

# NXP LPC32XX Linux User's Manual

Version history

<u>Release</u>	<u>Date</u>	<u>Comments</u>
1.00	01/15/2009	Initial release
1.01	2/12/2009	Minor fixes from feedback comments

## Table of contents

<b>1</b>	<b>Introduction.....</b>	<b>5</b>
1.1	<i>Copyrights and limitations .....</i>	5
1.2	<i>Where to start.....</i>	5
1.3	<i>Required hardware and software .....</i>	5
1.3.1	Host system requirements.....	5
1.3.2	Target board requirements.....	6
1.4	<i>About this document .....</i>	6
<b>2</b>	<b>BSP overview.....</b>	<b>7</b>
2.1	<i>Supported hardware .....</i>	7
2.2	<i>BSP information .....</i>	7
2.2.1	LTIB integration.....	7
2.2.2	u-boot 1.3.3.....	7
2.2.3	Linux.....	8
2.2.4	Pre-built GCC 4 toolchain .....	11
<b>3</b>	<b>Building your Linux system .....</b>	<b>12</b>
3.1	<i>Downloading and installing LTIB.....</i>	12
3.2	<i>Configuring LTIB .....</i>	12
3.2.1	First time running LTIB.....	12
3.2.2	LTIB options .....	13
3.2.3	LTIB build cycle.....	14
3.2.4	Re-building the system .....	14
3.2.5	More on LTIB.....	15
<b>4</b>	<b>Linux deployment methods.....</b>	<b>16</b>
4.1	<i>Install u-boot .....</i>	16
4.2	<i>Setting up kernel boot.....</i>	16
4.2.1	Kernel command line.....	16
4.2.2	Network kernel image boot .....	16
4.2.3	Booting the kernel from NAND .....	17
4.3	<i>Mounting the root filesystem .....</i>	17
4.3.1	NFS mounted root filesystem .....	18
4.3.2	EXT2 (uncompressed) filesystem on an SDMMC card .....	18
4.3.3	JFFS2 root filesystem .....	18
<b>5</b>	<b>Additional information.....</b>	<b>20</b>
5.1	<i>Getting the BSP files.....</i>	20
5.1.1	Where to get the Linux u-boot and kernel patches .....	20
5.1.2	Where to get the pre-built toolchain .....	20
5.2	<i>Installing and setting up u-boot.....</i>	20
5.2.1	Phytec 3250 board .....	20
5.2.2	Hitec 3250 board .....	21

5.2.3	Booting the kernel image from MTD (NAND FLASH).....	22
5.2.4	LTIB and GCC toolchain compatibility and dependency issues .....	24
5.2.5	Board variants.....	24
5.2.6	Tftp and NFS help .....	24
5.2.7	Other links .....	25

## 1 Introduction

This document details the Linux 2.6.27.8 BSP for the LPC32xx SoC (System on Chip). This free BSP has been developed to support the LPC32xx SoC from NXP. The BSP provides an complete Linux port for the Phytex PHY3250 platform using the LPC3250 MCU, but can be easily ported to other platforms.

This document covers many aspects of the Linux and the Linux operating system. Some of the areas covered include:

- Peripheral support in the Linux 32XX BSP
- Board customization and new target hardware
- System and target board setup on the Phytex 3250 board
- Using LTIB to build a complete Linux system
- Deployment methods for the target board

### 1.1 Copyrights and limitations

The LPC32xx BSP is provided free of charge and with no support from NXP. Portions of the BSP are copyrighted by NXP Semiconductors.

### 1.2 Where to start

If you have limited or no experience with building Linux based systems, you should start with using LTIB (Linux Target Image Builder). LTIB handles most of the hard work for you deploying as complete Linux system with an easy to use menu interface. If you are this type of user, start at section 3.

If you already have experience setting up, building, and deploying a Linux system, you may want to skip to just getting the necessary patches for u-boot and the Linux kernel. If you are this type of user, jump to section 5.1 to learn where to get the patches. A pre-built version of u-boot.bin is available from the NXP website to help speed up the process. A pre-built u-boot image can be downloaded from NXP's website.

### 1.3 Required hardware and software

#### 1.3.1 Host system requirements

To develop Linux for the LPC32X0, a host PC running the Linux operating system is needed. Because of the many variations of Linux releases and supported packages, it is unknown if the tools included with the BSP will work correctly on a specific release of Linux.

This BSP and the supporting tools have been tested with the Fedora 9 Linux release. Although other releases may work fine, they are currently untested.

In some cases, default Linux distributions may lack all the necessary packages to build Linux. When developing, examine the error messages closely – they usually indicate if the error occurred due to a missing package. If this type of error occurs, add the missing package using *yum* or the software manager and try the build again.

### 1.3.2 Target board requirements

The LPC32xx Linux port supports the following boards.

#### 1.3.2.1 Phytex PhyCore 3250 board

The Phytex PhyCore 3250 board has been extensively tested with the Linux port. The supported versions of the board include:

CPU module 1304.0 and 1304.1

Carrier board 1305.0, 1305.1, 1305.2 and 1305.3

LCD module 1307.0 and 1307.1

Some versions of the board have slight logic differences that impact software. When configuring the Linux port, the specific versions of the target hardware can be selected prior to the kernel build.

