LPC313x Linux Quick Start Guide
Version 2.0

1. Introduction

This quick-start guide explains the steps necessary to get Linux up and running on the Embedded Artist's LPC313x board using LTIB or ELDK frameworks. This guide is meant for users new to Linux or as a method to get a Linux image up and running fast. The current LPC313x Linux BSP supports all the following chips:

- LPC313x series (LPC3131 / LPC3130) : Tested on EA3131 board.
- LPC314x series (LPC3143 / LPC3141) : Tested on NXP internal VAL314x board.
- LPC315x series (LPC3154 / LPC3152) : Tested on EA3152 board.

As per http://www.bitshrine.org/, LTIB is explained as:

"The LTIB (Linux Target Image Builder) project is a simple tool that can be used to develop and deploy BSPs (Board Support Packages) for various target platforms. Using this tool a user will be able to develop a GNU/Linux image for their target platform."

LTIB handles a lot of the work of building a complete Linux system, such as setting up the root filesystem, package selection, system init configuration, etc. Advanced users that prefer not to use LTIB may be interested in just getting the necessary kernel and bootloader patches and starting from there – please see the ELDK section of this guide for the locations of those patches.

As per http://www.denx.de/wiki/DULG/ELDK, ELDK is explained as:

The Embedded Linux Development Kit (ELDK) includes the GNU cross development tools, such as the compilers, binutils, gdb, etc., and a number of pre-built target tools and libraries necessary to provide some functionality on the target system. It is provided for free with full source code, including all patches, extensions, programs and scripts used to build the tools. Some versions of ELDK are available in two versions, which use Glibc or uClibc as the main C library for the target packages. Packaging and installation is based on the RPM package manager.

Note: The description in this document assumes using ELDK version 4.2 for ARM.

The steps covered include the following:

- Using LTIB framework
  - Getting and installing LTIB
  - Configuring LTIB and the build process
- Using ELDK framework
  - Getting and installing ELDK
  - Configuring & building Apex
  - Configuring and building the Linux kernel
- Boot process
  - Loading Apex or u-boot from
    - UART
    - SD/MMC card

- On board SPI-NOR flash
- On board NAND flash
- USB DFU

- Loading kernel & ramdisk using Apex boot-loader from
  - On board NAND flash
  - SD/MMC card (EXT2 formatted)
  - TFTP boot (available for EA LPC31xx Version 2.0 base boards only)
  - UART (Xmodem protocol)
- Porting Apex to LPC313x custom board
- Porting Linux 2.6.x to LPC313x custom board

From start to finish, the process can take anywhere from a few hours to a day depending on factors such as host machine speed, internet connection speed, current host system configuration, etc.

1.1 Copyrights and limitations

The LPC313x 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

The sections in this guide are meant to be followed in sequential order starting with Section 2. Prior to that, the requirements for the host, target, and development environment should be reviewed in Sections 1.2 to 1.6.

Although most procedures use the EA3131 board, the same procedures apply to the EA3152 board.

1.3 Host system requirements

To develop Linux for the LPC313x, 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.

LTIB and the supporting tools have been tested with the Fedora 12 and Ubuntu 9.10 Linux releases. Most packages in LTIB will correctly build on these host machines, although some packages may require additional software installed on the host. Although other releases may work fine, they are currently untested. Your host machine will also need a connection to the internet to download packages for the target system. Packages are downloaded and cached on the host machine as needed.

ELDK and the supporting tools have been tested with the Ubuntu 8.04 (Hardy Heron) and Fedora 9 Linux release. Although other releases may work fine, they are currently untested.

1.3.1 Additional host machine software requirements

As LTIB is installing itself, it may fail due to missing host system packages. If this occurs, carefully read the LTIB error messages and install any other packages on your host machine required by LTIB.

Similarly, as ELDK is installing itself, it may fail due to missing host system packages. If this occurs, carefully read the ELDK error messages and install any other packages on your host machine required by ELDK. For additional help refer to http://www.denx.de/wiki/DULG/ELDK.

1.4 Target board requirements

Embedded Artists' LPC3131 OEM Board (mounted on the LPC31xx Base Board) is required to run the generated boot-loader and kernel images. The board is available from Embedded Artists' at http://www.embeddedartists.com/products/kits/lpc3131_kit.php.

A newer version (2.0) of base boards is also available with on board DM9000 Ethernet controller for Linux development.

1.5 Development environment requirements

The target boot configuration is setup to support RARP, TFTP and DHCP. A RARP, TFTP & DHCP servers should be available on the network where your Linux target board is plugged in. If a DHCP or RARP server isn't available, enter a manual IP configuration for the target network configuration.

1.6 Basic required Linux skills

Skills such as TFTP boot configuration, basic network setup, mounting and using a storage card (such as SD/MMC), and installing new host system packages are a few of the skills assumed to be known by the user in this guide.

There are already a lot of resources on the internet explaining these types of things so they won't be covered here in this guide.

2. Using LTIB framework

2.1 Getting and installing LTIB

Getting and installing LTIB is easy! Open a web browser on your host machine and go to the http://www.bitshrine.org website. Look for the section for LTIB installation. Follow the directions there to 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 and will make updates harder to manage.

LTIB will begin installing by downloading files and packages over the internet. If any errors occurred during installation, correct the error before continuing. Most errors are due to missing or out-of-date packages on the host machine. If you do see missing packages, they can be added or updated with the host machine's software management tools.

Once LTIB has finished installing, run LTIB per the website instructions. A menu should appear requesting selection of the platform. Continue to the next section to setup LTIB to build Linux for the EA3131 board.

2.2 Configuring LTIB

Once LTIB has been installed and executed for the first time, a menu will appear requesting which platform to use with LTIB, Select the Embedded Artists board with either the NXP LPC3131 or LPC3152 SoC option. Choose the Exit option and save the configuration to continue the setup process.

Another menu should shortly appear that allows you to customize the LTIB build options for the EA31xx platform. If this is the first time running LTIB, a number of default options are already selected for you that can be used to build a working Linux system.

Although the default options are fine for the first build, you can change or add build options by selecting LTIB menu items. For example, if you select the "Configure the kernel" option from the LTIB menu, the Linux kernel menu will appear during the build process that allows customization of the kernel settings.

2.3 LTIB build process

The LTIB build process for the EA3131 & EA3152 boards builds the bootloader, the Linux kernel image, and root filesystem. These are built with the toolchain selected in the LTIB main menu and the configurations stored in LTIB,

As the Linux system is being built, the directory called "rootfs" will be built under the ./ltib directory. This directory contains the files of the root filesystem. The u-boot bootloader (u-boot.bin) and the kernel image (uImage) will be placed in the ./rootfs/boot directory.

Files and packages will be built and placed in the rootfs directory's subdirectories based on the package selections in the LTIB package selection menu. These include packages such as busybox, mp3play, mtd-utils, etc.

Based on the "Target System Configuration" LTIB menu item, the startup process of the Linux kernel may be altered. This section of the LTIB menu can be used to adjust network settings or starting services.

