.conf` file.
3. Once connected as station mode, add a new managed mode interface called `wlan1`

```
DM-37x# iw phy0 interface add wlan1 type managed
```

4. Configure `wlan1` as AP mode and set IP address of `wlan1`

```
DM-37x# ifconfig wlan1 10.4.30.34 netmask 255.255.255.0 up
```

5. Check the channel of the Wifi Channel of the `wlan0` station

```
DM-37x# iw dev wlan0 scan
```

6. Depending on the number of available WiFi networks, this list may vary. Look for the corresponding SSID of the network connection on `wlan0` then identify the *DS Parameter set: Channel X*
7. Edit the `/etc/hostapd.conf` file and exit the channel in `hostapd.conf` to match the value of `X` from step 6.
8. Change the interface on `hostapd.conf` to `interface=wlan1`
9. Save the `hostapd.conf` file
10. Invoke `hostapd` with the following:

```
DM-37x# hostapd -B /etc/hostapd.conf -P /var/run/hostapd.pid
```

11. Enable IP forwarding:

```
DM-37x# echo 1 > /proc/sys/net/ipv4/ip_forward
```

12. Edit /etc/udhcpd.conf
13. Change the interface from eth0 to wlan1
14. Save /etc/udhcpd.conf
15. Invoke udhcpd which will assign the IP addresses to the connecting devices

```
DM-37x# udhcpd /etc/udhcpd.conf
```

16. Start iptables with nat enabled.

```
DM-37x# iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
```

4.13.4 Setting Regulatory Domains Using the CRDA (Central Regulatory Domain Agent)

Users has the ability to changing regulatory domains in compliance with regulatory restrictions world wide. The BSP offers a userspace 'iw' applications to enable system designers to set their specific regulatory domain.

Select regulatory domain for United States

```
DM-37x# iw reg set US
```

Select regulatory domain for Canada

```
DM-37x# iw reg set CA
```

All others regulatory domain have been disabled. [Contact Logic PD](http://www.logicpd.com/contact/)²³ if your design needs support for additional regulatory domains outside the US and Canada.

Hostapd has a country_code option inside it, so setting this to US (default) sets CRDA automatically if you want to be an Access Point.

If you modify /etc/hostapd.conf and change the country_code=US to CA, you'll see CRDA switch to Canada. Since IC and FCC have the same limits and testing, there is really no difference between them.

4.14 Bluetooth

There are two Bluetooth Stacks available in the BSP. The following instructions in this guide are for the basic BlueZ stack enabled by default. For details on Bluetooth pairing, Connecting a Network Access Point (NAP), Audio Streaming, Human Interface Device or Serial over Bluetooth, users are suggested to see [Application Note 605: DM3730/AM3703 Torpedo + Wireless SOM Bluetooth with the DM37x Linux BSP.](#)

NOTE: Customers who have critical data to send over Bluetooth should ensure the SOM remains within the range of other Bluetooth devices. Due the volatile nature of wireless

²³ <http://www.logicpd.com/contact/>

signals and the potential for external influences, customers are also suggested to implement a retry mechanism in the event the SOM is unable to transmit data.

The Bluetooth controller has the same serial interface as the GPS controller. To use the Bluetooth device or the GPS device, the shared transport service must first be started.

1. With the *gpsdemo* application on your target, begin by adding the GPS module to the kernel.

```
DM-37x# modprobe st_drv
[ 124.591705] (stk) :sysfs entries created
[ 124.596252] (stk) : debugfs entries created
DM-37x#
```

2. Start the shared transport utility. Because the Bluetooth and GPS share the same interface to the wireless module, the shared transport utility manages traffic destined for Bluetooth or GPS.

```
DM-37x# /home/root/wl12xx/uim &
[1] 705
DM-37x#
```

3. Start the Bluetooth WiLink™ service.

```
DM-37x# modprobe btwilink
[ 80.523803] Bluetooth: Bluetooth Driver for TI WiLink - Version 1.0
[ 80.545806] (stc): st register(2)
[ 80.549346] (stc): chnl id list empty :2
[ 80.553771] (stk) : st kim start
[ 80.662017] (stk) :ldisc install = luim: Inside mainuim: Inside
st uart configuim:install set to 1
```

4.14.1 Start or Stop Bluetooth Interface

Use the command below to start the Bluetooth interface.

```
DM-37x# hciconfig hci0 up
```

Use the command below to stop the Bluetooth interface.

```
DM-37x# hciconfig hci0 down
```

4.14.2 Assign Hardware Name

Use the command below to assign a name to the hardware, substituting `<your_name>` with the identifying name you would like to use.

```
DM-37x# hciconfig hci0 name <your_name>
```

4.14.3 View Bluetooth Device Configuration

Use the command below to view the Bluetooth device configuration.

```
DM-37x# hciconfig -a hci0
```

4.14.4 Modify Bluetooth Device Configuration

Use the command below to modify the Bluetooth device configuration, using the hexadecimal identifier of your device class.

```
DM-37x# hciconfig hci0 class 0XXXXXXXXX
```

4.14.5 Scan for Bluetooth Devices

Use the command below to scan for remote Bluetooth devices.

```
DM-37x# hcitool scan
```

4.14.6 Query Bluetooth Device

Use the command below to query a specific Bluetooth device, substituting `XX:XX:XX:XX:XX:XX` with the six-byte MAC address of the device.

```
DM-37x# hcitool info XX:XX:XX:XX:XX:XX
```

4.15 USB Controller

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

```
DM-37x# /usr/sbin/lssusb
Bus 002 Device 001: ID 1d6b:0002
Bus 001 Device 002: ID 0471:3526 Philips
Bus 001 Device 001: ID 1d6b:0002
DM-37x# /usr/sbin/lssusb -v

Bus 002 Device 001: ID 1d6b:0002
Device Descriptor:
```

```

bLength          18
bDescriptorType   1
bcdUSB           2.00
bDeviceClass      9 Hub
bDeviceSubClass   0 Unused
bDeviceProtocol   1 Single TT
bMaxPacketSize0   64
idVendor          0x1d6b
idProduct         0x0002
bcdDevice         3.00
iManufacturer     3 Linux 3.0.0-BSP-dm37x-2.4-2 musb-hcd
iProduct          2 MUSB HDRC host driver
iSerial           1 musb-hdrc
bNumConfigurations 1
...

```

4.16 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 this peripheral is included in the demo images. The interface has been tested with various USB hubs, flash drives, keyboards, and mouse devices.

4.17 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. This interface has proven difficult to work with when run in OTG mode; therefore, Logic PD has chosen to build the demo images with the MUSB peripheral configured as either a dedicated host or dedicated device.

4.17.1 Use MUSB in Host Mode

When attempting to use the MUSB in host mode, it is important that you use a proper OTG cable with a real mini-A connector plugged into 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](http://www.lindy-usa.com/)²⁴ (PN 31634).

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

4.17.2 Use MUSB in Device Mode

The demo images include modules that support Ethernet and File-backed Storage Gadgets (FSG). To get started, load and boot the standard demo image.

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

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

²⁴ <http://www.lindy-usa.com/>

```
bash$ ifconfig usb0 <your network settings>
```

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

```
DM-37X# ifconfig usb0 <your network settings>
```

To use the device as a file-backed storage gadget, follow the steps below. This example assumes that you have an ext2fs or FAT filesystem on an SD/MMC card attached to `/dev/mmcblk0p1`.

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

```
DM-37x# umount /mnt/mmcblk0p1
```

1. Load the kernel module.

```
DM-37x# rmmod g_ether
DM-37x# modprobe g_file_storage file=/dev/mmcblk0p1
```

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 distributions such as Fedora and Ubuntu 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.

```
DM-37x# mount /dev/mmcblk0p1 /mnt/mmcblk0p1
DM-37x# ls /mnt/mmcblk0p1
```

4.18 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 UARTB and UARTC.

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.

```
DM-37x# stty 115200 < /dev/ttyO2
```


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

```
DM-37x# 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.

```
DM-37x# cat /dev/ttyO2
```

4.19 I2C

The DM37x Linux BSP supports the *i2cset* and *i2cget* commands. For example, to enable VPLL2 in the TPS65950, use the commands below.

```
DM-37x# i2cset -f -y 1 0x4b 0x8e 0x2e
DM-37x# i2cget -f -y 1 0x4b 0x8e
0x2e
DM-37x# i2cset -f -y 1 0x4b 0x91 0x5
DM-37x# i2cget -f -y 1 0x4b 0x91
0x05
```

The commands are reading/writing the PMIC at address group *0x4b* and from the internal registers *0x8e* and *0x91*.

4.20 SPI

The *spi-test* command uses the [Aardvark I2C/SPI Activity Board](http://www.totalphase.com/products/activity_board/)²⁵ as a SPI device. For additional details, see the *DM37x Linux BSP Software Test Plan* included in the DM37x Linux BSP download.

```
DM-37x# spi-test
```

Source code for *spi-test* can be obtained with the command below.

```
bash$ ./ltib -p spi-test -m prep
```

4.21 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.

²⁵ http://www.totalphase.com/products/activity_board/

To enable charging at 25 uA, use the command below.

```
DM-37x# i2cset -f -y 1 0x4b 0x6d 0x1c
```

For more information, see the [TPS65950 OMAP PMIC Technical Reference Manual](#).²⁶

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.

```
DM-37x# date 201202221649.15
Wed Feb 22 16:49:15 UTC 2012
DM-37x# hwclock -w
DM-37x# hwclock
Wed Feb 22 16:49:32 2012  0.000000 seconds
DM-37x#
#turn system off for a short time; the BSP will
#automatically use the PMIC stored RTC value when power returns.

#(after power up completes)
DM-37x# date
Wed Feb 22 16:51:09 UTC 2012
```

4.22 Analog-to-digital Converters

The TPS65950 has several analog-to-digital converters (ADCs) available for use. This sample script will report the value of the backup battery and main battery.

```
#To read backup battery voltage, charging must be enabled
i2cset -f -y 1 0x4b 0x6d 0x1c

#Have to write to GPBR1, located at I2C address 0x49, register 0x91
i2cset -f -y 1 0x49 0x91 0x90

#Turn on the ADC, write 0x1 to CTRL1
i2cset -f -y 1 0x4a 0x00 0x01

#Enable ADCIN, write 0xff to SW1SELECT_LSB
i2cset -f -y 1 0x4a 0x06 0xff

#Enable ADCIN, write 0xff to SW1SELECT_MSB
i2cset -f -y 1 0x4a 0x07 0xff

#Enable VBAT prescaler, write 0x2 to BCIC1L1.
i2cset -f -y 1 0x4a 0x97 0x02

#Start Conversion, write 0x20 to CTRL_SW1
```

²⁶ <http://www.ti.com/product/tps65950#technicaldocuments>

```

i2cset -f -y 1 0x4a 0x12 0x20

sleep 1

#ADC9
GPCH9_LSB=`i2cget -f -y 1 0x4a 0x49`
echo "GPCH9_LSB is $GPCH9_LSB"
GPCH9_MSB=`i2cget -f -y 1 0x4a 0x4a`
echo "GPCH9_MSB is $GPCH9_MSB"

#ADC12
GPCH12_LSB=`i2cget -f -y 1 0x4a 0x4f`
echo "GPCH12_LSB is $GPCH12_LSB"
GPCH12_MSB=`i2cget -f -y 1 0x4a 0x50`
echo "GPCH12_MSB is $GPCH12_MSB"

let ADC9=`printf %d $GPCH9_LSB`/64+`printf %d $GPCH9_MSB`*4
let ADC12=`printf %d $GPCH12_LSB`/64+`printf %d $GPCH12_MSB`*4
let VAL9=($ADC9*4399)/1000
let VAL12=($ADC12*5865)/1000
echo "ADC9  value is $ADC9 , $VAL9 millivolts      Backup Battery
voltage"
echo "ADC12 value is $ADC12 , $VAL12 millivolts    Main Battery Voltage"

```

Below is a sample execution of this script.

```

DM-37x# source adc-script
GPCH9_LSB is 0x80
GPCH9_MSB is 0x5a
GPCH12_LSB is 0x80
GPCH12_MSB is 0xa6
ADC9  value is 362 , 1592 millivolts      Backup Battery voltage
ADC12 value is 666 , 3906 millivolts      Main Battery Voltage

```

Below is a sample execution of this script after one hour.

```

DM-37x# source adc-script
GPCH9_LSB is 0xc0
GPCH9_MSB is 0x63
GPCH12_LSB is 0x00
GPCH12_MSB is 0x9e
ADC9  value is 399 , 1755 millivolts      Backup Battery voltage
ADC12 value is 632 , 3706 millivolts      Main Battery Voltage

```

4.23 BQ27000 Gas Gauge Support

To enable gas gauge support:

1. Enter the kernel configuration menu with `./ltib -c` and select *Configure the kernel*.
2. Add support for 1-wire device drivers under Device Drivers > Dallas's 1-wire support.

3. Add support for 1-wire master OMAP HDQ driver under Device Drivers > Dallas's 1-wire support > 1-wire masters > OMAP HDQ driver.
4. Add support for 1-wire slave BQ27000 under Device Drivers > Dallas's 1-wire support > 1-wire slaves > BQ27000 slave support.
5. Add support for the BQ27x00 family under Device Drivers > Power Supply Class Support > BQ27x00.
6. Add support for the generic PDA power driver under Device Drivers > Power Supply Class Support > Generic PDA/phone power driver.

To check the status of the gas gauge, look in the `/sys/class/power_supply/bq27000-battery` directory.