The Phytex board includes a bootloader called S1L that sets up the target hardware prior to the Linux bootloader. The latest version of the bootloader can be downloaded from NXP's of Phytex's website

#### 1.3.2.2 Hitex LPC3250 stick

The Hitex LPC3250 stick is not yet supported.

## 1.4 About this document

The intended use of this document is to provide initial help in developing a Linux operating system for the LPC32X0 MCU. The steps required to download the BSP all the way to deploying Linux to the Phytex PHY3250 board are covered.

This document is broken into the following sections:

BSP overview

*A general overview of the LPC32xx port of Linux*

Getting the BSP

*Where to get the patches, toolchain, and LTIB*

Building your Linux system

*Using LTIB to configure and build Linux, Linux kernel configuration*

Linux deployment methods

*NFS deployment, SDMMC (EXT2) card deployment, MTD (JFFS2) deployment*

Additional information

*Other resources, links, and other information*

## 2 BSP overview

This section gives a brief overview of the BSP contents, features, and supported peripherals. This BSP has been developed to support Linux kernel 2.6.27.8.

The term ‘BSP’ refers to the files specific to the LPC32xx Linux port. These include the kernel and u-boot patch file(s), pre-built GCC toolchain, and necessary LTIB files to build a Linux distribution.

Not all the files are necessary to build a Linux based system for the LPC32xx. Developers are welcome to use their own toolchains or develop a Linux system without using LTIB.

### 2.1 Supported hardware

This BSP supports the NXP LPC32xx MCU and most of its built-in peripherals. Several external peripherals that may be required to use optional interfaces are also supported.

### 2.2 BSP information

#### 2.2.1 LTIB integration

LTIB (Linux Image Target Builder) provides a convenient method to build the bootloader, kernel image, and root filesystem and then deploy them to your target. LTIB provides an environment that allows easy setup, configuration, and build of the boot loader, Linux kernel, deployment methods, build tools, etc. If LTIB is used to develop a complete Linux system, the work of downloading and installing packages, building the root filesystem, and an assortment of other tasks is greatly simplified.

#### 2.2.2 u-boot 1.3.3

u-boot is the Linux bootloader supported by this BSP. u-boot provides support for booting the Linux kernel through Ethernet or from NAND FLASH.

##### 2.2.2.1 U-boot for the Phytex 3250 board

u-boot relies on the Stage 1 Loader (S1L) to pre-initialize the board and memory prior to executing. See the phy32xx\_bl.pdf file included with the LPC32xx BSP for more information on S1L. Once S1L has initialized the board, it will load and start u-boot from one of the supported S1L boot sources. Although u-boot relies on S1L for system pre-initialization, the S1L code is available for free from NXP’s website and can be easily added to the startup code of u-boot for developers that don’t want to use S1L.

The PHY3250 version of u-boot supports the following features:

- Persistent configuration of u-boot and boot parameters
- Kernel image boot from NAND FLASH or Ethernet
- Ethernet network configuration

### 2.2.2.2 U-boot for the Hitex 3250 stick

This is not yet supported.

## 2.2.3 Linux

This section describes the system and driver support included with the LPC32xx Linux 2.6.27.8 BSP.

### 2.2.3.1 LPC32xx architecture support

The following LPC32xx peripherals are supported as part of the core system architecture:

- Timer 0 (system tick)
- Interrupt controller
- Clock and power manager (Clock query and control for various drivers)
- DMA

### 2.2.3.2 LPC32xx driver support

The following Linux drivers are provided as part of the BSP:

- Watchdog timer (modified PNX4008 driver)
- Color LCD controller (AMBA CLCD driver)
- SD card controller (modified AMBA driver MMC driver with DMA)
- Ethernet controller
- Touchscreen controller
- Battery backed RTC
- Standard UARTs (up to 4)
- High speed UARTs (up to 3)
- SPI (up to 2 implemented with the SSP interface)
- Key scanner
- I2S with DMA and the UDA1380 CODEC
- I2C (2 dedicated channels, 1 OTG channel)
- USB host with the ISP1301 transceiver
- MTD (NAND SLC controller)

#### 2.2.3.2.1 *Phytec 3250 board driver support*

The following additional support is provided for the PHY3250 board:

- SMSC Ethernet PHY
- PFC8564 RTC clock (I2C)
- AT256 serial EEPROM (SPI)

#### 2.2.3.2.2 *Hitex 3250 board driver support*

This is not yet supported.

### 2.2.3.3 LPC32xx driver overview

This section explains which drivers to include in the Linux kernel configuration if you want them in your Linux kernel image.

### 2.2.3.3.1 Architecture driver overview

Some Linux drivers are new to the LPC32xx, while existing drivers were used when possible. The table below shows the kernel CONFIG\_ values needed to enable the kernel support for that interface.