Once the rootfs directory has been completely built, the "Target Image Generation" LTIB menu item specifies the type of deployment the Linux system will use on the EA3131 board. If NFS is selected, the root filesystem directory at ./ltib/rootfs is used as the root filesystem mount point. If another option is used such as JFFS2 or ramdisk, another file will be created to be used for deployment.

To save time rebuilding the kernel, make sure the “Leave the kernel sources after building” option is enabled on the first build.

3. Using ELDK framework

For ease of development following directory structure is suggested. Rest of the section assumes this directory structure in its command illustrations.

Create the following structure in your development area:

- `/home/xxx/projects/lpc313x`
  - `/home/xxx/projects/lpc313x/eldk42`
  - `/home/xxx/projects/lpc313x/apex`
    - `/home/xxx/projects/lpc313x/apex/work_1.6.8` <-- this will hold the current working sources for apex
  - `/home/xxx/projects/lpc313x/uboot`
    - `/home/xxx/projects/lpc313x/uboot/work_2009.11` <-- this will hold the current working sources for u-boot
  - `/home/xxx/projects/lpc313x/kernel`
    - `/home/xxx/projects/lpc313x/kernel/work_2.6.28.2` <-- this will hold the current working sources for Linux
  - `/home/xxx/projects/lpc313x/patches`
  - `/home/xxx/projects/lpc313x/temp_dir`
  - `/home/xxx/projects/lpc313x/downloads` <- put all tarballs etc you download from internet here

3.1 Getting and installing ELDK

- Download the latest ISO image of ELDK4.2 for ARM at ftp://ftp.denx.de/pub/eldk/4.2/arm-linux-x86/iso/arm-2008-11-24.iso.
- Follow the instruction listed in ftp://ftp.denx.de/pub/eldk/4.2/arm-linux-x86/distribution/README.html. It is pretty simple. Mount the .iso file using

```
➢ sudo mount -o loop downloads/arm-2008-11-24.iso temp_dir
```

- CD to temp_dir and run the install script and pass the directory where you want to install the tools. Note, the iso contains both pre-built GCC4.2.2 for ARM and also pre-built root file system for ARM target. We will use the pre-built rootfs during our development.

```
➢ cd temp_dir
➢ sudo ./install -d /home/xxx/projects/lpc313x/eldk42
```

- Run the ELDK script described in section "1.6. Mounting Target Components via NFS" of README.html file for future NFS root development.

3.2 Building Apex boot-loader

- Download the Apex1.6.8 tarball from ftp://ftp.buici.com/pub/apex/apex-1.6.8.tar.gz.
- Untar the apex the sources.

```
➢ cd apex
➢ tar -xzf ../downloads/apex-1.6.8.tar.gz
➢ mv apex-1.6.8 work_1.6.8
```

- Apply LPC313x apex patch present as part of the release tarball.

```
➢ cat ../patches/apex-1.6.8_lpc313x.patch | (cd work_1.6.8; patch -
  p1)
```

- Prior to building apex add ELDK tools to your path. From bash shell do

```
➢ export ARCH=arm
➢ export CROSS_COMPILE=arm-linux-
➢ export
  PATH=/home/xxx/projects/lpc313x/eldk42/usr/bin:/home/xxx/projects/
  lpc313x/eldk42/bin:$PATH
```

- To build apex assuming pwd = /home/xxx/projects/lpc313x

```
➢ make -C apex/work_1.6.8 ea313x_v1_config apex.bin
```

- Once build completes the apex binary image for deployment can be found at apex/work_1.6.8/apex.bin
- For systems with version 2.0 I/O boards use ea313x_v2_config.
- To change apex configuration issue the following command.

```
➢ make -C apex/work_1.6.8 ea313x_v1_config menuconfig
```

- Don't forget to backup the modified config.

```
➢ cp apex/work_1.6.8/.config apex/work_1.6.8/src/mach-
  lpc313x/myconfig
➢ make -C apex/work_1.6.8 myconfig apex.bin
```

3.3 Building u-boot boot-loader

- Get the "u-boot-2009.11.tar.bz2" u-boot source code from ftp://ftp.denx.de/pub/u-boot/u-boot-2009.11.tar.bz2 site.
- Untar the u-boot the sources.

```
➢ cd uboot
➢ tar -xjf ../downloads/u-boot-2009.11.tar.bz2
➢ mv u-boot-2009.11 work_2009.11
```

- Apply LPC313x uboot patch present as part of the release tarball.

```
➢ cat ../patches/u-boot-lpc313x-2009.11.patch | (cd work_2009.11;
  patch -p1)
```

- Prior to building u-boot add ELDK tools to your path. From bash shell do

```
➢ source ../../eldk42/eldk_init arm
```

- To build uboot assuming pwd = /home/xxx/projects/lpc313x/uboot

```
➢ make -C work_2009.11 lpc3131ea_config
make: Entering directory
`/home/nxp/projects/lpc313x/uboot/work_2009.11'
Configuring for lpc3131ea board...
make: Leaving directory
`/home/nxp/projects/lpc313x/uboot/work_2009.11'

➢ make -C work_2009.11
```

- You can also build u-boot using following command-line without modifying the environment.

```
➢ make ARCH=arm CROSS_COMPILE=<Toolchain path>/arm-linux-gnu
  distclean
➢ make EA3131_config
➢ make ARCH=arm CROSS_COMPILE=<Path of Tool chain>/arm-linux-gnu-
```

- Once build completes the u-boot binary image for deployment can be found at uboot/work_2009.11/u-boot.bin.

3.3.1 Creating u-boot initial section image

- For UART boot and USB-DFU boot the image has to be sliced into two (a) initial section image (init-u-boot.bin) and (b) main image. For this NXP provides a utility in form of a C file called mkimage.c. One should natively compile this tool on their Linux host machines and use the output file.

```
➢ cp <u-boot_release_dir>/makebootimage.c
  /home/xxx/projects/lpc313x/uboot
➢ cd  home/xxx/projects/lpc313x/uboot
➢ gcc -o mkbootimg makebootimage.c
➢ ./mkbootimg work_2009.11/u-boot.bin work_2009.11/init-u-boot.bin
copy_len: 14a00
```

- Or you could use "dd" command as below

```
➢ dd if=work_2009.11/u-boot.bin of=work_2009.11/init-u-boot.bin
  bs=1024 count=78
```

- For more details u-boot boot process on LPC313x see section "5. Boot process using U-boot boot loader".
- Note, to build u-boot image for EA3152 boards use 'make EA3152_config'. When ea31xx.h is modified always do "make clean" before building *u-boot.bin.*

3.3.2 Creating DFU bootable u-boot image

LPC313x/4x/5x boot ROM supports booting images via Device Firmware Upgrade (DFU) protocol over USB interface. For security reasons boot ROM expects the DFU download image to be TEA encrypted. See LPC313x/4x/5x user manual for more details. NXP provides a Linux command line tool to encrypt boot images as per boot ROM

requirements. To encrypt the initial u-boot section image issue the following commands on Linux PC.

```
➢ tools/unsimgcr -pd uboot/work_2009.11/init-u-boot.bin
  uboot/work_2009.11/init-u-boot.rom
```