```
DM-37x# cd /sys/class/power_supply/bq27000-battery
DM-37x# ls
capacity          energy_now        time_to_empty_avg
charge_full       power            time_to_empty_now
charge_full_design present          time_to_full_now
charge_now        status           type
current_now       subsystem        uevent
cycle_count       technology       voltage_now
device            temp
DM-37x# cat voltage_now
3798000
DM-37x#
```

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](#)²⁷
- [AN 493 DM3730/AM3703 Torpedo SOM Design Checklist](#)²⁸
- [AN 498 DM3730/AM3703 Torpedo + Wireless SOM Design Checklist](#)²⁹

4.24 Smart Reflex

SmartReflex is a power-management technique provided by TI for automatic control of the operating voltage of a module to reduce active power consumption. SmartReflex achieves the optimal performance/power trade-off for all devices across the technology.

The scripts below can be used to start and stop SmartReflex. SmartReflex is enabled by default.

Start:

```
DM-37x# /etc/rc.d/init.d/smartreflex start
```

Stop:

²⁷ <http://support.logipd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=696>

²⁸ <http://support.logipd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=586>

²⁹ <http://support.logipd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=637>

```
DM-37x# /etc/rc.d/init.d/smartreflex stop
```

4.25 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.
- 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.
- Suspend – The suspend state can be directed by pressing the S2 button on the DM3730 Development Kit baseboard. The DM37x Linux BSP can also enter suspend mode from the command line using the command below.

```
DM-37x# echo mem > /sys/power/state
```

To exit the suspend state, press the S2 button, tap the LCD screen, or type a key at the shell prompt.

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](#)³⁰
- [WP 491 DM3730/AM3703 Torpedo SOM Thermal Management](#)³¹
- [WP 530 DM3730/AM3703 Torpedo + Wireless SOM Thermal Management](#)³²

4.26 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.

4.26.1 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.

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

³⁰ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=717>

³¹ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=605>

³² <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=656>

```
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.

4.26.2 */sys/kernel/debug* Directory

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; however, a few examples are provided in the sections below.

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.

4.26.2.1 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/clock* 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.

```
DM-37x# cat
/sys/kernel/debug/clock/virt_26m_ck/osc_sys_ck/sys_ck/dpll1_ck/rate
600000000
DM-37x# cat
/sys/kernel/debug/clock/virt_26m_ck/osc_sys_ck/sys_ck/dpll2_ck/rate
260000000
DM-37x# cat
/sys/kernel/debug/clock/virt_26m_ck/osc_sys_ck/sys_ck/dpll3_ck/rate
400000000
DM-37x# cat
/sys/kernel/debug/clock/virt_26m_ck/osc_sys_ck/sys_ck/dpll4_ck/rate
864000000
DM-37x# cat
/sys/kernel/debug/clock/virt_26m_ck/osc_sys_ck/sys_ck/dpll5_ck/rate
120000000
```

4.26.2.2 Pin Mux

Below is an example in which we view the pin mux of the CPU. Specifically, we display the current configuration of the *sys_boot0* and *cam_d7* pins.

```
DM-37x# cat /sys/kernel/debug/omap_mux/sys_boot0
name: sys_boot0.gpio 2 (0x48002a0a/0x9da = 0x011c), b ah26, t NA
mode: OMAP PIN INPUT PULLUP | OMAP MUX MODE4
signals: sys_boot0 | NA | NA | dss_data18 | gpio 2 | NA | NA |
safe_mode
```

```
DM-37x# cat /sys/kernel/debug/omap_mux/cam_d7
name: cam d7.cam d7 (0x48002124/0x0f4 = 0x0100), b 128, t NA
mode: OMAP PIN INPUT | OMAP MUX MODE0
signals: cam d7 | NA | NA | NA | gpio 106 | NA | NA | safe mode
```

The output above shows the GPIO number each pin is associated with, the pull up state, the configuration mode, etc. See the CPU reference manual pin mux section for full details of each field.

4.26.2.3 General Purpose Bus Configuration

In the example below, we use the virtual file *debug* directory to view the GPMC configuration of each chip select.

```
DM-37x# cat /sys/kernel/debug/omap_gpmc
CONFIG: 00000210
STATUS: 00000b01
IRQSTATUS: 00000000
IRQENABLE: 00000000
CS0: 00001800 00090900 00090902 07020702
      0008080a 000002cf 00000f70
CS1: 00001000 00080801 00000000 08010801
      00080a0a 03000280 00000f48
CS2: 6a411213 000c1503 00050503 0b051506
      020e0c15 0b0603c3 00000f50
CS3: 00001210 00131000 001f1f01 10030e03
      010f1411 80030600 00000f58
```

See GPMC register definition section in the CPU reference manual for further details.

4.27 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.

4.28 Additional Peripheral Test Information

Additional information on testing peripherals and device drivers can be found in the *DM37x Linux BSP Software Test Plan* included in the DM37x Linux BSP download. Please post a question to the Logic PD [TDG forum](#) for additional details.

4.29 CPU Benchmarks

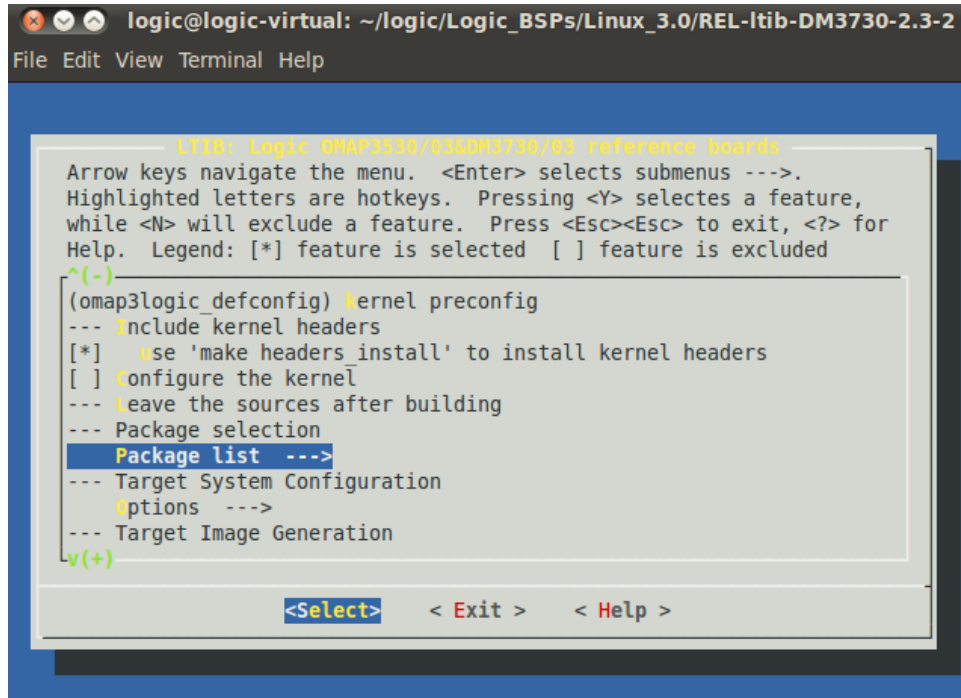
Adding the miscellaneous benchmark package will add dhrystone, whetstone, and linpack benchmark tools to the */usr/bin* sub-directory.

To set CONFIG_PKG_MISC_BENCHMARKS=y using the LTIB configuration menu, follow the steps below.

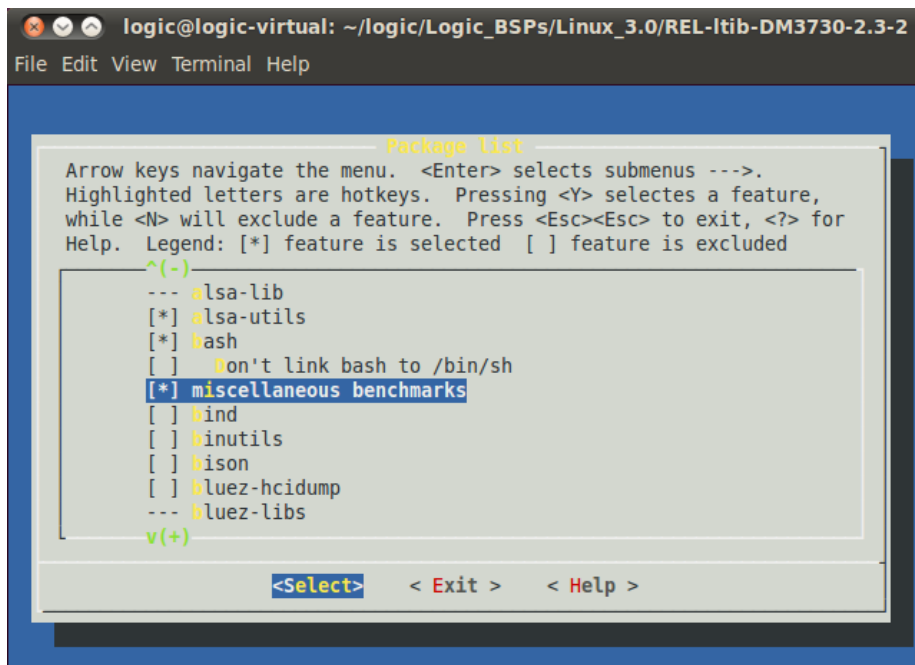
1. At the bash prompt, enter the following command to start the LTIB menu system.

```
bash$ ./ltib -c
```

2. In the main menu, select Packages list.



3. In the *Package list* window, select miscellaneous benchmarks.



4. Finally, save your changes by exiting each sub menu. Wait for LTIB to build your images.

Below are example commands for running the miscellaneous benchmarks.

```
DM-37x# cd /usr/bin
DM-37x# 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

...

Str_2_Loc:          DHRYSTONE PROGRAM, 2'ND STRING
                should be:  DHRYSTONE PROGRAM, 2'ND STRING

Microseconds for one run through Dhrystone:      1.0
Dhrystones per Second:                          1039861.4

DM-37x# whetstones 5000

Loops: 5000, Iterations: 1, Duration: 3.77607 sec.
C Converted Double Precision Whetstones: 132.4 MIPS
DM-37x# linpack
Rolled Double Precision Linpack

Rolled Double Precision Linpack

      norm. resid      resid      machep      x[0]-1      x[n-
1]-1
      1.7             7.41628980e-14  2.22044605e-16 -1.49880108e-14 -
1.89848137e-14
      times are reported for matrices of order 100
      dgefa      dgesl      total      kflops      unit      ratio
times for array with leading dimension of 201
      0.05      0.01      0.06      10987      0.18      1.12
      0.06      0.00      0.06      10987      0.18      1.12
      0.05      0.00      0.05      12556      0.16      0.98
      0.03      0.00      0.03      26634      0.08      0.46
times for array with leading dimension of 200
      0.02      0.00      0.02      29297      0.07      0.42
      0.02      0.00      0.02      29298      0.07      0.42
      0.02      0.00      0.02      29297      0.07      0.42
      0.02      0.00      0.02      33805      0.06      0.36
Rolled Double Precision 26634 Kflops ; 10 Reps
DM-37x#
```

4.30 Use *Peekpoke* to Examine/Modify Registers

LTIB has the *peekpoke* package included. *Peekpoke* 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.

```
DM-37x# peekpoke -c 1 -l -r 0x4830A204
4830a204 /l -> 2b89102f
DM-37x#
```

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

```
DM-37x# peekpoke -l -w 0x48307250 0x2

Texas Instruments X-Loader 1.42 BSP-dm37x-2.3-2 for dm3730logic (2013-
01-16 14:43:47)
DRAM: 256MB (ProductID defined)
Starting U-boot on MMC
Reading boot sector
454256 bytes read from MMC to 80400000

U-Boot 2011.06 BSP-dm37x-2.3-2 (Jan 16 2013 - 14:42:37)

OMAP3630/3730-GP ES2.1, CPU-OPP2, L3-200MHz, Max CPU Clock 1 Ghz
Logic DM37x/OMAP35x reference board + LPDDR/NAND
...
```

On your Linux host PC, the command below will extract the source for the *peekpoke* package in the *rpm/BUILD/peekpoke-1* sub-directory.

```
bash$ ./ltib -p peekpoke -m prep
```

4.31 Filesystem Commands

This section provides useful filesystem commands.

4.31.1 *df* Command

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

```
DM-37x# df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/root              57479        36964      17618   68% /
tmpfs                 120412         48      120364    0% /dev
```

```

/dev/mmcblk0p1      1961952      61632      1900320      3% /mnt/mmcblk0p1
shm                 120412          0      120412      0% /dev/shm
rwfs                 512          0          512      0% /mnt/rwfs
DM-37x#

```

4.31.2 *cat /proc/mtd* Command

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

```

DM-37x# cat /proc/mtd
dev:      size  erasesize  name
mtd0: 00080000 00020000 "x-loader"
mtd1: 001a0000 00020000 "u-boot"
mtd2: 00060000 00020000 "u-boot-env"
mtd3: 00500000 00020000 "kernel"
mtd4: 01400000 00020000 "ramdisk"
mtd5: 1e480000 00020000 "fs"
DM-37x#

```

4.31.3 *flash_eraseall* Command

The *flash_eraseall* command erases a filesystem partition. In the example below, the U-Boot environment variables are erased. Care must be taken when running from a YAFFS root filesystem in NAND to not erase the root partition - this will cause the system to crash.

```

DM-37x# flash_eraseall /dev/mtd2
Erasing 128 Kibyte @ 60000 -- 100% complete.
DM-37x#

```

4.31.4 *badblocks* Command

The *badblocks* command searches a device for bad blocks. In the example below, the filesystem NAND partition is searched for bad blocks.

```

DM-37x# badblocks -v /dev/mtdblock5
Checking blocks 0 to 496127
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
DM-37x#

```

4.32 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.