Functionality	Driver	Kernel CONFIG_ selection
VFP (hardware float)	ARM926 VFP driver	CONFIG_VFP
MTD (NAND)	LPC32xx NAND SLC driver	CONFIG_MTD_NAND_SLC_LPC32XX
Ethernet 10/100	LPC32xx Ethernet MII driver	CONFIG_LPC32XX_MII
Key scanner	LPC32xx key scanner driver	CONFIG_KEYBOARD_LPC32XX
Touchscreen	LPC32xx touchscreen driver	CONFIG_TOUCHSCREEN_LPC32XX
Standard serial ports	8250 UART driver	CONFIG_SERIAL_8250
High speed UARTs	LPC32xx high speed UART driver	CONFIG_SERIAL_HS_LPC32XX
I2C	PNX4008 I2C driver	CONFIG_I2C_PNX
SPI	LPC32xx SSP/SPI driver	CONFIG_SPI_LPC32XX
Watchdog timer	PNX4008 watchdog timer driver	CONFIG_LPC32XX_WATCHDOG
LCD	ARM PL110 driver	CONFIG_FB_ARMCLCD
I2S audio (ALSA)	LPC32xx audio driver	CONFIG_SND_LPC3XXX_SOC
UDA1380 audio CODEC	UDA1380 CODEC driver	CONFIG_SND_LPC3XXX_SOC_UDA1380
USB host	OHCI/PNX drivers	CONFIG_USB_OHCI_HCD
SD/MMC	ARM PL180 driver (modified)	CONFIG_MMC_ARMMMC
RTC	LPC32xx RTC driver	CONFIG_RTC_DRV_LPC32XX

Some interfaces require additional configuration as shown in the table below. For example, specific I2C ports can be enabled and disabled here to further customize the kernel build.

Functionality	Driver	Kernel CONFIG_ selection
Standard UART enables	8250 UART driver	CONFIG_MACH_LPC32XX_UART5_ENABLE CONFIG_MACH_LPC32XX_UART3_ENABLE CONFIG_MACH_LPC32XX_UART4_ENABLE CONFIG_MACH_LPC32XX_UART6_ENABLE

High speed UART enables	LPC32xx high speed UART driver	CONFIG_MACH_LPC32XX_HAUART1_ENABLE CONFIG_MACH_LPC32XX_HAUART2_ENABLE CONFIG_MACH_LPC32XX_HAUART7_ENABLE
I2C channel enables	PNX4008 I2C driver	CONFIG_MACH_LPC32XX_I2C0_ENABLE CONFIG_MACH_LPC32XX_I2C1ENABLE CONFIG_MACH_LPC32XX_USBOTG_I2C_ENABLE
SPI channel enables	LPC32xx SSP/SPI driver	CONFIG_MACH_LPC32XX_SSP0_ENABLE CONFIG_MACH_LPC32XX_SSP1_ENABLE

### 2.2.3.3.2 Board driver overview

The tables below shows the kernel CONFIG\_ values needed to enable the kernel support for interfaces specific to a target board. These drivers are usually not related to the LPC32xx microcontroller.

#### 2.2.3.3.2.1 Phytex 3250 board

Functionality	Driver	Kernel CONFIG_ selection
SMSC ethernet PHY	SMSC PHY driver	CONFIG_SMSC_PHY
PFC8564 RTC clock	PCF8563 driver	RTC_DRV_PCF8563
AT256 serial EEPROM	AT serial EEPROM driver	CONFIG_SPI_AT25

Other options required by the Phytex 3250 board:

Functionality	Driver	Kernel CONFIG_ selection(s)
Enable Phytex board <sup>1</sup>	NA	CONFIG_MACH_PHY3250
Audio on I2S1 <sup>1</sup>	LPC32xx audio driver	CONFIG_SND_LPC32XX_USEI2S1
LCD module revisions <sup>2</sup>	NA	CONFIG_PHY3250_QVGA_PANEL_1307_0 CONFIG_PHY3250_QVGA_PANEL_1307_1
Carrier board revisions <sup>2</sup>	NA	CONFIG_PHY3250_CARRIER_1305_0 CONFIG_PHY3250_CARRIER_1305_1 CONFIG_PHY3250_CARRIER_1305_2 CONFIG_PHY3250_CARRIER_1305_3
CPU module revisions <sup>2</sup>	NA	CONFIG_PHY3250_CPU_MODULE_1304_0 CONFIG_PHY3250_CPU_MODULE_1304_1

<sup>1</sup>Must be selected on the Phytex board for function to work..

<sup>2</sup>Option must be selected based on CPU module, LCD module, or carrier board version.

#### 2.2.3.3.2.2 Hitex 3250 board

This is not yet supported.

### **2.2.3.3 Board neutrality in the BSP**

The entire BSP is targeted to the LPC32xx without any specific target board. Drivers that require board specific functions use driver callbacks that are passed through the driver's platform data pointer. This allows the drivers to remain unbound to any specific board.

Specific board drivers functions needed by architecture drivers are defined in the kernel file tree at `./arch/arm/mach-lpc32xx/board-xxx.c`. For example, the `board-phy3250.c` file contains necessary board support for the MMC driver, LCD driver, SPI driver, etc. as related to the Phytex 3250 board. Other boards are supported with other board files.