- Note, the "*unsimgcr*" tool is available in the Linux BSP tar ball available on NXP website (http://ics.nxp.com/support/documents/microcontrollers/zip/lpc313x.linux.patch.tar.bz2).
- Below is the help screen of "*unsimgcr"* tool.

```
➢ ./unsimgcr
LPC3130/31/41/52 Un-Secure Image Creator Utility v1.1
Command error: need an input & output file names
   Wrong number of input parameters!

Usage: UnsImgCr [options] inputfile outputfile
 -pd             Creates image for USB boot mode.
 -pc             Creates image with CRC check boot mode.
 -p              [default] Creates image with no-CRC check boot mode.
<inputfile> is treated as the LPC31xx image. The signed/encrypted
image is written to the specified output file. It is assumed that the
<inputfile> contains uninitialized header that will be overwritten
with header data.
➢
```

3.4 Building kernel

- Download the kernel sources tarball from ftp://ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.28.2.tar.bz2
- Un-tar the kernel the sources.

```
➢ cd kernel
➢ tar -xjf ../downloads/linux-2.6.28.2.tar.bz2
➢ mv linux-2.6.28.2 work_2.6.28.2
```

- Apply LPC313x kernel patch present as part of the release tarball.

```
➢ cat ../patches/linux-2.6.28.2_lpc313x.patch | (cd work_2.6.28.2;
  patch -p1)
```

- Prior to building kernel add ELDK tools to your path. From bash shell do

```
➢ export ARCH=arm
➢ export CROSS_COMPILE=arm-linux-
➢ export
  PATH=/home/xxx/projects/lpc313x/eldk42/usr/bin:/home/xxx/projects/
  lpc313x/eldk42/bin:$PATH
```

- To build kernel assuming pwd = /home/xxx/projects/lpc313x

```
➢ make -C kernel/work_2.6.28.2 ARCH=arm CROSS_COMPILE=arm-linux-
  ea313x_defconfig zImage
```

- Once build completes the kernel binary image for deployment can be found at kernel/work_2.6.28.2/arch/arm/boot/zImage
- To change kernel configuration issue the following command.

```
➢ make -C kernel/work_2.6.28.2 ARCH=arm CROSS_COMPILE=arm-linux-
  ea313x_defconfig menuconfig
```

- Don't forget to backup the modified config.

```
➢ cp kernel/work_2.6.28.2/.config
  kernel/work_2.6.28.2/arch/arm/configs/myconfig
➢ make -C kernel/work_2.6.28.2 ARCH=arm CROSS_COMPILE=arm-linux-
  myconfig zImage
```

4. Boot process using Apex boot loader

Booting Linux on Embedded Artists' LPC313x boards is a two step process.

1. Loading Apex: LPC313x has on chip bootrom which loads properly formatted images from multiple sources, including SPI Flash, NOR Flash, UART, USB, SD Card, and NAND Flash. The boot interface is selected based on the states of GPIO0, GPIO1 and GPIO2 pins at reset. See LPC313x user manual for more details. The apex.bin image built as part of LTIB and ELDK frameworks has proper image header for bootROM to load the image. Section "4.1 Loading Apex" gives more details on how to prepare the interface to load Apex.
2. Loading kernel and ramdisk: Once Apex boots, Apex is used to load kernel and boot the kernel. Apex boot-loader supports loading kernel from UART, NAND, SD/MMC card, Ethernet and USB interfaces. See section "4.2 Loading kernel & ramdisk using Apex " for more details on how to load kernel & ramdisk from various interface.

4.1 Loading Apex

4.1.1 UART boot

Set the EA LPC3131 board in UART boot mode by setting jumpers – BOOT0(H), BOOT1(H) and BOOT2(L). Configure the serial port on EA3131 board as described in section "4.1.1.1 Configuring the serial port in EA3131 board".

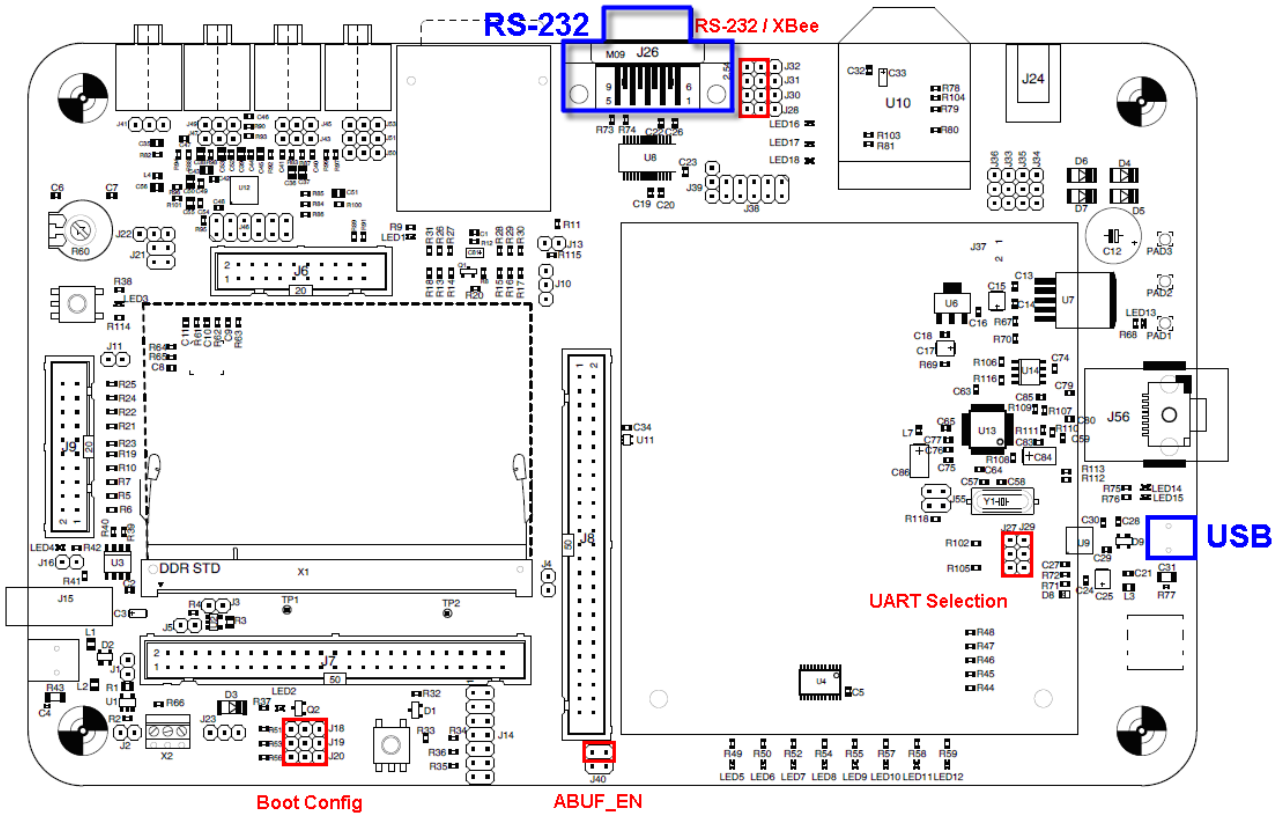


Fig 1. EA3131 jumpers that may need modifications

4.1.1.1 Configuring the serial port in EA3131 board

Two different modes can be used for the serial port:

a) UART (DB9 connector):