Table 4.32:AM/DM37xx Regulator Voltages

Regulator Path	Name	Min (microvolts)	Max (microvolts)	Voltage (microvolts)
twl_reg.17/regulator/regulator.2	VUSB1V5			1500000
twl_reg.18/regulator/regulator.3	VUSB1V8			1800000
twl_reg.19/regulator/regulator.4	VUSB3V1			3100000
twl_reg.0/regulator/regulator.5	vdd_mpu_iva	600000	1450000	
twl_reg.1/regulator/regulator.6	vdd_core	600000	1450000	
twl_reg.6/regulator/regulator.7	VMMC1	1850000	3150000	
twl_reg.3/regulator/regulator.8	VDAC	1800000	1800000	
twl_reg.5/regulator/regulator.9	VDVI	1800000	1800000	
twl_reg.7/regulator/regulator.10	VMMC2	1850000	3150000	
twl_reg.9/regulator/regulator.11	VAUX1	3000000	3000000	
twl_reg.12/regulator/regulator.12	VAUX3	2800000	2800000	
twl_reg.13/regulator/regulator.13	vaux4	1800000	1800000	

To read the names from Linux:

```
DM-37x# cat /sys/devices/platform/omap/omap_i2c.1/i2c-1/1-004b/twl_reg.6/regulator/regulator.7/name
VMMC1
DM-37x#
```

To read the current voltage for a given regulator:

```
DM-37x# cat /sys/devices/platform/omap/omap_i2c.1/i2c-1/1-004b/twl_reg.6/regulator/regulator.7/microvolts
3150000
DM-37x#
```

To read the minimum voltage setting for given regulator:

```
DM-37x# cat /sys/devices/platform/omap/omap_i2c.1/i2c-1/1-004b/twl_reg.6/regulator/regulator.7/min_microvolts
3150000
DM-37x#
```

To read the maximum voltage setting for a given regulator:

```
DM-37x# cat /sys/devices/platform/omap/omap_i2c.1/i2c-1/1-  
004b/twl_reg.6/regulator/regulator.7/max_microvolts  
3150000  
DM-37x#
```

To check the state of the regulator:

```
DM-37x# cat /sys/devices/platform/omap/omap_i2c.1/i2c-1/1-  
004b/twl_reg.6/regulator/regulator.7/state  
enabled  
DM-37x#
```

4.33 Reading Temperature Sensor

The internal temperature sensor is not necessarily calibrated and shows the temperature of the die, not the ambient temperature. Temperature ranges may vary with processor voltage and load.

Read the temperature of the processor die:

```
DM-37x# cat /sys/devices/platform/temp_sensor/Celsius  
27.7  
DM-37x#
```

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

This section explains how to configure both a Ubuntu 12.04 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 12.04 environment. These steps have been tested using the Virtual Machine SDK for the DM37x Linux BSP v2.4-2 and the DM37x Linux BSP v2.4-2. Other versions may require slight changes to the steps below.

5.1 Set Up TFTP Server in Ubuntu 12.04

This section describes the steps needed to install and run the TFTP server in Ubuntu 12.04.

1. Install a TFTP server in Ubuntu 12.04.

```
bash$ sudo apt-get install xinetd tftpd tftp
```

2. Create the TFTP configuration file.

```
bash$ sudo gedit /etc/xinetd.d/tftp
```

3. Add the following content to the TFTP configuration file.

```
service tftp
{
  protocol          = udp
  port              = 69
  socket type       = dgram
  wait              = yes
  user              = nobody
  server            = /usr/sbin/in.tftpd
  server args       = /tftpboot
  disable           = no
}
```

4. Create the */tftpboot* folder. This must match the folder assigned to *server_args* in the TFTP configuration file in the step above.

```
bash$ sudo mkdir /tftpboot
bash$ sudo chmod -R 777 /tftpboot
bash$ sudo chown -R nobody /tftpboot
```

5. Restart the xinetd service.

```
bash$ sudo /etc/init.d/xinetd restart
```

Now the TFTP server is up and running.

6. Copy the kernel build into */tftpboot*.

```
bash$ cp ~/logic/Logic_BSPs/Linux_3.0/REL-ltib-DM3730-2.4-2/rootfs/boot/uImage /tftpboot/uImage
```

5.2 Setup NFS Server in Ubuntu 12.04

This section describes the steps needed to install and run the NFS server in Ubuntu 12.04. In addition, steps are provided that explain how to move the files from the build root filesystem to the NFS root filesystem.

1. Set up a directory for the NFS server.

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

2. Populate the *ltib-omap* directory with the content of the root filesystem directory in the LTIB tree.

```
bash$ sudo tar -C ~/logic/Logic_BSPs/Linux_3.0/REL-ltib-DM3730-2.4-2/rootfs -cf - . | sudo tar -C /opt/nfs-exports/ltib-omap -xf -
```

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

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

4. 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/ltib-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/ltib-omap*. You can change the IP range/netmask bits to whatever you need; the options for doing so are described in more detail [here](http://linux.die.net/man/5/exports).³³

- Restart the nfs-kernel-server service.

```
bash$ sudo /etc/init.d/nfs-kernel-server restart
```

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

```
bash$ 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 12.04 Linux host PC.

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

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

- Get a DHCP IP address and set the *ipaddress* variable.

```
OMAP Logic # run get_dhcp_address
```

- Set the IP address of the TFTP and NFS server to access the kernel and root filesystem. In this example, the TFTP and NFS server is at 192.168.120.53. This IP address must be the same as the one on your Linux host PC running the TFTP and NFS servers. At this time, both the TFTP and NFS services must reside on the same Linux host PC.

```
OMAP Logic # setenv serverip 192.168.120.53
```

- Set the IP address of the DM3730/AM3703 target system. The IP address in this example was set to 192.168.120.64.

```
OMAP Logic # setenv ipaddr 192.168.120.64
```

- Configure U-Boot to load the kernel from a TFTP server.

```
OMAP Logic # setenv kernel_location tftp
```

- Configure U-Boot to find the kernel. Note that it does not have a leading slash but requires a trailing slash. If you have the kernel in the root of the TFTP server's

³³ <http://linux.die.net/man/5/exports>

directory (i.e., */tftpboot*), then *setenv tftpdir* suffices. If it is in a subdirectory like */tftpboot/foo/bar*, then you'll need *setenv tftpdir foo/bar/*.

```
OMAP Logic # setenv <tftpdir>
```

7. Configure U-Boot for the root filesystem on an NFS.

```
OMAP Logic # setenv rootfs_location nfs; setenv rootfs_type nfs
```

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

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

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

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

```
OMAP Logic # saveenv
```

10. Execute the following command to boot 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 # run nfsboot
```

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 glib 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.

6.1 "Hello World" Application Example

6.1.1 Build "Hello World" Application

Before starting to build the "Hello World" application, be sure you can build the BSP using the default configuration described in Section 2.4.

1. Unpack the "Hello World" application. Since the application is not part of Logic PD's DM37x Linux BSP distribution, a network connection on your host PC is necessary to download it from the global package pool. LTIB will automatically detect when a package is not available in the local BSP and download it if necessary.
2. Download the "Hello World" source package and install it in */rpm/BUILD/helloworldx-x*.

```
bash$ ./ltib -p helloworld -m prep
```

3. Next, build and deploy the application with the command below.

```
bash$ ./ltib -p helloworld -m scbuild && ./ltib -p helloworld -m
scdeploy
```

The "Hello World" application is a simple example. By reviewing the source and the Makefile, you can see what is needed for an application to compile and link against the kernel.

NOTE: Alternatively, you can use the *./ltib -m* shell command to switch the environment to the environment LTIB uses when performing the build. From the LTIB shell, you can build your application as if your environment was set up specifically for your "Hello World" application.

6.1.2 Transfer "Hello World" Application to Root Filesystem

In the build example above, the second part of the last command (*./ltib -p helloworld -m scdeploy*) will package up the "Hello World" application in the root filesystem, along with all the other applications and packages already in the filesystem. The location of the "Hello World" application is in */usr/bin*. If booting from SD and using a RAM-based root filesystem,

simply copying *rootfs.ext2.gz.uboot* from the LTIB build directory to the SD card will include the “Hello World” application.

6.1.2.1 Transfer “Hello World” Application using TFTP

With the SOM at the Linux prompt, the “Hello World” application can be transferred to the SOM and executed without the need to copy over the entire root filesystem. There are countless ways to transfer the executable to the SOM. For this example, we will use TFTP.

There are many applications available to serve TFTP files. On a Linux desktop, *atftpd* is a popular TFTP server daemon.

1. Start by copying *rpm/BUILD/helloworld-x.x* to the TFTP server directory. This will make the “Hello World” application available to TFTP clients.
2. On the SOM, bring up Ethernet using the information provided in Section 4.2, 4.12, or 4.13.1, depending on your hardware.
3. With Ethernet running, use the command below at the SOM Linux prompt to transfer the file over to the SOM in the current directory. Be sure to replace *<server ip>* with the IP address of the TFTP server.

```
DM-37x# tftp -l hello -g <server ip>
```

4. Since TFTP does not preserve the file permissions, the executable permission must be set. Use the command below to set the “Hello World” application permissions to be executable by all.

```
DM-37x# chmod ugo+x hello
```

5. Execute the “Hello World” application by typing the application name at the prompt.

```
DM-37x# ./hello
Hello world
DM-37x#
```

6.1.3 Run “Hello World” Application

Execute the “Hello World” application by typing the application name at the prompt.

```
DM-37x# hello
Hello world
DM-37x#
```

Note that the file can also be executed using */usr/bin/hello*. However, since */usr/bin* is in the default *\$PATH* environment variable, the full path does not need to be specified.

6.2 GPS Demo Application Example

6.2.1 Build GPS Demo Application

The Logic PD DM37x Linux BSP includes a "Hello World"-like example application that demonstrates how to write an application using the GPS unit. The following steps explain how to unpack the source code, view and modify (prep) the package, and build.

NOTE: The GPS feature was relatively new at the time this document was written. If you are using the DM37x Linux BSP version 2.2-2 or older, please post a question to the Logic PD [TDG forum](#) to obtain the package or simply download a newer version of the DM37x Linux BSP (see Section 2 for a link).

1. Prep the GPS demo application source package and install it in `/rpm/BUILD/testGPS-X.X`.

```
bash$ ./ltib -p gpsdemo -m prep
```

2. Next, build and deploy the application.

```
bash$ ./ltib -p gpsdemo -m scbuild && ./ltib -p gpsdemo -m scdeploy
```

The GPS demo application is a simple example to display GPS data. By reviewing the source and the Makefile, you can see what is needed to support GPS in your application.

NOTE: As an alternative approach to the above commands, you can enter the `./ltib -c` command, select "Package list," and choose the "GPS demo" package to include the GPS demo application in your build. This will not provide source code for you to view and modify; however, it will provide a runtime binary in your filesystem for you to use. See Section 2.5 for more information on the `./ltib -c` command and on including packages in your build.

6.2.2 Transfer GPS Demo Application to Target

The GPS demo application is included in the root filesystem using the LTIB `scdeploy` option.

To transfer the GPS demo application to the target, you can update the root filesystem on your target using the procedures in Section 2.4.2. You can also follow the "Hello World" example in Section 6.1.2 to push only the GPS demo image to the target.

6.2.3 Run GPS Demo Application

1. With the GPS demo application on your target, begin by adding the GPS module to the kernel.

```
DM-37x# modprobe gps_drv
[ 124.591705] (stk) :sysfs entries created
[ 124.596252] (stk) : debugfs entries created
DM-37x#
```

2. Start the shared transport utility. Because the Bluetooth and GPS share the same interface to the wireless module, the shared transport utility manages traffic destined for Bluetooth or GPS.

```
DM-37x# /home/root/wl12xx/uim &
[1] 705
DM-37x#
```

3. Configure Log files. The logging is enabled by default to aid in development and testing. To keep the logging, skip step 3. For users who plan to use GPS for extended period of time, having logging enabled is not recommended, because it will fill up the file system. To disable logging:

Edit /system/etc/gps/config/pathconfigfile.txt

Locate the following entries and change them to the following to disable logging

```
SESSION_LOG_CONTROL 0
SENSOR_CONTROL 0
```

4. Start the *navd* service. The *navd* service manages the *GPS.h* Application Programming Interface (API) to forward GPS commands and handles updates to the GPS application.

NOTE: Disregard the *Could not set new working dir* warning. This warning does not hinder the GPS operation.

```
DM-37x# navd --android_log NAVD -p3 \
-nav\"-c/system/etc/gps/config/pathconfigfile.txt\" &
[2] 706
cmd line navd --android log NAVD -p3 -nav"-
c/system/etc/gps/config/pathconfigfile.txt"
MCP | initializing sighandler
could not set new working dir: No such file or directoryMCP | main |
starting...
Note: this task requires root privileges
```

5. Finally, launch the GPS demo application.

```
DM-37x# gpsdemo
[ 165.513885] (stc): st register(9)
[ 165.517761] (stc): chnl id list empty :9
[ 165.522247] (stk) : st kim start
[ 165.631225] (stk) :ldisc install = 1uim: Inside mainuim: Inside
st uart configuim:install set to 1
uim:opening /dev/ttyO1, while already open
uim:
[ 165.648864] (stc): st tty open cleanup
uim: In
[ 165.653625] (stk) : line discipline installed side set baud rateuim:
set baud rate() done
uim
[ 165.663116] (stk) :Logic TIIInit 10.6.15.bts: Installed N TI WL Line
displine
```

The GPS demo application will display GPS data continuously on the terminal. Use **Ctrl+C** to exit the application.