To port the Linux kernel to new boards, only a new `board-<platform>.c` file is needed with the required architecture functions supported by the board.

### **2.2.4 Pre-built GCC 4 toolchain**

Building a cross toolchain can be quite a task. A pre-built toolchain for building the LPC32xx kernel, u-boot, and root filesystem is included as part of the BSP. This toolchain is based on GCC 4.3.2 with GLIBC 2.7. The toolchain has support for the VFP unit in the LPC32xx ARM926EJ-S core.

## 3 Building your Linux system

This section explains how to configure and build a Linux system for the Phytex 3250 board using LTIB. Deployment of the root filesystem will be handled via NFS.

### 3.1 Downloading and installing LTIB

LTIB needs to be downloaded and installed on the Linux host machine. To start this process, go to <http://www.bitshrine.org/> and follow the instructions there to download and install LTIB. *It is highly recommended to use the netinstall script to install LTIB or use a direct CVS download (explained in the LTIB FAQ). Using a fixed snapshot image may not get you the latest files.*

Installation of LTIB should be followed as per the instructions on the [www.bitshrine.org](http://www.bitshrine.org) website. The LPC32xx BSP will be automatically downloaded and installed *after* the LTIB packages have been installed from the web and the Phytex 3250 board has been selected in the LTIB platform selection menu. The LTIB installation package can take a few hours or more to install. *Per the LTIB instructions, make sure you perform your build with a user account – do not use root!*

After installation is complete, a screen will appear that will allow you to select a platform. Select the “NXP LPC32xx on the Phytex 3250 board” platform and save the configuration. You will then be passed to the LTIB configuration menu.

### 3.2 Configuring LTIB

#### 3.2.1 First time running LTIB

If this is the first time you are running LTIB, a default configuration will be provided for you that will setup the pre-built toolchain, build u-boot, build a kernel image, and setup a root filesystem. A pre-selected Linux kernel configuration is also provided with most of the supported interfaces selected for the LPC32xx and the Phytex 3250 board. A pre-selected set of packages has also been made for your root filesystem to allow robust function. One of the pre-selected packages is *busybox*, which will provide most of the basic system command line functions and the console shell. *Busybox* is also preconfigured as part of Phytex 3250 port of LTIB. Deployment options have also been setup, with NFS being the default root filesystem mount setup.

Although each LTIB configuration items is explained here, it is recommended that the default selections and configurations be tried for the first build to verify that everything is working. This is the fastest way to verify that your host system is correctly configured and that you target board is working properly. For now, just exit the menu and save the configuration. Once you exit the menu, the build process will start.

*The pre-selected Linux kernel configuration is setup for specific versions numbers of the Phytex 3250 boards. Although these selected versions should work fine for most boards, they may need to be adjusted. To change these values, you will need to bring up the Linux kernel configuration menu and select the correct board and module versions in the System Type - - -> menu. This is explained in Section 5.2.5.1.*

*The next time you LTIB, the LTIB configuration menu will not show up and the build process will start again with the previously saved options. You can show the menu again with the LTIB -config option.*

### 3.2.2 LTIB options

The LTIB configuration menu allows some customization over how your Linux system is built. The options are broken into the following categories – toolchain, bootloader, kernel, package list, system configuration, and deployment. Specific options for each category are selected from the LTIB menus. A general overview of each section is provided.

#### 3.2.2.1 Toolchain selection and setup

You can either select a pre-built toolchain or setup your own toolchain with LTIB.

Start by selecting the “Target C library type). Selecting it will enter into a submenu that will allow choices between GLIBC and UCLIBC. Your choice of a pre-built toolchain is tailored by this option. *Currently, only GCC 4.3.2 with GLIBC 2.7 is supported as a pre-built toolchain option.*

If you select GLIBC as the C library, you have an option of selecting a toolchain of “GCC-4.3.2-glibc-2.7” or “custom”. If you select the “custom” option, you will need to provide your own toolchain as well as the toolchain path, toolchain prefix, and toolchain build options.

If you select the pre-built GCC 4.3.2 toolchain, the toolchain will be download and installed from GPP and use the presetup toolchain options.

#### 3.2.2.2 Bootloader selection and options

With the “bootloader choice” option, you can select either the u-boot bootloader or “none”. If you select none, the bootloader and it’s patches will not be downloaded and built.

#### 3.2.2.3 Kernel build options

You can choose which kernel (version and board) in the kernel section of the menu. Selecting the “Configure the kernel” option will bring up the kernel configuration menu before the kernel starts building. If the “Leave the kernel sources after building” option isn’t selected and the Linux source tree hasn’t been built yet, the Linux source tree will be erased after the Linux kernel has been built.

### 3.2.2.4 Package list

This submenu allows you to select the packages you want in your Linux system. Some options are important for building a root filesystem. A few select packages have been pre-selected including busybox and the device nodes. *It is possible that the some packages may not compile with the ARM architecture.*

### 3.2.2.5 Target system configuration

This submenu allows customization of the kernel startup. You can set up kernel options such as networking addresses, custom startup, or start services. Depending on which packages you install, the menu may change to provide additional support for this installed packages.