In this case, J27/J29 (see EA3131 jumpers that may need modifications) must be set to the Upper position (RS232 position), while J28/J30/J31/J32 must be set to the Left position (RS232 position). A serial cable must be connected between the EA3131 board (DB9 connector) and the PC (serial connector).

b) USB-to-UART bridge (mini-USB connector):

In this case, J27/J29 (see EA3131 jumpers that may need modifications) must be set to the Lower position (USB position). A USB cable must be connected between the EA3131 board (mini-USB connector) and the PC (USB connector). A Virtual COM driver must be installed in this case, so please refer to the EA3131 board's User Manual for specifics instructions.

Note: independently of the above configuration chosen, the EA3131 board can be powered via the mini-USB or the External Power Supply connectors (or even from both connectors at the same time).

4.1.1.2 Start a PC Terminal application program

Configure a terminal application (which should be able to transfer files in binary mode, such as TeraTerm Pro) with 115200-8-n settings. If using USB-to-serial bridge port on EA board, the appropriate Virtual COM port has to be selected. By the time USB-to-serial enumerates, the bootROM of LPC313x would have transmitted the initial string. Hence reset the board using the "reset" button after opening the terminal application.

Note: the default installation of TeraTerm Pro allows only up to COM4 ports. To increase the number of COM ports accessible by TeraTerm Pro, change the following line in TERATERM.INI (C:\Program Files\TTERMPRO):

MaxComPort=4 to MaxComPort=10

4.1.1.3 Load the programmer code

Once the EA313x board is powered and connected to the PC, reset the board. The following message should appear in the terminal window:

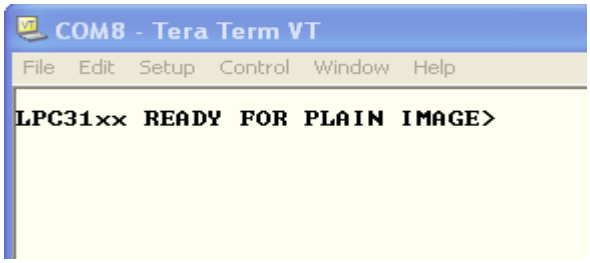
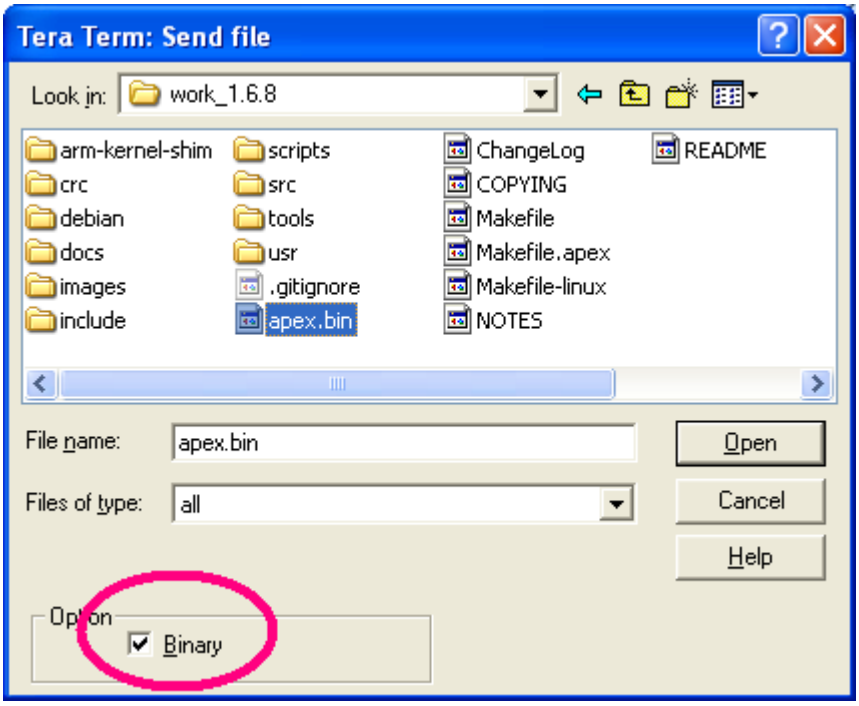


Fig 2. Initial prompt after resetting the board

Select *File -> Send* file option from the terminal application's menu and the next screen will appear:



Don't forget to check the Binary option, as the file to send is a binary image.

Fig 3. Send file selection: apex.bin

Choose the "apex.bin" file, check the Binary option, and press Open to start the file transfer. After download, the following message will appear:

```
LPC31xx READY FOR PLAIN IMAGE>
Download finished

* NAND flash 256 MiB total, 128 KiB erase, 2048 B page
(0x2c/0xaa/0x80/0x15)


APEX Boot Loader 1.6.8 -- Copyright (c) 2004-2008 Marc Singer
  compiled for Unspecified target on 2009.May.15-14:50:24

    APEX comes with ABSOLUTELY NO WARRANTY.  It is free software and
    you are welcome to redistribute it under certain circumstances.
    For details, refer to the file COPYING in the program source.

  apex => mem:0x11029000+0xfafc    (64252 bytes)
  env  => lnand:512k+256k          (empty)

    Use the command 'help help' to get started.

# wait 10

# copy $kernelsrc $bootaddr
# copy ext2://1/zImage 0x30008000
1113872 bytes transferred
# copy $ramdisksrc $ramdiskaddr
# copy ext2://1/ramdisk_image.gz 0x32000000
1659961 bytes transferred
```

```
# boot
ARCH_ID: 9998 (0x270e)
ATAG_HEADER
ATAG_MEM: start 0x30000000  size 0x04000000
ATAG_CMDLINE: (52 bytes) 'console=ttyS0,115200n8 root=/dev/ram0 rw
loglevel=7'
ATAG_INITRD2: start 0x32000000  size 0x00300000
ATAG_END
Booting kernel at 0x30008000...
Uncompressing
Linux.....................................................................
.......... done, booting the kernel
```

4.1.2 SD/MMC card

Set the EA LPC3131 board in SD/MMC boot mode by setting jumpers – BOOT0(L), BOOT1(H) and BOOT2(H). Load apex.bin file to SD/MMC card as described in section "4.1.2.1 Formatting the card". Configure the serial port on EA3131 board as described in section "4.1.1.1 Configuring the serial port in EA3131 board". Configure the terminal window on PC as described in section "4.1.1.2 Start a PC Terminal application program". Insert SD/MMC card into the slot on EA LPC3131 board. Now power-up the board, the Apex boot screen should appear on the terminal window.