NOTE: The GPS demo application works best with a terminal width of 103 characters.

NOTE: The GPS antenna must have a clear, unobstructed view of the sky. If the GPS is unable to get a lock, reorient the antenna for better GPS reception.

6.3 Debug with GNU Debugger

The GNU debugger (GDB) is included in the DM37x Linux BSP tool set. Below is an example of how to use the GDB to debug the "Hello World" sample application.

6.3.1 Configure Build Options

When debugging an application, the compiled application must include a symbol table so the debugger can identify memory locations by the variable names indicated in the source. For the "Hello World" application, we need to tell the compiler to include the symbol table in the output file.

In addition, it is far simpler to trace through a program when the compiler has not optimized the result assembly. The compiler has many optimizations that, when employed, often show an execution path that does not follow the C source (or whatever language may be used for the source). We need to tell the compiler to turn off optimizations before debugging.

For the "Hello World" example, we need to use a text editor to edit the compiler options in *rpm/BUILD/helloworld-x.x/Makefile*. In this file, locate the line starting with *CFLAGS=*. This line sets the compiler options and there will likely be one or more options already there. We will need to be sure to include the options *-ggdb* and *-O0*. The first option will include debugging symbols, while the second option turns off compiler optimizations. With the file updated, the "Hello World" application can be built using the procedures in Section 6.1.1.

6.3.2 Set Up GDB

With the image built and located on the SOM, we can start the GDB.

1. Begin by bringing up the Ethernet interface as indicated in Sections 4.2, 4.12, or 4.13 depending on your hardware. Note the SOM IP address.
2. At the Linux prompt on the SOM, enter the command below.

```
DM-37x# gdbserver :3000 hello
Process hello created; pid = 736
Listening on port 3000
```

The GDB server manages the remote debugging session on the SOM. The user interface for debugging takes place on a host PC with the "Hello World" source and the GDB. The *:3000* argument indicates the Ethernet port number to use. The port number is somewhat arbitrary, as long as the port number is unique and common between the GDB server and GDB debugger on the host PC.

At this point, the "Hello World" application is ready with the program counter stopped at the first line of the source. All other processes on the SOM continue to run.

3. From the "Hello World" source code directory on your host PC, start the GDB with the command below.

```
Ubuntu: /opt/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-
gdbtui
```

You will see two windows open in your Linux terminal. The upper window will show the source code currently being debugged. The lower window will show the GDB command history.

The GDB command set is large; typing *help* at the GDB prompt can help narrow down what command you need to use when debugging your programs. Additional information on GDB can be found on the [GDB: The GNU Project Debugger wiki page](http://www.gnu.org/software/gdb/documentation/).³⁴

4. Use the command below at the GDB prompt to establish the connection to the GDB server on the SOM, where *<ip>* is the IP address of the SOM.

```
(gdb) target remote <ip>:3000
Remote debugging using <ip>:3000
0x400fb7b0 in ?? ()
(gdb)
```

5. Next, load the "Hello World" symbol file at the GDB prompt.

```
(gdb) symbol-file hello
Reading symbols from
/media/extpart sdb5 500GB/svn sandbox/eps svn/software/prod
ucts/linux/LTIB/trunk/ltib-20101216/rpm/BUILD/helloworld-
1.1/hello...done.
(gdb)
```

Now the symbols are loaded.

6. Use the GDB *list* command to update the top window with the current location of the program counter in the "Hello World" source code.

```
(gdb) list
(gdb)
```

You should now see the "Hello World" source code displayed in the top window.

NOTE: The "Hello World" application is small, so the entire source can be displayed. For larger applications, however, the upper window can be used to display a portion of that application source code. The GDB *list* command by itself will display the source at the current program counter. You can use the *list <file>:<line>* command, where *<file>* is the file name you want displayed and *<line>* is the line number you want to show in the window.

7. At this point, the "Hello World" application has not started executing. To demonstrate the use of GDB, we will set a break point just after printing "hello world" to the screen. At the GDB prompt, set a break point and have it indicated on the upper window.

```
(gdb) break hello.c:6
```

³⁴ <http://www.gnu.org/software/gdb/documentation/>

```
Breakpoint 1 at 0x8438: file hello.c, line 6.
```

8. Next, start the “Hello World” application and verify it stops at line 5.

```
(gdb) c
Continuing.

Breakpoint 1, main () at hello.c:6
```

The command window indicates the program has stopped at line 6. Likewise, the top window shows line 6 highlighted, which indicates the stopped state of the program counter.

If we had variables in our “Hello World” application, we could use the GDB *print* command to display the content of those variables. The values of variables within the lexical environment of the current stack frame can be displayed, plus any global variables.

At this point, you may notice the “Hello World” string is not displayed on the Linux terminal of the SOM. But, the program counter shows that the *printf* (“hello world”) statement has been executed. This occurs because the STDIO device used by the *printf*() function has not flushed its buffer to the terminal. The strings printed to the terminal will be flushed when the STDIO device is explicitly flushed, closed, or the application program terminates. At that point, all the *printf*() text remaining in the STDIO buffer will be sent to the Linux terminal display.

9. To let the “Hello World” application continue, use the GDB *c* command again.

```
(gdb) c
Continuing.

Program exited normally.
```

As noted earlier, with the “Hello World” application running to completion, you will see the SOM Linux terminal show “hello world” and the GDB server will report the status of the application.

```
hello world

Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
DM-37x#
```

10. Finally, use the *quit* command on the host PC to end the GDB session.

7 Loadable Module Development

There will likely be cases where you want to create a driver to be used by the Linux kernel. This section describes how to create a “Hello World” loadable module that is linked into the kernel at run time.

7.1 “Hello World” Module

It is beyond the scope of this document to explain how to write a driver for the Linux kernel. However, you can see how a module is created, built, installed, and removed by using the “Hello World” module.

7.1.1 Build “Hello World” Module

1. Begin by unpacking the *hello mod* application using the command below from the BSP directory. This will download the source and unpack it in the *rpm/BUILD/hello_mod-x* directory.

```
bash$ ./ltib -p hello_mod -m prep
```

2. Next, build the module.

```
bash$ ./ltib -p hello_mod
```

The resulting module is included in the root filesystem. From the BSP build directory, the module can be found in *rootfs/lib/modules/<kernel version>/misc/modexample.ko*. The built module can be loaded on the SOM by copying the root filesystem to the SOM and restarting, or by using TFTP (see the “Hello World” application example in Section 6.1).

7.1.2 Run “Hello World” Module

1. With the “Hello World” module (named *modexample*) located in the root filesystem in */lib/modules/<kernel version>/misc*, the module can be loaded using the *modprobe* command.

```
DM-37x# modprobe modexample
[102054.804199] say hello
```

Our “Hello World” module does not do much more than print *say hello*. However, you can see its operation upon loading the module.

2. Verify the module is loaded.

```
DM-37x# lsmod
Module                Size  Used by
modexample             787    0
g_ether               57800  0
```

3. Unload the module.

```
DM-37x# rmmod modexample  
[102241.063415] wave goodbye
```

By removing the module, we see the module print *wave goodbye*.

4. Verify the module is no longer loaded.

```
DM-37x# lsmod  
Module                Size  Used by  
g ether                57800  0  
DM-37x#
```

8 Cameras and DSP (DVSDK)

The DSP requires software and hardware overhead to operate and is not enabled by default. This section explains how to enable the DSP in your build. The steps in this section assume you have completed a successful build using the default configuration described in Section 2.4.

Note: Systems designed to use the DSP drivers may not properly function following suspend/resume states. Customers looking for both DSP support and suspend/resume functionality can [contact Logic PD](#) for additional design support.

8.1 Prepare Build Tools

Using the DSP requires the Code Generation Tools (CGT) provided by Texas Instruments (TI).

1. First, verify you can build the DM37x Linux BSP by following the steps outlined in Section 2.4.
2. Next, download version 6.1.14 of the CGT from one of the TI links below. **NOTE:** You must have a my.TI account set up in order to access these downloads.
 - http://software-dl-1.ti.com/dsps/forms/self_cert_export.html?prod_no=ti_cgt_c6000_6.1.14_setup_linux_x86.bin&ref_url=http://software-dl.ti.com/dsps/dsps_registered_sw/sdo_ccstudio/codegen/C6000/6.1.14
 - https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm

Other versions may or may not work with your BSP and Logic PD may not be able to provide support if a different version is used. The above links also include additional documentation on the TI code generation tools.

3. Once downloaded, use the commands below to run the installer. Follow the prompts to install CGT to the default directory `/opt/TI/C6000CGT6.1.14/`.

```
bash$ chmod a+x ~/Downloads/ti_cgt_c6000_6.1.14_setup_linux_x86.bin
bash$ sudo ~/Downloads/ti_cgt_c6000_6.1.14_setup_linux_x86.bin
```

4. Configure LTIB to build the DVSDK packages.
 - a. Configure LTIB. See Section 2.5.4 for more information.

```
bash$ ./ltib -c
```

- b. In the main menu, under the *Target Image Generation* heading, choose Options.

```

eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

LTIB: Logic OMAP3530/03&DM3730/03 reference boards
Arrow keys navigate the menu. <Enter> selects submenus ---. Highlighted
letters are hotkeys. Pressing <Y> selects a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded

^(-)
[ ] Produce cscope index
(omap3logic_defconfig) kernel preconfig
--- Include kernel headers
[*] Use 'make headers_install' to install kernel headers
[ ] Configure the kernel
--- Leave the sources after building
(eric) SCM username to replace '%s' in REPOSITORY variables
--- Package selection
    Package list --->
--- Target System Configuration
    Options --->
--- Target Image Generation
    Options --->
(BSP-dm37x-2.0-trunk) BSP Release Level (supplied by platform defconfig)
---
Load an Alternate Configuration File
v(+)

<Select>    < Exit >    < Help >
  
```

- c. Under the *Choose your root filesystem image type* heading, choose Target image.

```

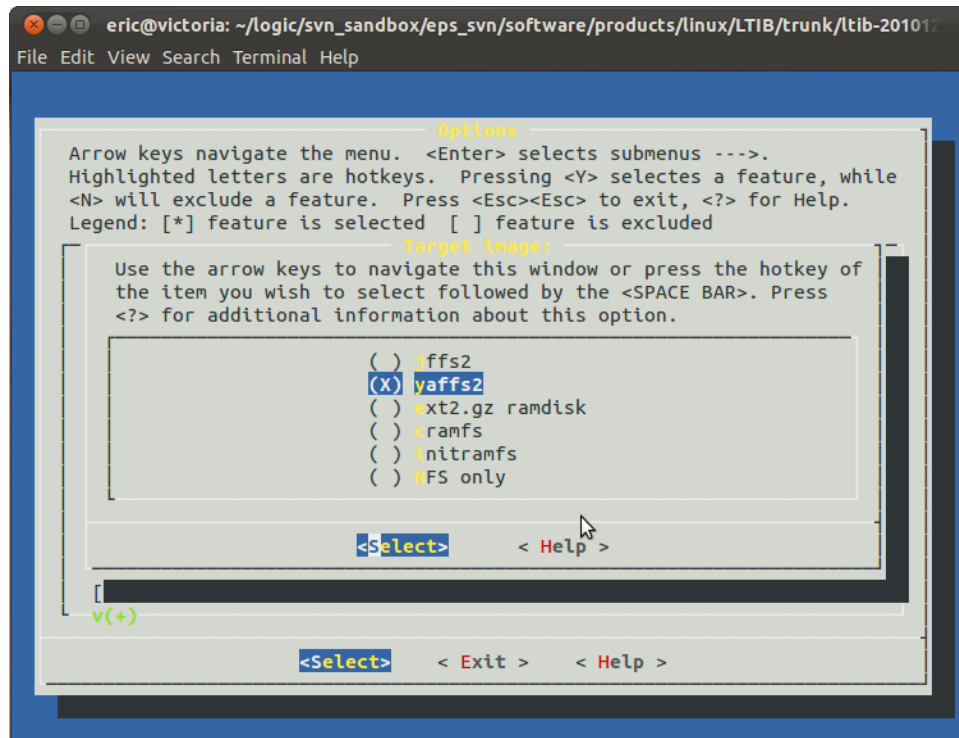
eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

Options
Arrow keys navigate the menu. <Enter> selects submenus ---. Highlighted
letters are hotkeys. Pressing <Y> selects a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded

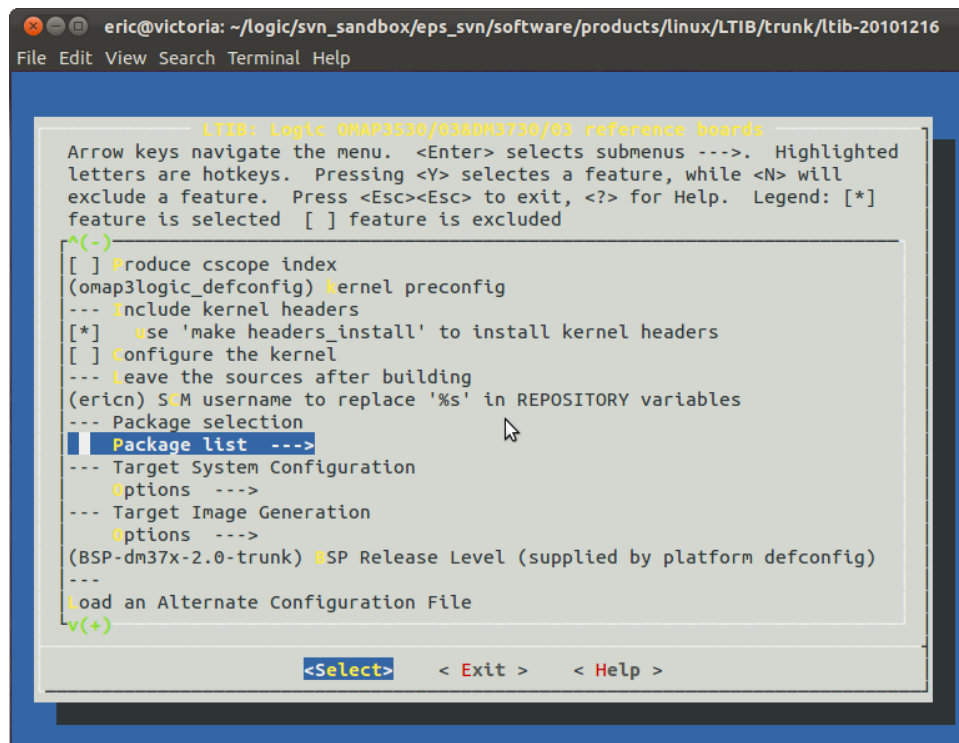
--- Choose your root filesystem image type
    Target image: (ext2.gz ramdisk) --->
[ ] Create a combined ELF image
[ ] Run a command after building the target image
[ ] read-only root filesystem
[*] mount debugfs on /debug during startup
[*] create a ramdisk or initramfs image that can be used by u-boot
() rootfs target directory
[*] keep temporary staging directory
[ ] Convert hard links to symbolic links
[*] remove man pages etc from the target image
[*] remove the /boot directory
[*] remove the /usr/src/ directory
[*] remove the /usr/include directory
[*] remove the /usr/share/locale directory
() remove these directories
v(+)

<Select>    < Exit >    < Help >
  