### 3.2.2.6 Target image generation

This submenu allows further customization of the target image. You can specify the image type to build (such as JFFS2, compressed EXT2, or a flat NFS tree) and specify if the target filesystem is read-only.

## 3.2.3 LTIB build cycle

After all the LTIB options are selected and you exit LTIB (and save the configuration), the LTIB build cycle will start. The build cycle downloads and builds any packages enabled in the LTIB configuration and then generates the target root filesystem image. LTIB will only build packages that have changed since the last build (generally).

A shortened build cycle is similar to this:

- Download and install toolchain (if not using custom toolchain)
- Download, build, and install support packages
- Build root filesystem skeleton
- Move target libraries to root filesystem
- Download u-boot and patches and build it – move u-boot to root filesystem
- Download Linux kernel and patches and build it – move kernel to root filesystem
- Download needed target packages and build them – move to root filesystem
- Setup kernel boot configuration
- Buld root filesystem image (ie, JFFS2)

The generated root filesystem area is in your LTIB directory under `./rootfs`. You can usually NFS mount this directory. The `u-boot.bin` (u-boot image) and `uImage` (kernel image) files are located in boot directory of the root filesystem.

## 3.2.4 Re-building the system

To rebuild the system, invoke `ltib` again. Note that some options on the LTIB configuration menu may need to be selected to force builds of some components (such as the Linux kernel). The LTIB configuration menu can be shown with the `-config` option.

### 3.2.5 More on LTIB

Go to <http://www.bitshrine.org/> for more detailed information on the use of LTIB.

## 4 Linux deployment methods

You've built u-boot, the Linux kernel, and root filesystem as explained in Section 3. This section explains how to get the Linux images running on the board.

On powerup or reset, u-boot loads and starts the Linux kernel. The Linux kernel can be either loaded over the network using tftp or from the NAND FLASH if it was previously saved there.

Once the kernel is up and running, the root filesystem is usually mounted. The filesystem can be mounted from RAM, an SD card, NAND using JFFS2, or over the network using NFS.

### 4.1 Install u-boot

Before the Linux kernel can be booted, the u-boot bootloader needs to be put on the target board. See Section 5.2 on how to setup u-boot.

### 4.2 Setting up kernel boot

#### 4.2.1 Kernel command line

The u-boot bootargs environment variable contains the command line passed to the Linux kernel when it's executed from u-boot. The kernel command line defines options such as the console, network configuration, root filesystem location, MTD partitions, etc.

The standard console on UART5 can be setup for 115.2Kbps-8-N-1 operation by setting bootargs to:

```
bootargs=console=ttyS0,115200n81
```

The Linux root filesystem is setup by this command line. The bootargs variables will be edited and added to in the different deployment subsections. All command line options are beyond the scope of this document.

#### 4.2.2 Network kernel image boot

To boot the kernel over the network using tftp, set the following environment variables in u-boot.

```
bootfile=uImage  
fileaddr=80100000  
serverip=<your host's IP address>  
bootcmd=dhcp; bootm  
bootdelay=1
```

After the environment variables are setup, save them and use the dhcp command to verify that the kernel loads. An example output for a host machine located at address

192.168.1.62 is shown below:

```
uboot> setenv bootfile uImage
uboot> setenv fileaddr 80100000
uboot> setenv serverip 192.168.1.100 (Used your host machine address here!!!)
uboot> bootcmd 'dhcp; bootm'
uboot> bootdelay 1
uboot> saveenv
Saving Environment to NAND...
Erasing Nand... Writing to Nand... done
uboot> dhcp
      HW MAC address: 00:50:C2:8C:ED:B9
ENET:auto-negotiation complete
ENET:Link status up
ENET:FULL DUPLEX
ENET:100MBase
BOOTP broadcast 1
DHCP client bound to address 192.168.1.62
TFTP from server 192.168.1.100; our IP address is 192.168.1.62
Filename 'uImage'.
Load address: 0x80100000
Loading:
#####
#####
done
Bytes transferred = 1638108 (18fedc hex)
```

*Make sure your kernel image (uImage) is available in your tftp file area. Either link to the file in the LTIB root filesystem or copy the file there. See Section 5.2.6.1 for information on setting up a tftp server.*

### 4.2.3 Booting the kernel from NAND

See Section 5.2.3 for how to setup your kernel to boot from NAND FLASH.

## 4.3 Mounting the root filesystem

The root filesystem mount point is selected through the kernel command line. This command line is specified by the u-boot bootargs environment variable and it passed to the Linux kernel when it is started. A few examples of root filesystem mount options are shown in this section.

### 4.3.1 NFS mounted root filesystem

The NFS mounted root filesystem is the easiest to do, but ties the target board to the host machine through the network interface. This is great for development, but is impractical for stand alone products.