4.1.2.1 Formatting the card

This section gives the step-by-step instructions in creating LPC3130/31 bootable partition on SD/MMC cards using "fdisk" utility available on Linux PC.

- Invoke fdisk on the device node associated with SD card. Use 'dmesg' command to figure out "/dev/sdxx" device Linux used for the current USB card reader. The "/dev/sdxx" log entries appear at the very end of the dmesg output.

```
$ sudo fdisk /dev/sde
[sudo] password for xxx_user:
```

- Print the current partition table entries.

```
Command (m for help): p
Disk /dev/sde: 32 MB, 32112640 bytes
1 heads, 62 sectors/track, 1011 cylinders
Units = cylinders of 62 * 512 = 31744 bytes
Disk identifier: 0xde283a86
Device Boot Start End Blocks Id System
/dev/sde1 2 899 27838 6 FAT16
/dev/sde2 900 1011 3472 df BootIt
Command (m for help):
```

- Note, always create "bootit" (partition type 0xDF) partition as second partition. So that when the card is plugged back into a Windows PC it doesn't format "bootit" partition. Windows will not complain as long as the first partition is either FAT or NTFS partition.
- You could use 'm' command under "fdisk" to get help on other "fdisk" commands.
- Delete all existing partitions on the card one at a time

```
Command (m for help): d
Partition number (1-4): 1
Command (m for help): d
Partition number (1-4): 2
```

- Now create new partitions. To specify the amount of space you need to specify start block and end block for each partition. This is usually the cylinders numbers. Since they vary from card to card it is little confusing what to specify. So we create the second partition first with +1M (1 MB size).

```
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (1-1011, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1011, default 1011): +1M
Command (m for help): t
Selected partition 2
Hex code (type L to list codes): df
Changed system type of partition 2 to df (BootIt)
Command (m for help):
```

- Now create first partition of type FAT16 or FAT32. The card used in illustration is 32MB only so we will create FAT16 in this example.

```
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (34-1011, default 34):
Using default value 34
Last cylinder or +size or +sizeM or +sizeK (34-1011, default 1011):
Using default value 1011
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): 6
Changed system type of partition 1 to 6 (FAT16)
Command (m for help): p
Disk /dev/sde: 32 MB, 32112640 bytes
1 heads, 62 sectors/track, 1011 cylinders
Units = cylinders of 62 * 512 = 31744 bytes
Disk identifier: 0xde283a86
Device Boot Start End Blocks Id System
/dev/sde1 34 1011 30318 6 FAT16
/dev/sde2 1 33 1022+ df BootIt
Partition table entries are not in disk order
Command (m for help):
```

- Now write the table and exit from fdisk

```
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
$
```

- Now dump the boot image to /dev/sde2 partition using "dd" command as follows. If you are using latest LPC313x CDL, the bin files generated by make system can be written directly to the card. If not then you need to create the image in the format described in LPC313x User manual Section 6–4.2.

```
$ sudo dd if=./apex.bin of=/dev/sde2 bs=512
[sudo] password for xxxuser:
102+1 records in
102+1 records out
52528 bytes (53 kB) copied, 0.186911 s, 281 kB/s
$
```

- Now the card is ready for booting. Don't forget to "sync" the card before ejecting. Also don't forget to put LPC3130/31 in SD/MMC boot mode.
- The apex.bin file can be found at:
    a. LTIB framework: ./ltib/rootfs/boot
    b. ELDK framework: `/home/xxx/projects/`lpc313x/apex/work_1.6.8/apex.bin

4.1.3 On board SPI-NOR flash

Set the EA LPC3131 board in SPI boot mode by setting jumpers – BOOT0(L), BOOT1(L) and BOOT2(H). Program “apex.bin” into on-board SPI-NOR flash. Refer AN10811 Programming SPI flash on EA3131 boards (with software), V1 (May 1, 2009) application note available on http://www.standardics.nxp.com/support/documents/microcontrollers/?scope=LPC3131 website for SPI-flash programming instructions. Configure the serial port on EA3131 board as described in section “4.1.1.1 Configuring the serial port in EA3131 board”. Configure the terminal window on PC as described in section 4.1.1.2. Now power-up the board, the Apex boot screen should appear on the terminal window.

4.1.4 On board NAND flash

LPC313x patch adds NAND based block device support Apex 1.6.8. LPC313x has NAND controller with HW ECC. A new command called "lpcnand" is added to Apex to format NAND flash device as defined by boot ROM. This command writes block0-page0 with params and block0-page1 with bad block list. The bad block indicator programmed by the NAND vendor will be removed if an erase command is issued to the block. Hence the bad block list should be created before programming any other part of the device. Hence this command should only be executed only once on a board fresh from factory.

```
apex> lpcnand format
```

To program nand flash with apex.bin image,

- Copy the image an ext2 formatted SD card.
- Load apex on the board using UART boot or SD boot method.
- At apex prompt copy the image from SD card to SDRAM .

```
apex> copy ext2://1/apex.bin 0x30008000
62664 bytes transferred
apex>
```

- Now copy the image from SDRAM to NAND flash. Always copy the image to SDRAM as intermediate step since the nand driver in apex always operate on pages boundary. Copying images directly from SD card to NAND is not supported.

```
apex> copy 0x30008000+62664 lnand:128k
```

- Note, the number 62664 is size of the apex.bin image. This number will vary depending on your build image size. But always program apex.bin at offset 128k onwards.

4.1.5 USB DFU

Not supported.

4.2 Loading kernel & ramdisk using Apex

4.2.1 SD/MMC card (EXT2 formatted)

Apex 1.6.8 with LPC313x patch supports loading images from ext2 formatted SD/MMC cards.

4.2.1.1 Preparing SD/MMC card

To format the SD/MMC card do the following:

- Insert card into USB reader/SD slot connected to your Linux PC.
- Open shell terminal and find out the device node SD card is associated. Use 'dmesg' command to figure out "/dev/sdxx" device Linux used for the current USB card reader. The "/dev/sdxx" log entries appear at the very end of the dmesg output.
- Now issue format command on the partition to which you would like to copy the zImage and ramdisk.

```
$ sudo mke2fs /dev/sde1
[sudo] password for xxxx:
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
7584 inodes, 30284 blocks
1514 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=31195136
4 block groups
8192 blocks per group, 8192 fragments per group
1896 inodes per group
Superblock backups stored on blocks:
      8193, 24577

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 31 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
$ sync
$
```

- Mount the formatted partition and copy the zImage and ramdisk files.

```
$ sudo mount /dev/sde1 /media/disk/
$ sudo cp kernel/work_2.6.28.2/arch/arm/boot/zImage /media/disk/
$ sync
$ sudo cp eldk/arm/images/ramdisk_image.gz /media/disk/
$ sync
```

```
$
```

- Note, the above commands show the path of the files when ELDK framework is used to build kernel. For users using LTIB framework the image are available at:
  - Apex bootloader image ---> ./ltib/rootfs/boot/apex.bin
  - Linux kernel image ---> ./ltib/rootfs/zImage
  - Compressed ramdisk (EXT2) ---> ./ltib/rootfs.ext2.gz

4.2.1.2 Configuring Apex to boot from SD/MMC

Once Apex boots to load the kernel and ramdisk issue the following commands at Apex prompt:

```
apex> copy ext2://1/zImage $bootaddr
apex> copy ext2://1/ramdisk_image.gz $ramdiskaddr
```

Note, when LTIB framework is used the ramdisk filename rootfs.ext2.gz should be used in the above command. Also to list all the files present on the SD/MMC card use the following command at apex prompt.

```
apex> info ext2://1/
./
../
lost+found/
apex>
```

As part of Apex configuration the default startup command can be configured to autoboot from SD card. The ea313x_v1_config configuration file present as part of the patch already sets the startup command to boot from SD card. To modify the startup command, change the value of CONFIG_ENV_DEFAULT_STARTUP in ea313x_v1_config file. The following config variables affect the SD boot.