```

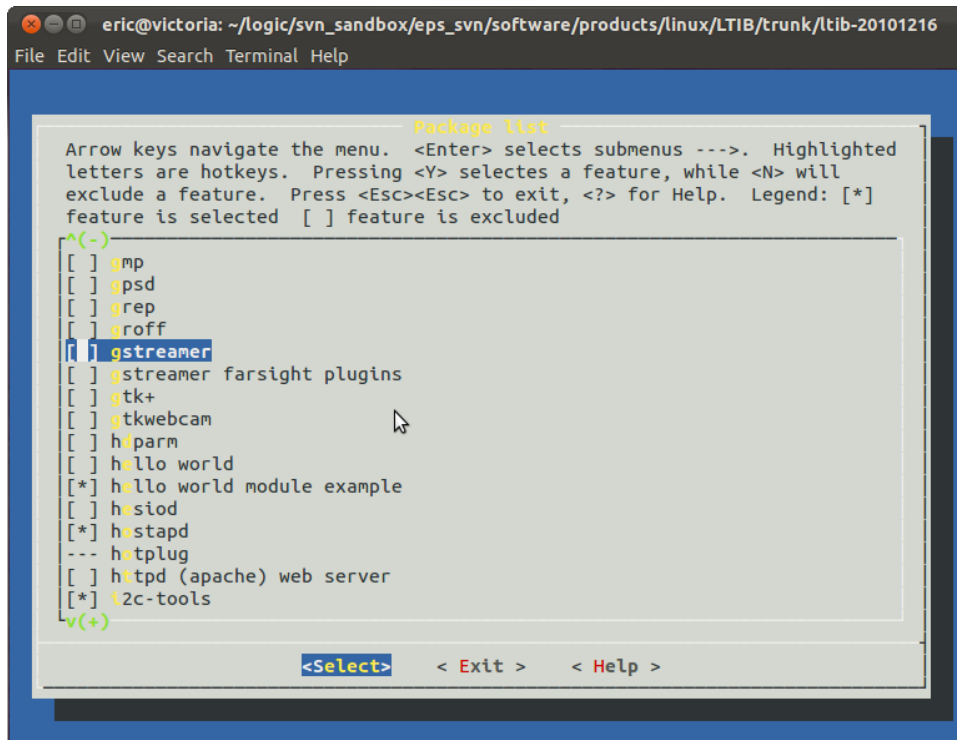
- d. Select yaffs2 and exit back to the main menu.



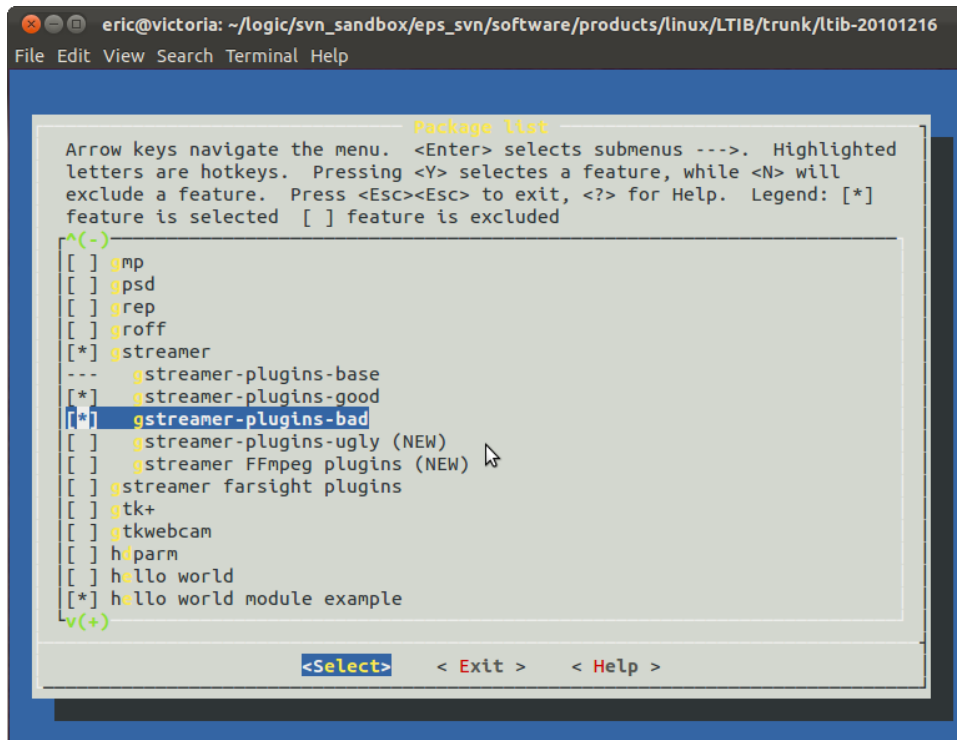
- e. In the main menu, select Packages list.



- f. In the *Package list* window, select gstreamer.



- g. Under the *gstreamer* heading, choose the following packages:
- gstreamer-plugins-base
 - gstreamer-plugins-good
 - gstreamer-plugins-bad



- h. Under the *zlib* heading, choose the following packages:
- mediactl
 - yavta

```

eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

Package list
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selects a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded

^(-)
[ ] wget
[ ] which
[*] wireless-tools
[*] iw
[*] wpa_supplicant
[ ] xfsprogs
X11 --->
[ ] yaffs_utils
[ ] zaptel
--- zlib
[*] mediactl
[*] yavta
[ ] v4l_utils
[ ] lpd_v4l2apps
--- Additional Package selection
[ ] lib-soup

<Select>  < Exit >  < Help >

```

- i. Next, select TI DVSDK Packages.

```

eric@victoria: ~/logic/svn_sandbox/eps_svn/software/products/linux/LTIB/trunk/ltib-20101216
File Edit View Search Terminal Help

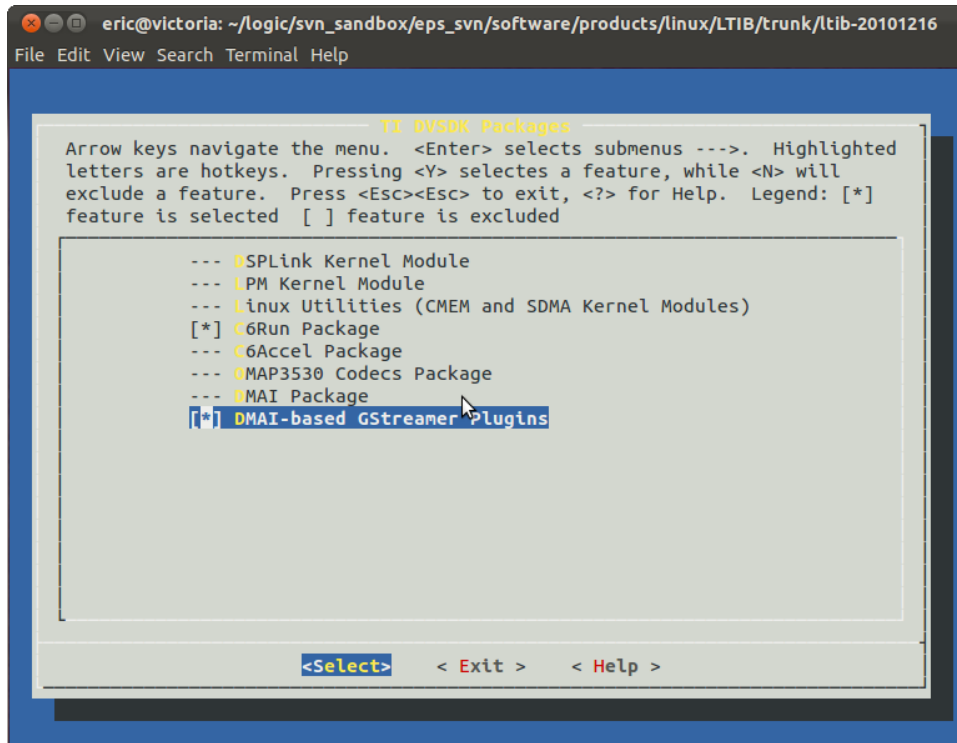
Package list
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selects a feature, while <N> will
exclude a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*]
feature is selected [ ] feature is excluded

^(-)
[*] draw-test
[*] pwr-button
[*] input-device
[*] SPI test program
[*] NEON test program
[ ] Logic PD Demo
[*] FPU speedtest
[ ] GPS demo
TI DVSDK Packages --->
--- Common package selection list
[ ] asterisk
[ ] atk
[ ] autoconf
[ ] automake
--- alsa-lib
[*] alsa-utils
v(+)

<Select>  < Exit >  < Help >

```

- j. In the *TI DVSDK Packages* window, select the following packages:
- C6Run Package
 - C6Accel Package
 - DMAI-based GStreamer Plugins



- k. Exit the main menu and save your configuration.

At this time, LTIB will rebuild the BSP with the DVSDK packages. Allow the build to complete and note any errors.

NOTE: The YAFFS image type is required in this case because the DVSDK package consumes a lot of RAM. In most cases, this prevents the use of the default RAMdisk root filesystem.

NOTE: If a package changes from "[*]" or "[]" to "---", this means the package is automatically selected due to other dependencies. This also means you cannot deselect this package unless you first deselect the dependent packages.

NOTE: Do not set C6X_C_DIR as described in the TI C6000 tool suite instructions. \This will prevent you from successfully building the DVSDK in the LTIB environment.

NOTE: One of the GStreamer plugins may causes some error messages to be emitted during GStreamer init. It has not been determined yet which plugin causes these errors. However, since this error only occurs while the plugin scanner is running at startup, it shouldn't cause any issues. Below is an example of the error seen.

```
(gst-plugin-scanner:773): GStreamer-CRITICAL **: gst_caps_ref: assertion
`GST_CAPS_REFCOUNT_VALUE (caps) > 0' failed
```


8.2 Run Time Configuration

1. Once you have the BSP built with the DVSDK packages, make a bootable SD card by copying the necessary files from the build (see Section 3.2.10.5 or Section 3.2.13.2).
2. The U-Boot `$otherbootargs` environment variable must then be updated to allocate memory for DSPLink, CMEM, and related utilities.

```
OMAP Logic # setenv otherbootargs 'ignore_loglevel early_printk
no_console_suspend mem=55M@0x80000000 mem=128M@0x88000000'
OMAP Logic # saveenv
```

Review Section 3.2 for information regarding the U-Boot bootloader; review Section 3.2.2 for details regarding the `$otherbootargs` environment variable.

3. You can now cycle the power on the DM3730 Development Kit and boot into Linux. Please note that on early DM37x Linux BSP releases, the modules `cmemk`, `dsplinkk`, `lpm_omap3530`, and `sdmak` failed to load at boot. This is indicated by the boot log just prior to the login prompt, as seen in the output below.

```
DirectFB: Setup DirectFB for touch input Restoring ALSA state for
soundcard omap3logic
FATAL: Module cmemk not found.
FATAL: Module dsplinkk not found.
FATAL: Module lpm_omap3530 not found.
FATAL: Module sdmak not found.
Enabling PM off mode:

Welcome to the LTIB Embedded Linux Environment

!!!!!! WARNING !!!!!!!

The default password for the root account is: root
please change this password using the 'passwd' command
and then edit this message (/etc/issue) to remove this message.

To enable DHCP on ethernet, type "ifup eth0"

DM-37x login:
```

4. If you see the modules fail to load, as shown in the example above, enter the command below in your LTIB directory to force LTIB to update the module dependencies.

```
bash$ ./ltib -p modeps

Processing platform: Logic OMAP3530/03&DM3730/03 reference boards
=====
using config/platform/omap_logic/.config

Processing: modeps
=====
```

```
Started: Fri Nov 16 16:34:12 2012
Ended:   Fri Nov 16 16:34:14 2012
Elapsed: 2 seconds
```

```
Build Succeeded
```

```
bash$
```

5. When complete, recopy the images onto the kit and reboot.

8.3 Example DSP and Camera Use

Below are some examples of commands that demonstrate the DSP and camera functionality. Some commands make use of a camera connected to the development kit via the parallel camera interface or via the USB port. Be aware that not all baseboards include a parallel camera connector. Please verify a parallel camera can be connected directly to your baseboard before proceeding with the parallel camera examples.