*Prior to setting up NFS, be sure to export your root filesystem via the NFS server. For the LTIB generated root filesystem, this is located in the rootfs directory of where you installed LTIB See Section 5.2.6.2 for help on setting up NFS.*

Setting up the Phytex 3250 board to use an NFS mounted file system requires setting up the command line argument to use the /dev/nfs for the root directory. A sample command line argument is shown below (this assumes a host IP address of 192.168.1.51 and will have to be changed for any specific machine).

```
bootargs= 'console=ttyS0,115200n81 root=/dev/nfs rw  
nfsroot=192.168.1.51:/home/user/ltib/rootfs ip=dhcp init=/sbin/init'
```

This specific command line tells the kernel to use ttyS0 (UART 5) for the console at 115K-8-N-1 parameters, a read/write filesystem from /dev/nfs with nfsroot pointing to the root filesystem on a machine at IP address with the associated path. The Phytex board will get an IP address via DHCP and then initialize the program at /sbin/init (which will likely link to busybox).

One bootargs has been setup, u-boot will tell the kernel to mount the root filesystem from the host machine at 192.168.1.51.

### 4.3.2 EXT2 (uncompressed) filesystem on an SDMMC card

To use an SD card with an EXT2 filesystem as a root filesystem, first partition an SD card for EXT2 and format the partition on the host machine. Copy the root filesystem from the LTIB rootfs directory to the root directory of the EXT2 formatted SD card. Plug the SD card into the target board and powerup or reset the board. Stop the boot sequence in u-boot and change the bootargs value to mount the root filesystem from the SD card's first partition instead. The bootargs value is below:

```
bootargs= 'console=ttyS0,115200n81 root=/dev/mmcblk0p1 ip=dhcp init=/sbin/init'
```

This specific command line tells the kernel to use ttyS0 (UART 5) for the console at 115K-8-N-1 parameters and use a read/write filesystem from /dev/mmcblk0p1. The Phytex board will get an IP address via DHCP and then initialize the program at /sbin/init (on the SD card which will likely link to busybox).

### 4.3.3 JFFS2 root filesystem

A JFFS2 filesystem allows the root filesystem to be stored and mounted from a partition in NAND FLASH. *Care must be taken when using this approach as accidentally wiping*

*out the wrong area of FLASH could prevent your system from working correctly until FLASH is restored.*

To use this mount method, a JFFS2 images of the root filesystem needs to be created. *This can be done with LTIB target Image Generation submenu by selecting a target image type of JFFS2. You will also need to set the JFFS2 erase block size to 16K instead of the default 64K. After LTIB completes building, the rootfs.jffs2 image is created.*

Start by powering up or resetting the target board and stopping the boot sequence at u-boot. Using u-boot tftp file transfer, transfer the rootfs.jffs2 image to the target board. An example to transfer the file from a host machine at IP address 192.168.1.51 follows:

```
tftp 0x80100000 192.168.1.51:rootfs.jffs2
```

The file is loaded to address 0x80100000. After the file transfer has completed, round up the transfer size to the next block boundary. Each block size is 16384 bytes. If 196234 bytes transferred, the rounded up transfer size is 196608 (0x30000) bytes. *The JFFS2 image will have been built on a block boundary automatically if the block erase size was set correctly when the JFFS2 image was built.*

The 4th MTD partition has been reserved for the Linux root filesystem on the Phytec 3250 board. This partition starts at block 356 (offset 0x590000) with a total of 3740 (4096-356) blocks (size 0x3A70000). The sequence to erase and write the data to the partition is as follows:

```
nand erase 0x590000 0x3a70000  
nand write.jffs2 0x80100000 0x590000 (image transfer size rounded up)
```

*Next, set up the command line to use JFFS2 for the root filesystem.*

```
Bootargs='console=ttyS0,115200n81 root=/dev/mtdblock3 roottype=jffs2 ip=dhcp  
init=/sbin/init'
```

This specific command line tells the kernel to use ttyS0 (UART 5) for the console at 115K-8-N-1 parameters and use a JFFS2 filesystem from /dev/mtdblock3. The Phytec board will get an IP address via DHCP and then initialize the program at /sbin/init (in NAND).

## 5 Additional information

### 5.1 Getting the BSP files

#### 5.1.1 Where to get the Linux u-boot and kernel patches

The Linux u-boot and kernel patches can be directly downloaded from <http://www.bitshrine.org/>.

The links are below:

Linux kernel 2.6.27.8 patch

<http://www.bitshrine.org/gpp/kernel-arm-2.6.27.8-lpc32xx.patch>

u-boot 1.3.3 patch

<http://www.bitshrine.org/gpp/u-boot-1.3.3-lpc32xx.patch>

#### 5.1.2 Where to get the pre-built toolchain

The pre-built toolchain can be directly downloaded from <http://www.bitshrine.org/>.

The link is below:

<http://www.bitshrine.org/gpp/tc-nxp-lnx-armvfp-4.3.2-1.i386.rpm>

### 5.2 Installing and setting up u-boot

u-boot (Universal bootloader) needs to be installed on the target board prior to being able to boot the Linux kernel. This section explains the process of getting u-boot to your target hardware.

#### 5.2.1 Phytec 3250 board

The Phytec 3250 board can store and boot the u-boot image from NAND FLASH or the root directory of the SDMMC card formatted in the FAT filesystem. It is recommended that the u-boot image be placed into FLASH so an SDMMC card doesn't need to be inserted into the board on powerup or reset.