For the LTIB environment, this can be done by checking the “Configure Apex” option from the LTIB main configuration menu (./ltib –config) and then rebuilding. If Apex doesn’t rebuild, use the following command to force Apex to rebuild the next time Apex is run.

```
➢ touch ./ltib/dist/lfs-5.1/apex/apex-common.tmpl
```

```
CONFIG_ENV_REGION_KERNEL="ext2://1/zImage"
CONFIG_ENV_REGION_RAMDISK="ext2://1/ramdisk_image.gz"
CONFIG_ENV_DEFAULT_STARTUP="wait 10;copy $kernelsrc $bootaddr;copy
$ramdisksrc $ramdiskaddr; boot"
```

The apex environment variables can be modified using ‘setenv’ command. The current EA313x implementation of Apex saves the environment on onboard NAND flash. To save the environment use “saveenv” command.

4.2.2 On board NAND flash

The LPC313x patch adds NAND based block device support Apex 1.6.8. Please format the NAND device as per LPC313x boot specification before programming the device. Check section "4.1.4 On board NAND flash" for more details.

To program kernel image into nand flash do the following. Always copy kernel image at 768k offset so that enough block are left for apex and its environment. See apex\work_1.6.8\src\mach-lpc313x\drv-nandc.c file for more details.

```
apex> copy ext2://1/zImage 0x30008000
1113872 bytes transferred
apex> copy 0x30008000+1113872 lnand:768k
```

You can also copy kernel using UART download mechanism built in Apex. To do that issue "xreceive" command as shown below. On terminal application window initiate the file transfer using XMODEM protocol with checksum option.

```
apex> xreceive 0x30008000
C77952 bytes received
apex> copy 0x30008000+77952 lnand:128k
77952 bytes transferred
apex>
```

To load kernel image from NAND do the following.

```
apex> copy lnand:768k+2m 0x30008000
2097152 bytes transferred
apex>
```

Similarly program the ramdisk into "lpcnand:4m" region (ie. at 4 Mbytes offset in NAND device).

JFFS2 based root file system is not tested at the release of this package. But all the need drivers and tools are made available in this release.

4.2.3 Ethernet using TFTP boot

Embedded Artist's LPC313x version 2.0 IO boards have DM9000 Ethernet controller onboard. The board doesn't have MAC-EEPROM populated by default, hence the MAC address has to be stored in on board NAND flash or SPI-flash. Current implementation of Apex provides "eth mac" command to set the mac address. Users could use apex environment variables to save different MAC address for different boards. The following apex screen dump provides command sequence to boot using TFTP. The following commands assume the IP address for server and the target are setting using "rarp" protocol.

```
apex>  eth mac 00:08:ee:00:80:43
apex>  ipconfig rarp
hostip 192.168.1.77
serverip 192.168.1.30
gatewayip 192.168.1.30
apex>  copy tftp://$serverip/zImage $bootaddr
# copy tftp://192.168.1.30/zImage 0x30008000
1114524 bytes transferred
apex>  copy tftp://$serverip/ramdisk_image.gz $ramdiskaddr
# copy tftp://192.168.1.30/ramdisk_image.gz 0x32000000
```

```
1659961 bytes transferred
# boot
```

To configure rarp setting on your development Linux pc do the following:

- Add the following entry to /etc/ethers file. If file is not present create it.

```
00:08:ee:00:80:43 192.168.1.77
```

- Then execute "$sudo arp -f". This will update the arp entries from the "/etc/ethers" file.

Configuring TFTP server on Linux PC is outside the scope of this document.

4.2.4 UART using Xmodem protocol

Apex 1.6.8 when enabled supports XMODEM protocol for serial file transfers. The file transfers are achieved using “xreceive” command.

- Download kernel image by issue following command. On PC terminal application select file transfer using XMODEM protocol and select the zImage file.

```
apex> xreceive $bootaddr
# xreceive 0x30008000
C1114624 bytes received
apex>
```

- Now download the ramdisk image using the method as above.

```
apex> xreceive $ramdiskaddr
# xreceive 0x32000000
C1114624 bytes received
apex>
```

- After both files are downloaded enter ‘boot’ command to boot linux.

4.2.5 USB using mass storage class

Current version (1.6.8) of Apex doesn’t support USB boot mechanism. Since USB interface is the only high-speed communication interface on Embedded Artists’ version 1.0 boards, NXP has created a USB boot mechanism with a 3rd party stack as an external linked library. Due to licensing issue the USB code can’t be distributed under GPL. Hence customers could download the pre-built binary module and do the following to integrate this feature for development purpose only.

- Download usbmsc.d module from NXP microcontroller website (also available in http://ics.nxp.com/support/documents/microcontrollers/zip/lpc313x.linux.patch.tar.bz2 tarball).
- Copy usb/usbmsc.d file to apex working directory at work_1.6.8\src\mach-lpc313x\usb
- Edit the apex configuration to enable “USB boot” using “make menuconfig”. Check ELDK and LTIB configuration sections for more detail on how to edit apex configuration in individual framework environments.
- Build apex for EA313x version 1 board.

- For LTIB framework users the source code is located at ./ltib/rpm/BUILD/apex
  - LTIB deletes the sources after build hence make sure the “Leave the Apex sources after building” option is checked in the LTIB main menu (./ltib –config).

To use USB boot mechanisms do the following.

- A new command "usb" has been added to Apex command list. When you use this command the device enumerates as mass storage drive to the PC/Linux host it is connected to. On PC you could copy only 1 file to the drive as soon as copy finishes within 2 secs the apex will disconnect the drive from PC. The command takes single parameter to specify the type of file (kernel image or ramdisk) you plan to download.

```
apex> usb k   <---- Download zImage. The board will enumerate to PC with
volume label as "zImage Disk"
apex> usb r   <---- Download ramdisk_image.gz. The board will enumerate
to PC with volume label as "rootfs Disk"
```

If no parameter is specified the command assume kernel download.