8.3.1 DSP Example

- DSPLink Example

```
DM-37x# cd /usr/share/ti/ti-dsplink-examples
DM-37x# ./loopgppp ./loop.out 1000 5000 0
```

- C6Accel Example

```
DM-37x# cd /usr/share/ti/c6accel-apps
DM-37x# ./c6accel_app
```

NOTE: You may see occasional errors when running the C6Accel example due to a bug in CGT tools v6.1.14. This is not a problem with the Logic PD DM37x Linux BSP. See issue DM37LINUX-670 in the [DM37x Linux BSP Release Notes](#)³⁵ for additional information.

- DMAI Example

```
DM-37x# cd /usr/share/ti/ti-dmai-apps/
DM-37x# ./image encode io1 dm3730.x470MV -c jpegenc -i /dev/zero -o
jpeg_test_encoded.jpeg -r 720x576 --iColorSpace 3 --benchmark
DM-37x# ./image encode io1 dm3730.x470MV -c jpegenc -i /dev/urandom -o
jpeg_test_encoded.jpeg -r 720x576 --iColorSpace 3 --benchmark
DM-37x# ./image decode io1 dm3730.x470MV -c jpegdec -i
jpeg_test_encoded.jpeg -o jpeg_test_decoded.yuv --oColorSpace 3 --
benchmark
DM-37x# ./video encode io1 dm3730.x470MV -c h264enc -i /dev/zero -n 100
-o h264_test_encoded.h264 -r 720x576 --benchmark
DM-37x# ./video encode io1 dm3730.x470MV -c h264enc -i /dev/urandom -n
10 -o h264_test_encoded.h264 -r 720x576 --benchmark
```

³⁵ <http://support.logicpd.com/DesktopModules/Bring2mind/DMX/Download.aspx?portalid=0&EntryId=1396>

```
DM-37x# ./video_decode_io2_dm3730.x470MV -c h264dec -i
h264_test_encoded.h264 -n 10 -o h264_test_decoded.yuv --benchmark
```

- C6Run Example

NOTE: C6Run uses a different CMEM configuration than the other DSPLink libraries. Make sure to restart CMEM with the original arguments before using other DSPLink libraries.

1. First, perform some initial setup.

```
DM-37x# cd /usr/share/ti/c6run-apps/
DM-37x# ./unloadmodules.sh
DM-37x# ./loadmodules.sh
```

2. Run a "Hello World" example.

```
DM-37x# examples/c6runapp/hello_world/hello_world_arm
DM-37x# examples/c6runapp/hello_world/hello_world_dsp
```

3. Run a CIO example.

```
DM-37x# examples/c6runapp/cio_example/cio_example_arm
DM-37x# examples/c6runapp/cio_example/cio_example_dsp
```

4. Run a benchmark example.

```
DM-37x# examples/c6runapp/emqbit/bench_arm
DM-37x# examples/c6runapp/emqbit/bench_dsp
```

5. Finally, restart CMEM with the original arguments.

```
DM-37x# /etc/rc.d/init.d/cmem restart
```

8.3.2 Parallel Camera Example

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. More information about the camera adapter board is available on the [Leopard Imaging website](http://shop.leopardimaging.com/product.sc?productId=24).³⁶

1. Prepare the system to use the LI-5M04 camera by setting the exposure and gain on the camera and setting up the video input hardware pipeline.

³⁶ <http://shop.leopardimaging.com/product.sc?productId=24>

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 DSP or camera use.

```
DM-37x# echo -ne "\033[9;0]\033[?251" > /dev/tty0
DM-37x# yavta -w '0x00980911 720' /dev/v4l-subdev8
DM-37x# yavta -w '0x00980913 16' /dev/v4l-subdev8
DM-37x# yavta -w '0x0098090e 125' /dev/v4l-subdev8
DM-37x# yavta -w '0x0098090f 175' /dev/v4l-subdev8
DM-37x# media-ctl -v -r -l '"mt9p031":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]'
```

2. Preview the LI-5M04 camera on the LCD.

```
DM-37x# media-ctl -v -f '"mt9p031":0 [SRGBG8 1298x970
(664,541)/1298x970], "OMAP3 ISP CCDC":2 [SRGBG10 1298x970], "OMAP3 ISP
preview":1 [UYVY 1298x970], "OMAP3 ISP resizer":1 [UYVY 640x480]'
DM-37x# gst-launch v4l2src device=/dev/video6 num-buffers=150 always-
copy=false queue-size=4 ! 'video/x-raw-
yuv,format=(fourcc)UYVY,width=640,height=480,framerate=30/1' ! queue !
tidisplaysink2 mmap-buffer=true
```

3. Capture a JPEG image from the LI-5M04 camera and display it.

```
DM-37x# media-ctl -v -f '"mt9p031":0 [SRGBG8 2610x1954
(7,49)/2610x1954], "OMAP3 ISP CCDC":2 [SRGBG10 2610x1954], "OMAP3 ISP
preview":1 [UYVY 2610x1954], "OMAP3 ISP resizer":1 [UYVY 2592x1944]'
DM-37x# gst-launch v4l2src device=/dev/video6 num-buffers=1 ! 'video/x-
raw-yuv,format=(fourcc)UYVY,width=2592,height=1944' ! TIImgenc1
engineName=codecServer codecName=jpegenc resolution=2592x1944
iColorSpace=UYVY oColorSpace=YUV420P qValue=97 ! filesink
location=still.jpg
DM-37x# gst-launch filesrc location=still.jpg ! jpegdec !
ffmpegcolspace ! videoscale ! 'video/x-raw-rgb,width=320,height=240'
! fbdevsink device=/dev/fb0
```

4. Capture an H.264 video from the LI-5M04 camera and play it back.

```
DM-37x# media-ctl -v -f '"mt9p031":0 [SRGBG8 1298x970
(664,541)/1298x970], "OMAP3 ISP CCDC":2 [SRGBG10 1298x970], "OMAP3 ISP
preview":1 [UYVY 1298x970], "OMAP3 ISP resizer":1 [UYVY 640x480]'
DM-37x# gst-launch v4l2src device=/dev/video6 num-buffers=300 always-
copy=false queue-size=4 ! 'video/x-raw-
yuv,format=(fourcc)UYVY,width=640,height=480,framerate=24/1' !
TIPrepEncBuf numOutputBufs=4 contiguousInputFrame=false ! tee name=tee
tee. ! queue ! tidisplaysink2 mmap-buffer=true tee. ! queue ! TIVidenc1
engineName=codecServer codecName=h264enc contiguousInputFrame=true !
queue ! avimux ! filesink blocksize=65536 location=video640x480.avi
```

```
DM-37x# gst-launch filesrc location=video640x480.avi ! avidemux !
TIIViddec2 engineName=codecServer codecName=h264dec ! queue !
tidisplaysink2
```

8.3.3 USB Webcam Example

The targeted USB webcam is the Logitech C210 webcam. Other webcams that support USB Video Class (UVC) standards may work as well, but will not necessarily be supported by Logic PD. More information can be found using the following resources:

- [Logitech website](http://www.logitech.com/en-us/webcam-communications/webcams/7022)³⁷
- [Ideas on Board website](http://www.ideasonboard.org/uvic/#devices)³⁸

The USB webcam should be connected after booting Linux to ensure that the `/dev/video*` nodes have consistent numbering. If you are using the DM3730 SOM-LV Development Kit, the webcam can be connected to a USB Host port (USB2/J7, USB4/J43, USB5/J44) or the USB OTG port (J6). If you are using the DM3730 Torpedo Development Kit, the webcam must be connected to the USB OTG port (J19) since the ISP1763 driver does not currently support isochronous transfers.

To connect a webcam to the USB OTG port, first connect the webcam to a self-powered USB hub. Then, connect the USB hub to the kit using an adapter that has a [USB mini-A male plug](http://www.logitech.com/en-us/webcam-communications/webcams/7022)³⁹ on one side and a USB-A female receptacle on the other. Note that some USB mini-A male adapters do not seem to ground the ID pin properly, making them unusable.

1. Prepare the system for using the C210 USB webcam by disabling the virtual console screen timeout.

```
DM-37x# echo -ne "\033[9;0]\033[?251" > /dev/tty0
```

2. Preview the C210 USB webcam on the LCD.

```
DM-37x# gst-launch v4l2src num-buffers=90 device=/dev/video9 !
'image/jpeg,width=320,height=240,framerate=30/1' ! jpegdec !
ffmpegcolospace ! tidisplaysink2
```

3. Capture a JPEG image from the C210 USB webcam and display it. The `jpegdec/jpegenc` is needed to add a standard JPEG header to the minimal MJPEG frame returned from the device.

```
DM-37x# gst-launch v4l2src num-buffers=1 device=/dev/video9 !
'image/jpeg,width=640,height=480,framerate=15/1' ! jpegdec ! jpegenc !
filesink location=usbstill.jpg
DM-37x# gst-launch filesrc location=usbstill.jpg ! jpegdec !
ffmpegcolospace ! videoscale ! 'video/x-raw-rgb,width=320,height=240'
! fbdevsink device=/dev/fb0
```

4. Capture an MJPEG video from the C210 USB webcam and play it back.

³⁷ <http://www.logitech.com/en-us/webcam-communications/webcams/7022>

³⁸ <http://www.ideasonboard.org/uvic/#devices>

³⁹ <http://www.digikey.com/product-search/en?x=14&y=18&lang=en&site=us&keywords=10-00003-ND>

```
DM-37x# gst-launch v4l2src num-buffers=90 device=/dev/video9 !  
'image/jpeg,width=640,height=480,framerate=15/1' ! queue ! avimux !  
filesink blocksize=65536 location=usbvideo.avi  
DM-37x# gst-launch filesrc location=usbvideo.avi ! avidemux ! jpegdec !  
ffmpegcolorspace ! tidisplaysink2
```

9 GTK Demo

GTK requires some software overhead to implement, so it is not enabled by default. This section explains how to enable GTK in your build and run the GTK demo application.

9.1 Prepare Build Tools

To build GTK, be sure to include the necessary packages on your build machine. Please refer to the *README-setup* file in the DM37x Linux BSP tar ball for an up-to-date list of the packages needed to build GTK.

Or, if you choose to use the package installer script *ltib_setup.sh* (see Section 2.3.1) that is in the tar ball, be sure to answer “YES” if prompted to install packages for GTK. If you are not prompted, the packages for GTK have been included by default. See Section 2.3.1.2 for more information on installing packages on your build machine.

9.2 Build

1. If you have not done so already, be sure to perform your first build of the BSP as described in Section 2.4.
2. Next, update your build to include the following GTK+ and Liberation fonts packages. See Section 2.5.4 for more information.
 - a. Begin by running the command below.

```
bash$ ./ltib -c
```

- b. In the main menu, select Packages list.
 - c. In the *Packages list* window, select the appropriate packages.
 - d. Exit the menu.
3. LTIB will build the BSP, including those packages.

9.3 Run Demo

1. Once the build completes, place the following files from the build into the root directory of an SD card. See Section 1.8 and/or Section 2.4 for additional information regarding these files.
 - ☐ *MLO*
 - ☐ *u-boot.bin*
 - ☐ *uImage*
 - ☐ *rootfs.ext2.gz.uboot*
2. Because the size of the RAM-based root filesystem is much larger with the GTK tools, you may need to allocate more RAM to the filesystem. To do this, follow the steps below.
 - a. Interrupt the boot sequence to get to the U-Boot shell.
 - b. Set the *ramdisksize* environment variable as shown below.

```
OMAP Logic # setenv ramdisksize 128000
```

- c. Save the environment for future boots.

```
OMAP Logic # saveenv
```

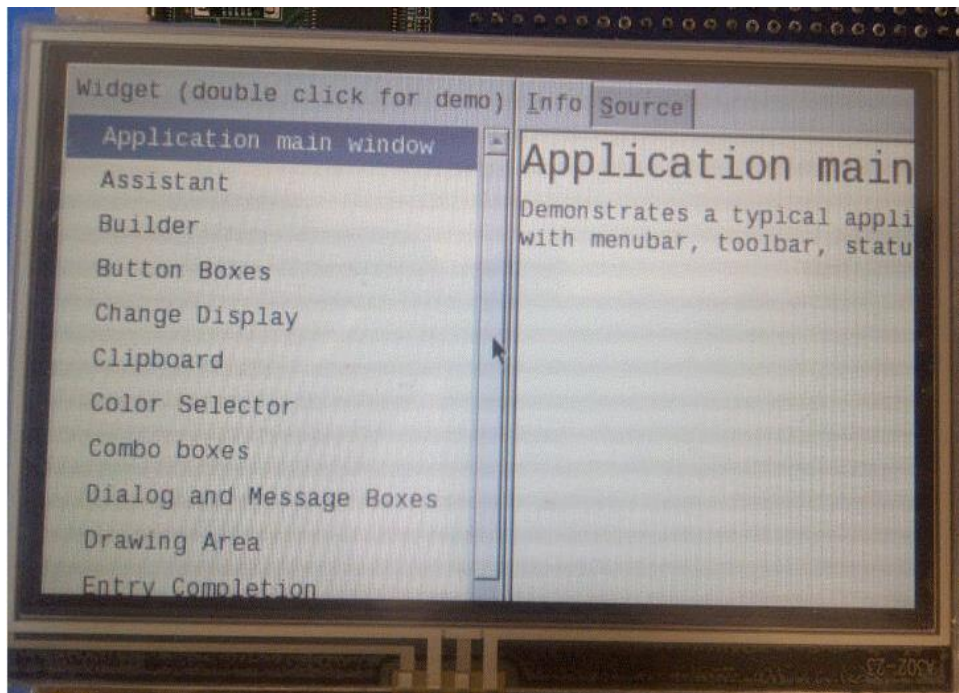
- d. Continue the boot cycle for the current session.

```
OMAP Logic # boot
```

3. When Linux is up and running, proceed to log in using root credentials.
4. Once logged in, launch the demo.

```
DM-37x# /usr/bin/gdk-pixbuf-query-loaders > /usr/etc/gtk-2.0/gdk-  
pixbuf.loaders  
DM-37x# mkdir /usr/etc/pango  
DM-37x# pango-querymodules > /usr/etc/pango/pango.modules  
DM-37x# /usr/bin/gtk-demo
```

You should see the following output on the LCD panel.