The Phytec board has a built-in bootloader called S1L. This loader will load u-boot from either the SD card or NAND FLASH into SDRAM and started execution of u-boot. U-boot doesn't do most of the board initialization (such as the SDRAM initialization or clock setup) and instead relies on S1L to handle that.

##### 5.2.1.1 S1L

The S1L source code, S1L documentation, latest S1L binary, and S1L installation software are available from NXP's website at:

<http://www.standardics.nxp.com/support/software/lpc32xx.cdl.drivers/>

It is recommended that you update your version of S1L in your board to the latest available version using the link above.

### 5.2.1.2 Setting up S1L to boot u-boot from NAND FLASH

To set up u-boot to load and boot from NAND FLASH, the u-boot.bin image needs to be transferred to the target board's memory first. This can be done in one of 2 ways:

#### Option 1

Use the command `load term raw 0x83fc0000` at the S1L prompt on the serial port and then send the u-boot.bin image to the target board over the serial port. After the transfer has completed, send a break to S1L to return to the prompt.

#### Option 2

Place the u-boot.bin in the root directory of an SDMMC card and insert the card into the target board. Use the command `load blk u-boot.bin raw 0x83fc0000` to load the u-boot.bin image into memory.

Once the image is in memory, use the command `nsave` to save the loaded image into NAND FLASH. Next, use the command `aboot flash raw 0x83fc0000` to tell S1L that you want to load the image in FLASH into memory and boot it on powerup or reset. Use the command `prompt linux> 1` to change the boot timeout to 1 second and the S1L prompt to `linux>`.

The board is now setup to boot u-boot. On powerup or reset, S1L will boot and initialize the board. After about 1 second, it will load the image (u-boot.bin) from FLASH into memory at address 0x83fc0000 and execute it.

### 5.2.1.3 Setting up S1L to boot u-boot from an SDMMC card

To set up u-boot to load and boot from an SDMMC card, copy the u-boot.bin image to the root directory of an SDMMC card and insert the card into the target board. Next, use the command `aboot blk u-boot.bin raw 0x83fc0000` to tell S1L that you want to load the u-boot.bin file on the SDMMC card into memory and boot it on powerup or reset. Use the command `prompt linux> 1` to change the boot timeout to 1 second and the S1L prompt to `linux>`.

The board is now setup to boot u-boot. On powerup or reset, S1L will boot and initialize the board. After about 1 second, it will load the file (u-boot.bin) from the root directory of the SDMMC card into memory at address 0x83fc0000 and execute it.

## 5.2.2 Hitex 3250 board

This is not yet supported.

### 5.2.3 Booting the kernel image from MTD (NAND FLASH)

To setup the target board to load the kernel image from NAND FLASH, a few steps need to be performed in u-boot.

#### 5.2.3.1 Phytex 3250 board

##### 5.2.3.1.1 MTD partitioning scheme

The Phytex 3250 board uses NAND FLASH to store the S1L and u-boot bootloaders, u-boot parameters, and optionally the kernel image and root filesystem. The default partitioning scheme is shown below:

Blocks 0 – 24	S1L bootloader
Blocks 25 – 99	u-boot bootloader _ u-boot parameters
Block 100 -256	Kernel image (is used)
Blocks 356+	root filesystem (JFFS2)

The partitioning scheme can be changed by modifying the MTD table in the board-phy3250.c file. The S1L and u-boot partitions should be changed with care to avoid accidentally erasing your bootloaders.

##### 5.2.3.1.2 Kernel boot setup

Prior to placing the kernel image into FLASH, the kernel image needs to be transferred to the board. Using tftp or another network based method to get the image transferred to your board. The example below shows a tftp transfer of the kernel image from a tftp server at IP address 192.168.1.100.

```

uboot> setenv bootfile uImage
uboot> setenv fileaddr 80100000
uboot> setenv serverip 192.168.1.100 (Used your host machine address here!!!)
uboot> dhcp
    HW MAC address: 00:50:C2:8C:ED:B9
ENET:auto-negotiation complete
ENET:Link status up
ENET:FULL DUPLEX
ENET:100MBase
BOOTP broadcast 1
DHCP client bound to address 192.168.1.62
TFTP from server 192.168.1.100; our IP address is 192.168.1.62
Filename 'uImage'.
Load address: 0x80100000
Loading:
#####
#####
done
Bytes transferred = 1638108 (18fedc hex)