- After Apex boots, issue commands in following sequence to boot Linux.

```
apex> usb
Downloading Kernel to address 0x30008000
done
apex> usb r
Downloading Ramdisk to address 0x32000000
done
apex> boot
```

5. Boot process using U-boot boot loader

Booting Linux on Embedded Artists' LPC313x boards is a two step process.

1. Loading U-boot: LPC313x has on chip bootrom which loads properly formatted images from multiple sources, including SPI Flash, NOR Flash, UART, USB, SD Card, and NAND Flash. The boot interface is selected based on the states of GPIO0, GPIO1 and GPIO2 pins at reset. See LPC313x user manual for more details. The u-boot.bin image built as part of LTIB and ELDK frameworks has proper image header for bootROM to load the image. For UART & USB-DFU boot modes the u-boot loading is split into two steps. See section "5.1 Loading U-boot" for more details on how to prepare the interface & u-boot image to load u-boot.
2. Loading kernel and ramdisk: Once u-boot boots, u-boot is used to load kernel and boot the kernel. U-boot boot-loader supports loading kernel from UART, NAND, SPI-NOR flash, SD/MMC card, Ethernet and USB interfaces. See section "5.2 Loading kernel & ramdisk using U-boot " for more details on how to load kernel & ramdisk from various interface.

5.1 Loading U-boot

As u-boot is the common boot loader for all LPC313x/4x/5x SOCs, it is designed to support SOCs with 192KB of IRAM and also LPC3130 SOC with 96KB of IRAM. In some cases, the entire u-boot image will not fit easily into internal memory. Moreover the boot ROM on LPC3131/4x/5x SOCs loads images up to 128KB only, while the typical u-boot image is approximately 132KB. Because of this limitation, the u-boot boot process is divided into two stages:

1. **Initial boot stage**:

- In this stage the on-chip boot ROM copies only the first 80K from the boot device into IRAM at address 0x11029000 and transfers control to IRAM.
- This initial part of the image contains code to initialize the processor, clocks and memory controller (SDRAM). After initialization the init code copies the *initial section* code from IRAM to external SDRAM memory and jumps to SDRAM boot stage. The code in this section is carefully crafted to be "position relative code" (ie. No long jumps etc).
- The remaining part of the initial section contains code to initialize the boot interface/devices and *.data* section of complete image.
- Below is the memory layout of the u-boot image.

Initial Section	Boot ROM image header.
	Initi code: Processor, clocks & SDRAM

	initialization code. **Position independent code**.
	Boot devices initialization sections (*.text, .rodata*, etc.).
	Complete *.data* section
Rest of the Image	Remaining u-boot code sections (*.text, .rodata, .bss, .got* etc)

2. **SDRAM boot stage**:
   - In this stage the ARM executes instructions from external SDRAM memory.
   - Once the core enters this stage it copies the whole u-boot image from corresponding boot device/interface and overwrites itself excluding *.data* section (initial 10K).
   - Calls u-boot standard boot procedure start_armboot(). From here on the LPC313x u-boot follows the standard u-boot boot process.

The two stage boot process is transparent to user when booting from NAND flash, SPI-NOR flash and SD/MMC cards interfaces, as single image is programmed to these boot devices. It is apparent while booting from UART and USB-DFU interface. For UART and USB-DFU boot modes the image has to be split into (a) initial section and (b) the whole image. The initial image should be provided when boot ROM request the transfer. Once the initial image is transferred the initial image will initialize SDRAM and re-request the image for transfer, this time whole u-boot image should be provided. See individual boot mode section below for more details.

5.1.1 UART boot

Set the EA LPC3131 board in UART boot mode by setting jumpers – BOOT0 (H), BOOT1 (H) and BOOT2 (L). Configure the serial port on EA3131 board as described in section 4.1.1.1. Also Configure a terminal application on PC as described in section "4.1.1.2 Start a PC Terminal application program".

5.1.1.1 Load the programmer code

Once the EA313x board is powered and connected to the PC, reset the board. The following message should appear in the terminal window:

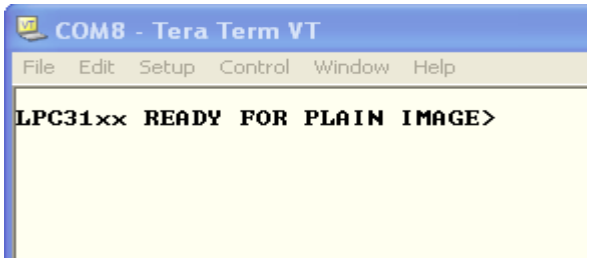
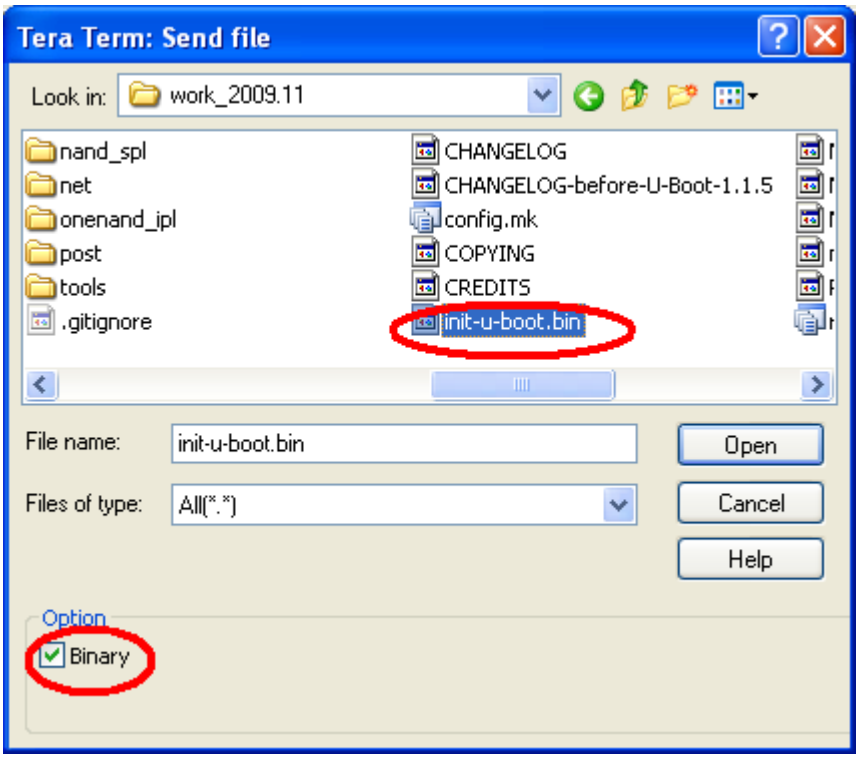


Fig 4. Initial prompt after resetting the board

Select *File -> Send* file option from the terminal application's menu and the next screen will appear:



Don't forget to check the Binary option, as the file to send is a binary image.