10 Customize LTIB

LTIB has a pool of compilers, bootloader source code, OS source code, application source code, and various utilities. When you have created something you want to include in the LTIB pool, this section describes how you can integrate it.

LTIB is a powerful tool capable of much more than what is documented in this user guide. If this document does not provide enough information on how to use LTIB, please take a look at the */doc/LtibFaq* file in the DM37x Linux BSP tar ball.

All file and path references below assume the reader is working in the *LTIB* directory (i.e., the directory in which the DM37x Linux BSP tar ball was unpacked).

The information below is provided as guidance, without any specific examples. Each package has its own requirements and may require more or less steps than those outlined below. The best way to create your own package is to use another package as a template (such as the *helloworld* or *hello_mod* packages) and create a similar, simple package. When the package build is successful, begin adding and updating the package as needed.

10.1 Definitions

Several words and phrases that will be used in the following sections have been defined below for clarity.

- **Package:** A group of source code files that, when compiled, make up an application to be included in the filesystem.
- **Service:** An application that is started when the system boots and stopped when the system is shut down.
- **Prep a package:** Extract a tar ball into *rpm/BUILD* and apply all patches.
- **Build a package:** Compile a package into files for the target (i.e., create executables).
- **Install a package:** Copy files of interest to the target's filesystem.
- **Deploy a package:** Installs, strips, and compresses the complete target filesystem, then generates a filesystem image.
- **Patch/merge:** If you make changes to a package (e.g., integrate tweaks, bug fixes), LTIB can capture your modifications in a patch. By creating a patch, the original package source remains unchanged, and the changes you have made to the package are kept as a separate patch file.

10.2 Integrate New Package

Start with your existing lump of code. You'll eventually need to make changes to it, but LTIB can capture and reproduce those changes for you later by using the *patchmerge* feature.

The following instructions assume you are working from the *LTIB* root directory where you installed the DM37x Linux BSP. The procedure below explains how to integrate a package and name it *myStuff*.

1. Compress your source into a tar ball. For example, use *tar -cjf myStuff-version.tar.bz2 myStuff-version/*. Note that you'll need to adhere to the following naming restrictions:
 - ❑ The directory name and the tar ball must have the same base name.
 - ❑ The version must be separated from the rest of the name by a hyphen.

- There must not be a period anywhere in the base.

One available concession from all these restrictions is that you can use underscores wherever you want.

2. Move the compressed tar ball to a package pool directory, such as *lpd-IP-package-pool/*.
3. Publish a checksum of the tar ball. For example, use *md5sum myStuff-version.tar.bz2 > myStuff-version.tar.bz2.md5*.
4. Create a directory for the package's spec file in the *dist/lfs-5.1* folder. For example, use *mkdir dist/lfs-5.1/myStuff*.
5. Create a spec file that ties the package together. For example, use *dist/lfs-5.1/myStuff/myStuff.spec*. It identifies the tar ball (and patch files, if any) and is where you'll write shell scripts to build and install. Review some other spec files to see how to construct the contents. The *dist/lfs-5.1/template* file provides a good example and is intended to be used for such purposes.
6. Try it out by using *./ltib -p myStuff.spec -m prep*. If all has gone well, you'll end up with *rpm/BUILD/myStuff-version*. If not, you'll have to figure out where you led LTIB astray. Fortunately LTIB is pretty good about printing out the shell commands it is executing.
7. Next, you need to update the *packages.lkc* and *pkg_map* files.
 - a. Navigate to *config/userspace*.
 - b. Open and edit the *packages.lkc* file. Create an entry for your new package. Keep in mind that the order of the entries in this file is the same order of the entries in the LTIB package selection window. If you need an example to follow, see the *PKG_HELLOWORLD* entry.
 - c. Open and edit the *pkg_map* file. The order of entries is the order that LTIB will build them. So, if your package depends on other packages being built, make sure your entry is positioned after all of your package's dependents. Follow the pattern seen in the file.
 - d. The left side of the equal sign will be the package selector that matches the "*config <name>*" of your new entry in *packages.lkc*, where *<name>* starts with "PKG_" as in *PKG_MY_NEW_APPLICATION*. The value on the right side of the equal sign will be the name of the *.spec* directory in *dist/lfs-5.1* and the prefix of the *.spec* file in that directory.
 - e. Note that LTIB understands versioning of spec files. If you have *helloworld-1.0.spec* and *helloworld-2.0.spec* in *dist/lfs-5.1/helloworld*, LTIB will pick *helloworld-2.0.spec* when you specify *-p helloworld* on the command line. If you need a particular version, you can use the full spec name on the command line (e.g., *-p helloworld-1.0.spec*).

10.2.1 Integrate New Service

This section will describe how to add a service that starts when the system boots and stops when the system is shut down. For this example, we'll use a service called *lpd-demo* that is a demonstration program that will run on startup and will continue in the background.

1. Integrate your package. See Section 10.2 for additional information.
2. Add a startup option to the platform's *sysconfig-xxxx.lkc* file, where *xxxx* is the platform of interest. In the case of the DM37x Linux BSP v2.2-2, this is

config/packages/omap_logic/sysconfig-omap.lkc. To determine which *sysconfig-xxxx.lkc* is sourced, review *main.lkc* and look for *source sysconfig-xxxx.lkc*, where *xxxx* in this case is indicated in the file.

Note that the startup selector should be dependent on the package. That way, LTIB won't ask to start a service if the service isn't built and installed. Consider the example below from the *sysconfig-omap.lkc* file, where we are creating an *lpc-demo* service.

```
config SYSCFG START LPC DEMO
    depends PKG LPD DEMO
    bool "start lpd-demo at boot"
    default
```

3. Modify *sysconfig.spec* for the platform. In this case, it will be *dist/lfs-5.1/sysconfig/sysconfig-omap_logic.spec*, determined by the *pkg_map* file mapping most likely mentioned in the *platform pkg_map* file. You need to update *all_services*, *all_services_r*, *cfg_services*, and *cfg_services_r*, and make your entry selected by whether the SYSCFG_START_LPD_DEMO is enabled.

Note that the "_r" versions of the services are reversed in that they stop the services in the reverse order that they were started. You'll need something similar to the example below. Assuming services for this example are A, B, C:

```
if [ "$SYSCFG START LPD DEMO" = "Y" ]
then
    lpd demo=lpd-demo
fi

all services="A B C lpd demo"
all services r="lpd demo C B A"
cfg services="$A $B $C $lpd demo"
cfg services r="$lpd demo $C $B $A"
```

4. Modify the *skell* package to add the service startup script. In our example, the */etc/rc.d/init.d/lpd-demo* is modified to be:

```
#!/bin/sh
# Enable lpd-demo on startup
#
action=$1
if [ "$1" = "stop" -o "$action" = "restart" -o "$action" = "init" ]
then
    echo "Stopping lpd-demo: "
    exec /etc/rc.d/rc.restart lpd-demo $action
fi

if [ "$action" = "reinit" ]
then
    action=start
fi

if [ "$action" = "start" -o "$action" = "restart" ]
then
```

```
lpd-demo&
fi
```

5. Execute `./ltib -c`.
6. In the configuration under the *Target System Configuration* menu, enable the *lpd-demo* package and the *lpd-demo* service. Now when you boot the resulting image, it should start the *lpd-demo* service/application.

10.2.2 Build

Now that you have prepared your package, it is time to work on building it. LTIB gives you some shell variables to help out; *TOP* and *TOOLCHAIN_** are some of the most useful. To see what shell variables your package would have available at build time, use `./ltib -m shell` in the LTIB directory, then enter `set | more`. Enter `exit` when you're done with the shell.

When you use `ltib -p myStuff.spec -m scbuild`, LTIB will find the "%Build" section of the *myStuff.spec* file and run the lines it finds there as a shell script. The initial directory will be *rpm/BUILD/myStuff-version* and the kernel will be at *\$TOP/rpm/BUILD/linux*. You can now do *make* or *configure* or *cd* or whatever -- it is your shell script to write. LTIB will stop if any line results in an error (\$?).

The *scbuild* command assumes you have done prep once. You can use *scbuild* over and over as needed.

10.2.3 Installing

Now you have some items that you want available on the target. Start writing another shell script in the "%Install" section of your spec file. The goal is to copy the files of interest to *\$RPM_BUILD_ROOT/%{pfx}*; LTIB will do the rest

You'll want to start with `rm -rf $RPM_BUILD_ROOT` to keep LTIB happy and prevent accumulation from previous *scinstall* invocations. You can then create directories (perhaps `mkdir -p` or `install -d`) and copy files.

You can even make symbolic links (*ln -s*). Be sure to stay within the target filesystem or use absolute paths that are resolvable on the target.

After a successful *scbuild* of your *myStuff* package, use `./ltib -p myStuff.spec -m scinstall`.

If you're really brave or you think it's going to work, use `./ltib -p myStuff.spec -m scdeploy`. This always does a *scinstall*, but it then goes the next step and makes a target elf that contains the explicitly configured items plus your package.

You can get a sneak peek of the target filesystem at the *rootfs* sub-directory.

10.2.4 Test

You'll need to copy *rootfs/boot/MLO*, *rootfs/boot/u-boot.bin*, *rootfs/boot/uImage*, and *rootfs.ext2.gz.uboot* to an SD card. Your files will show up at the location you specified with the *scinstall* command. **NOTE:** The executable files will have their symbols stripped.

10.2.5 Configure

To make your package more generally available, you need to tell LTIB that it is available and give LTIB the platforms for which it is available. To do this, you will edit various *.lkc* files under

config/ (see "How do I introduce my new package to the config system" in *doc/LtibFaq* for more information). You can also add an automatic startup script to be run at the end of the target's boot (see "How to add a daemon to init" in *doc/LtibFaq* for more information).

Once your package is configured (both defined and selected), you no longer have to type the `.spec` suffix on your package name.

Actually, the inverse is more important to remember for package debugging: If the `-p` argument to LTIB doesn't recognize your package name, *config* either doesn't understand it (not defined) or doesn't want it (not selected). However, you can always force it by suffixing `.spec` to your package name; LTIB will find the file in a sub-directory of *dist/lfs-5.1*.

10.3 Removing Drivers

With a plethora of drivers offered by Logic PD, not all of them are required on every device. This section explains how to remove drivers while building the kernel. Generally, this allows less use of memory and power.

10.3.1 General Instructions

Below is the general set of instructions to remove any driver that is provided with the kernel.

1. First, move to the appropriate directory.

```
bash$ cd ~/logic/Logic_BSPs/Linux_3.0/1027480_LogicPD_Linux_BSP_2.4-4
```

2. Build the Linux kernel using the preset configuration in Section 2.4.

```
bash$ ./ltib -b --preconfig config/platform/omap_logic/defconfig
```

3. Once the preset configuration is built, open the kernel configuration menu.

```
bash$ ./ltib -c
```

4. Select "Configure the kernel."

```

LTIB: Logic OMAP3530/03&DM3730/03 reference boards
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> selects a feature, while <N> will exclude
a feature. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] feature is
selected [ ] feature is excluded
^(-)
[ ] Produce cscope index
(omap3logic_defconfig) kernel preconfig
--- Include kernel headers
[*] use 'make headers install' to install kernel headers
[*] Configure the kernel
--- Leave the sources after building
--- Package selection
    Package list --->
--- Target System Configuration
    Options --->
--- Target Image Generation
    Options --->
(BSP-dm37x-2.4-2) BSP Release Level (supplied by platform defconfig)
---
Load an Alternate Configuration File
Save Configuration to an Alternate File

<Select>  < Exit >  < Help >

```

5. Select Exit and select Yes when asked if you wish to save your new configuration.

```

Do you wish to save your new configuration?

< Yes >  < No >