```

Two important values of the transfer are the load address(0x80100000) and number of bytes transferred (1638108). To save the kernel image in FLASH, the kernel image size needs to be rounded up to the next NAND block size. Each NAND block is 0x4000 bytes, so based on the kernel image size of 1638108 bytes, the kernel currently uses  $1638108/16384$  blocks = 99.98 blocks. We need to round this up to 100 and re-compute the kernel image size to save into FLASH. The actual size saved into FLASH will be the new rounded block count times the block size =  $100 * 0x400 = 1638400$  bytes = 0x190000 bytes. The MTD NAND commands in u-boot are then used to unlock, erase, write the number of kernel image bytes, and then relock the NAND data. The 3<sup>rd</sup> MTD partition is used to save the kernel image starting at block 100, or offset  $100*0x4000 = 0x190000$ . The sequence is shown below:

```
uboot> nand device 0
Device 0: NAND 64MiB 1,8V 8-bit... is now current device
uboot> nand unlock 190000 270000
device 0 offset 0x190000, size 0x270000
nand_unlock: start: 00190000, length: 2555904!
NAND flash successfully unlocked
uboot> nand erase 190000 270000
```

```
NAND erase: device 0 offset 0x190000, size 0x270000
Erasing at 0x3fc000 -- 100% complete.
OK
```

```
uboot> nand write 0x80100000 0x190000 0x190000
```

```
NAND write: device 0 offset 0x190000, size 0x190000
1638400 bytes written: OK
```

```
uboot> nand lock 190000 270000
NAND flash successfully locked
```

The last thing to do is tell u-boot to boot using the image from NAND by changing the bootcmd environment variable as shown below:

```
uboot> setenv bootcmd 'nboot 80100000 0 190000; bootm'
uboot> saveenv
Saving Environment to NAND...
Erasing Nand... Writing to Nand... done
uboot>
```

u-boot will now load the kernel image from the NAND FLASH into memory and then execute it.

### 5.2.3.2 Hitex 3250 stick

This is not yet supported.

## 5.2.4 LTIB and GCC toolchain compatibility and dependency issues

LTIB and the pre-built GCC toolchain have been tested on a Fedora Core 9 Linux system. Although they should run fine on modern Linux distributions, it is not possible to test it with every Linux distribution and version. Per field and customer tests, other non-Fedora Linux distributions seem to work fine. Some older versions of Fedora had issues running the pre-built toolchain.

## 5.2.5 Board variants

### 5.2.5.1 Phytec 3250 board

There are several different versions of the Phytec 3250 board hardware. These versions can be identified by examining the part and version numbers of the CPU module, carrier board, and LCD module.

The hardware will have version numbers as follows:

Phytec hardware	Part and version number
CPU module	1304.0 or 1304.1
Carrier board	1305.0, 1305.1, 1305.2, or 1305.3
LCD module	1307.0 or 1307.1

In the Linux kernel configuration menu, go to the System Type - - - > submenu and then the LPC32xx implementations - - - > submenu. Go into each revisions submenu and select the matching board version ID for each piece of hardware. Make sure only one option is selected in each revision category. When the revisions on the submenus match the hardware revisions, exit the menu by continuously selecting Exit. Be sure to save the config when asked. Once you exit the menu, the kernel will rebuild with the new configuration.

*Also see the CONFIG\_ values to selected board and module variants in Section 2.2.3.3.2.1.*

## 5.2.6 Tftp and NFS help

There is a lot of great information on the internet for setting up a tftp or NFS server. The general steps required to get them going are explained here.

### 5.2.6.1 Setting up a tftp server

As a general setup procedure, a tftp server is setup by editing the flags and values in the /etc/xinit.d/tftp file. Several import flags are shown below:

```
disable      = no
server_args  = -s /tftpboot/
```

Make sure the *disable* flag is set to *no* or your tftp service won't be enabled. Also set the *server\_args* value with the directory where your files are located when a tftp request occurs. In this specific setup, the /tftpboot directory is used to store files for the tftp server. If a tftp client requests a path/file, the path/file needs to be located or linked here .

When the target board powers up, it will request the kernel image file 'uImage' from the host via tftp if tftp kernel boot is setup on the target. If uImage is not located or linked in /tftpboot, the transfer will fail.

*If tftp continues to fail, verify that your tftp service is running on the host machine.*

### 5.2.6.2 Setting up an NFS server

As a general setup procedure, the /etc/exports file needs to be edited to add the root filesystem directory for NFS. The change will appear similar to the entry below:  
<ROOTFS-PATH> 192.168.1.\* (rw,no\_root\_squash,no\_subtree\_check,sync)

After the edits are made, you will need to restart the NFS server (or enable it if it is disabled).

*For example, if LTIB is installed in your home directory. The following entry would export your LTIB root filesystem to IP clients in the range of 192.168.1.\*.  
/home/user/ltib/rootfs 192.168.1.\* (rw,no\_root\_squash,no\_subtree\_check,sync)*

### 5.2.7 Other links

Other related links are shown below.

The LPC32xx Common Driver Library (CDL) can be found at <http://www.standardics.nxp.com/support/software/lpc32xx.cdl.drivers/>. This package provides a generic set of reference drivers, startup code, and examples for the LPC32xx and supported boards.

The Phytex PhyCore 3250 board can be found at <http://www.phytec.com/products/rdk/ARM-XScale/phyCORE-ARM9-LPC3250.html>. This board is supported by the Linux LPC32xx BSP.

The NXP standard ICs website is a good resource to get User's Manuals, training materials, or other information. <http://www.standardics.nxp.com/>

Link to the LTIB project at savannah.org  
<https://savannah.nongnu.org/projects/ltib>

Link to the GPP used for the LTIB package pool  
<http://www.bitshrine.org/>