```

6. Next, select Device Drivers

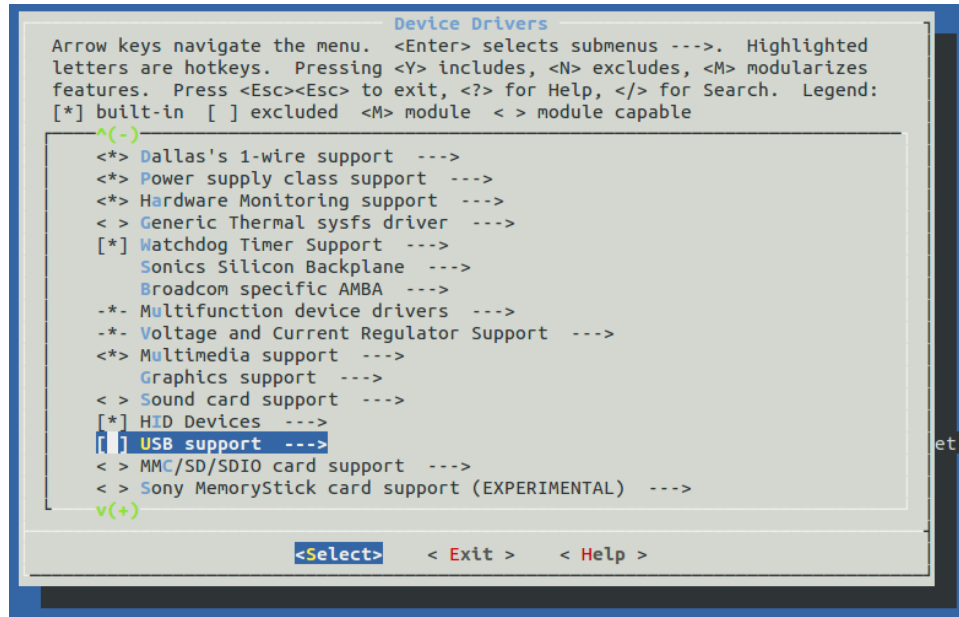
```

Linux/arm 3.0.0-BSP-dm37x-2.4-2 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend:
[*] built-in [ ] excluded <M> module < > module capable
^(-)
Kernel Features --->
Boot options --->
CPU Power Management --->
Floating point emulation --->
Userspace binary formats --->
Power management options --->
[*] Networking support --->
    [*] Device Drivers --->
        File systems --->
        Kernel hacking --->
        Security options --->
        -- Cryptographic API --->
        Library routines --->
---
Load an Alternate Configuration File
Save an Alternate Configuration File

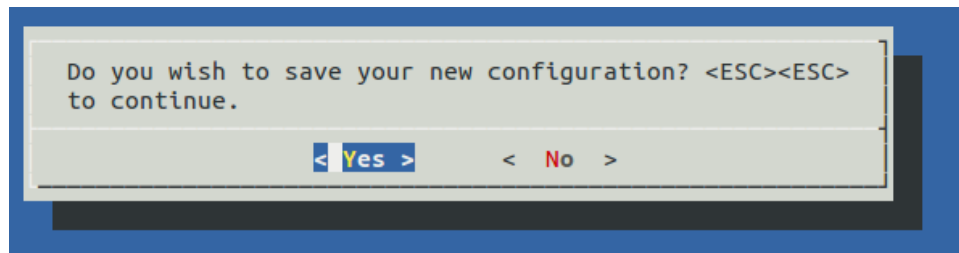
<Select>  < Exit >  < Help >

```

7. Once in this directory, select the drivers that are to be deleted. In this example, the USB driver will be removed.



8. Press the space bar to remove the '*''. This will remove all the USB support.
9. Select Exit and select Yes when asked if you wish to save your new configuration.



The new configurations will be applied and built. The *uImage* file to boot the SOM can be found at *REL-Itib-DM3730-2.4-2/rootfs/boot/uImage*.

10.3.2 Remove Specific Interfaces

This section provides guidelines for removing specific drivers. Once you get to the kernel configuration menu, search for the configuration definitions in the table below using the search "/" option in LTIB. This will give you the direct path to the driver for disabling it in the LTIB kernel configuration menu.

The table below provides the configuration setting for specific drivers.

Table 10.1: Driver-Specific Configuration Settings

Interface	Configuration Variable
1-Wire	CONFIG_W1_SLAVE_BQ27000
	CONFIG_POWER_SUPPLY
	CONFIG_BATTERY_BQ27X00
	CONFIG_BATTERY_BQ27X00_PLATFORM
	CONFIG_HWMON
	CONFIG_HDQ_MASTER_OMAP

Interface	Configuration Variable
Bluetooth	CONFIG_BT_WILINK CONFIG_WIRELESS CONFIG_WEXT_CORE CONFIG_WEXT_PROC CONFIG_BT CONFIG_BT_L2CAP CONFIG_BT_SCO CONFIG_BT_RFCOMM CONFIG_BT_RFCOMM_TTY CONFIG_BT_BNEP CONFIG_BT_HIDP
Display Hardware Drivers	CONFIG_OMAP2_VRAM CONFIG_OMAP2_VRFB CONFIG_OMAP2_DSS CONFIG_OMAP2_VRAM_SIZE CONFIG_OMAP2_DSS_DEBUG_SUPPORT CONFIG_OMAP2_DSS_DPI CONFIG_OMAP2_DSS_VENC CONFIG_OMAP2_DSS_DSI CONFIG_OMAP2_DSS_MIN_FCK_PER_PCK CONFIG_OMAP2_DSS_SLEEP_BEFORE_RESET CONFIG_OMAP2_DSS_SLEEP_AFTER_VENC_RESET CONFIG_FB_OMAP2 CONFIG_FB_OMAP2_DEBUG_SUPPORT CONFIG_FB_OMAP2_NUM_FBS CONFIG_PANEL_GENERIC_DPI CONFIG_PANEL_OMAP3LOGIC CONFIG_BACKLIGHT_LCD_SUPPORT CONFIG_LCD_CLASS_DEVICE CONFIG_LCD_PLATFORM CONFIG_BACKLIGHT_CLASS_DEVICE CONFIG_BACKLIGHT_GENERIC CONFIG_DISPLAY_SUPPORT CONFIG_DUMMY_CONSOLE CONFIG_FRAMEBUFFER_CONSOLE CONFIG_FONTS CONFIG_FONT_8x8 CONFIG_FONT_8x16 CONFIG_LOGO CONFIG_LOGO_LINUX_MONO CONFIG_LOGO_LINUX_VGA16 CONFIG_LOGO_LINUX_CLUT224 CONFIG_FB_CFB_FILLRECT CONFIG_FB_CFB_COPYAREA CONFIG_FB_CFB_IMAGEBLIT CONFIG_FB
Ethernet	CONFIG_SMSC_PHY CONFIG_NET_ETHERNET CONFIG_SMSC911X CONFIG_NETDEV_1000 CONFIG_NETDEV_10000 CONFIG_WLAN
LED Driver	CONFIG_LEDS_GPIO CONFIG_LEDS_GPIO_PLATFORM CONFIG_LEDS_TRIGGERS
MMC/SD/SDIO Card Driver	CONFIG_MMC_BLOCK CONFIG_MMC_BLOCK_MINORS CONFIG_MMC_BLOCK_BOUNCE CONFIG_SDIO_UART
MMC/SD/SDIO Host Controller Drivers	CONFIG_MMC_OMAP CONFIG_MMC_OMAP_HS CONFIG_NEW_LEDS CONFIG_LEDS_CLASS

Interface	Configuration Variable
Serial UART	CONFIG_OMAP3LOGIC_UART_A CONFIG_OMAP3LOGIC_UART_B CONFIG_OMAP3LOGIC_UART_C
SPI Interface	CONFIG_OMAP3LOGIC_AT25160AN CONFIG_OMAP3LOGIC_SPI1_CS0 CONFIG_OMAP3LOGIC_SPI1_CS1 CONFIG_OMAP3LOGIC_SPI1_CS2 CONFIG_OMAP3LOGIC_SPI1_CS3 CONFIG_OMAP3LOGIC_SPI3_CS0 CONFIG_OMAP3LOGIC_SPI3_CS1
Touch	CONFIG_TOUCHSCREEN_TSC2004 CONFIG_INPUT_TOUCHSCREEN
USB Host Controller Drivers	CONFIG_USB_ISP1763 CONFIG_USB_ISP1763_HCD CONFIG_USB_ISP1763_HCD_SELECT
USB OTG Driver	CONFIG_TWL4030_USB
Wireless Chipset	CONFIG_WL12XX_MENU CONFIG_WL12XX CONFIG_WL128X_FIRMWARE_SOURCE CONFIG_WL127X_FIRMWARE_SOURCE CONFIG_WL12XX_HT CONFIG_WL12XX_SDIO CONFIG_WL12XX_PLATFORM_DATA

11 Basic Driver/Kernel Debugging Information

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

11.1 *dmesg*

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.

```
DM-37x# 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 otherbootargs parameter is set to ignore_loglevel, which means you cannot turn off the console log messages. Remove that from otherbootargs in the U-Boot environment and use the command below to set the log level that will be output to the console. All messages are still saved in */var/log/messages*

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

More information on kernel log levels can be found in the Linux [Debugging by print wiki article](http://elinux.org/Debugging_by_print/wiki/article).⁴⁰

11.2 Dynamic Debug

To see all possible debug messages in a specific file, use *grep* for the desired file within *dynamic_debug/control*.

The example below shows all possible debug messages available in *mcbasp.c*.

```
DM-37x# cat /sys/kernel/debug/dynamic_debug/control | grep mcbasp.c
arch/arm/plat-omap/mcbasp.c:1819 [mcbasp]omap mcbasp probe - "Initializing
OMAP McBSP (%d).\012"
arch/arm/plat-omap/mcbasp.c:211 [mcbasp]omap mcbasp config - "Configuring
McBSP%d phys base: 0x%08lx\012"
arch/arm/plat-omap/mcbasp.c:185 [mcbasp]omap mcbasp rx dma callback - "RX
DMA callback : 0x%x\012"
arch/arm/plat-omap/mcbasp.c:171 [mcbasp]omap mcbasp tx dma callback - "TX
DMA callback : 0x%x\012"
```

⁴⁰ http://elinux.org/Debugging_by_printing#Log_Levels

```

arch/arm/plat-omap/mcbsp.c:152 [mcbsp]omap mcbsp rx irq handler - "RX
IRQ callback : 0x%x\012"
arch/arm/plat-omap/mcbsp.c:132 [mcbsp]omap mcbsp tx irq handler - "TX
IRQ callback : 0x%x\012"
arch/arm/plat-omap/mcbsp.c:96 [mcbsp]omap mcbsp dump reg - "*****
McBSP%d regs *****\012"
arch/arm/plat-omap/mcbsp.c:98 [mcbsp]omap mcbsp dump reg - "DDR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:100 [mcbsp]omap mcbsp dump reg - "DDR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:102 [mcbsp]omap mcbsp dump reg - "DXR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:104 [mcbsp]omap mcbsp dump reg - "DXR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:106 [mcbsp]omap mcbsp dump reg - "SPCR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:108 [mcbsp]omap mcbsp dump reg - "SPCR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:110 [mcbsp]omap mcbsp dump reg - "RCR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:112 [mcbsp]omap mcbsp dump reg - "RCR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:114 [mcbsp]omap mcbsp dump reg - "XCR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:116 [mcbsp]omap mcbsp dump reg - "XCR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:118 [mcbsp]omap mcbsp dump reg - "SRGR2:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:120 [mcbsp]omap mcbsp dump reg - "SRGR1:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:122 [mcbsp]omap mcbsp dump reg - "PCR0:
0x%04x\012"
arch/arm/plat-omap/mcbsp.c:123 [mcbsp]omap mcbsp dump reg -
"*****\012"
DM-37x#

```

11.3 Enable Specific Debug Message

You can enable only specific messages per file or within a specific file. This section provides examples about how to show debug messages specific to *mcbsp.c* or a specific debug message within *mcbsp.c*.

Use the command below to enable all messages within *mcbsp.c*.

```

DM-37x# echo "file mcbsp.c +p" >
/sys/kernel/debug/dynamic_debug/control

```

Use the command below to enable only the message on line 120 in *mcbasp.c*.

```
DM-37x# echo "file mcbasp.c line 120 +p" >
/sys/kernel/debug/dynamic_debug/control
```

To disable a message or set of messages, use the *-p* command. More information about using dynamic debug can be found [here](#).⁴¹

11.4 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.

```
DM-37x# lsmod
Module                Size  Used by
g ether                57800  0
DM-37x#
```

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

```
DM-37x# modinfo <module_name>
```

The *systool* command lists the options that are set for a loaded module.

```
DM-37x# systool -v -m <module_name>
```

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

```
DM-37x# modprobe -c | less'
```

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

```
DM-37x# modprobe -c | grep <module_name>
```

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

```
DM-37x# modprobe --show-depends <module_name>
```

⁴¹ <https://www.kernel.org/doc/Documentation/dynamic-debug-howto.txt>

For example:

```
DM-37x# modprobe --show-depends wl12xx
insmod /lib/modules/3.0.0-BSP-dm37x-2.4-
2/kernel/drivers/net/wireless/wl12xx/wl12xx.ko
DM-37x# modprobe --show-depends wl12xx_sdio
insmod /lib/modules/3.0.0-BSP-dm37x-2.4-
2/kernel/drivers/net/wireless/wl12xx/wl12xx.ko
insmod /lib/modules/3.0.0-BSP-dm37x-2.4-
2/kernel/drivers/net/wireless/wl12xx/wl12xx_sdio.ko
```

The *is* command lists available module parameters.

```
DM-37x# ls /sys/module/<module_name>/parameters
```

12 Reporting Problems

For more information, please look in the *doc/* sub-directory of the LTIB installation. There you will find an FAQ and other documentation describing the proper use of the program. If you do have a bug to report to Logic PD, please include the following information:

1. Full name of LTIB package (ISO or TAR archive) and its MD5SUM
2. Build log that you obtained by entering the LTIB directory and entering the command below

```
bash$ ./ltib -f 2>&1 | tee my-ltib-error-log
```

13 Appendix A: Enable MCS0 and MCS7

MCS0 and MCS7 are enabled by default when the omap3logic_defconfig is used in the kernel. If the kernel is configured to use the omap3logic_defconfig-performance on the Torpedo + Wireless, MCS0 and MCS7 may not be available for Wifi operation. If MCS0 or MSC7 are desired:

1. Run Itib to configure the kernel:
2. Go into the Kernel Hacking menu.
3. Select "Lock debugging: detect incorrect freeing of live locks"
4. Select "Lock debugging: prove locking correctness"
5. Select "Lock usage statistics"
6. Exit and build