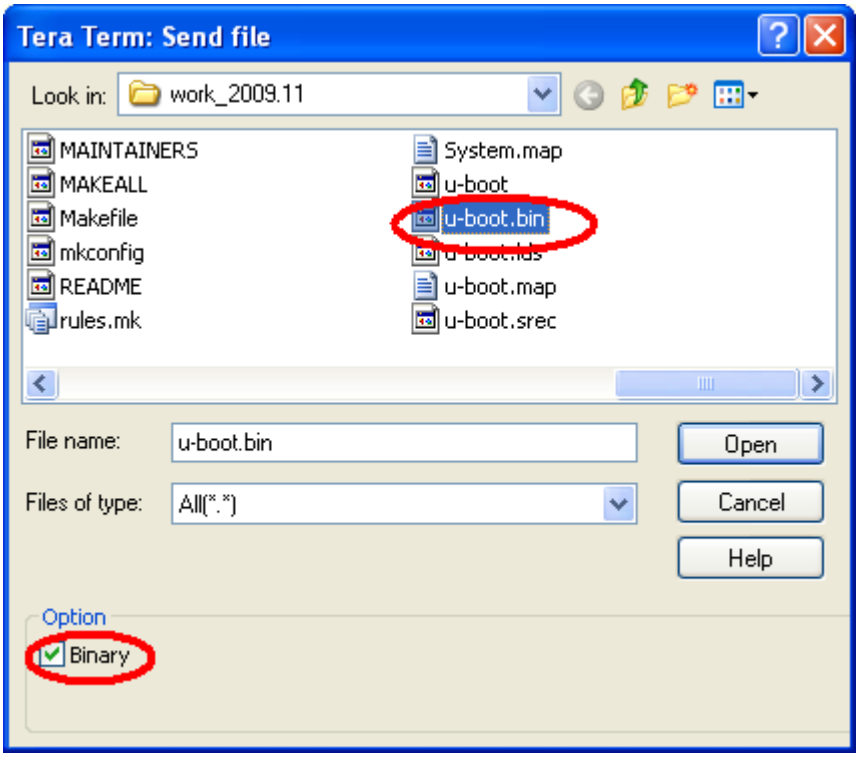
Fig 5. Send file selection: init-u-boot.bin

Choose the initial section image file created from *u-boot.bin* using *mkimage* tool (see section "3.3.1 Creating u-boot initial section image" for more details). Check the "*Binary*" option, and press Open to start the file transfer. After download, the following message will appear:

```
LPC31xx READY FOR PLAIN IMAGE>
Download finished

NAND:  256 MiB
Bad block table found at page 131008, version 0x01
Bad block table found at page 130944, version 0x01
flash params are already written into flash
BOOTMODE: UART
READY TO RECEIVE DATA IN BINARY MODE
```

Now download the complete u-boot.bin image using binary file transfer as described in previous step. Select *File->send* file option from the terminal application's menu and select file as shown in Fig:Send file selection: u-boot.bin.



Don't forget to check the Binary option, as the file to send is a binary image.

Fig 6. Send file selection: u-boot.bin

After download, the following message will appear:

```
178420 bytes are transfered


U-Boot 2009.11 (Mar 25 2010 - 18:02:15)

DRAM:   64 MB
In:     serial
Out:    serial
Err:    serial
Net:    dm9000
Hit any key to stop autoboot:  0
EA3131-NXP #
```

5.1.2 SD/MMC card

Set the EA LPC3131 board in SD/MMC boot mode by setting jumpers – BOOT0 (L), BOOT1 (H) and BOOT2 (H). Load *u-boot.bin* file to SD/MMC card as described in section "4.1.2.1 Formatting the card" (replace *apex.bin* with *u-boot.bin* file in all the steps listed). Configure the serial port on EA3131 board as described in section 4.1.1.1. Configure the terminal window on PC as described in section 4.1.1.2. Insert SD/MMC card into the slot on EA LPC3131 board. Now power-up the board, the u-boot boot screen should appear on the terminal window.

5.1.3 On board SPI-NOR flash

Set the EA LPC3131 board in SPI boot mode by setting jumpers – BOOT0(L), BOOT1(L) and BOOT2(H). Program "*u-boot.bin*" into on-board SPI-NOR flash. Configure the serial port on EA3131 board as described in section 4.1.1.1. Configure the terminal window on PC as described in section 4.1.1.2. Now power-up the board, the Apex boot screen should appear on the terminal window.

5.1.3.1 Programming SPI-NOR flash using u-boot

You can use *u-boot.bin* itself to program SPI-NOR flash device. For this you need to boot *u-boot* using UART or USB-DFU or other boot modes.

- Once you are at *u-boot* prompt you can load the image into SDRAM using one of following methods:
  - Load *u-boot.bin* from SD/MMC card. See section "5.2.1.1 Loading images from SD/MMC card using U-boot" for detailed steps.
  - Load *u-boot.bin* using UART (Y-MODEM) interface. See section "5.2.4.1 U-boot file transfer using Y-modem protocol over UART" for detailed steps.
  - Load *u-boot.bin* using TFTP file transfer. See section "5.2.3.1 U-boot file transfer using TFTP protocol" for detailed steps.
  - Load *u-boot.bin* using USB-DFU file transfer (EA313x board is USB gadget mode). See section "5.2.5.2 U-boot file transfer using USB-DFU class" for detailed steps.
  - Load *u-boot.bin* using USB flash drives (EA313x board is in USB host mode). See section "5.2.6.2 Loading images from USB flash drives using U-boot" for detailed steps.
- Once the target image is downloaded into SDRAM. Do the following to program SPI-NOR flash with the image data in SDRAM.

```
EA3131-NXP #
EA3131-NXP # sf probe 0 0 0
16384 KiB AT45DB321D at 0:0 is now current device
EA3131-NXP # usb start
(Re)start USB...
USB:   LPC31xx init hccr 19000100 and hcor 19000140 hc_length 64
Register 10011 NbrPorts 1
USB EHCI 1.00
scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
EA3131-NXP # fatload usb 0 31000000 u-boot.bin
reading u-boot.bin
.................

194840 bytes read
EA3131-NXP # sf erase 0 2fa00
EA3131-NXP # sf write 31000000 0 2fa00
EA3131-NXP #
```

- It is recommended that the *length* parameter in "*sf write*" and "*sf erase*" command is rounded-up to the nearest 512 byte boundary for proper write operation. Note, in the above command sequence the rounded value is 0x2FA00.
- Now change the boot jumpers for SPI-NOR boot and reset the board.
- Similarly one can program kernel image into SPI flash and boot from it. On EA board 4Mbytes of SPI-NOR flash is used and programming both kernel and Ramdisk in SPi-flash is not practical. Smaller kernel and ramdisk images cand programmed in SPI-flash. Check "spi_boot" and "spiram_boot" script in ea31xx.h config file for kernel boot commands.

5.1.4 On board NAND flash

LPC313x patch adds NAND based block device support to U-boot.

5.1.4.1 LPC313x NAND Support in U-boot

LPC313x has NAND controller with HW ECC, while it improves read/write performance it complicates the bad block management in u-boot and kernel. The NAND controller on LPC313x/4x/5x when interfacing with large block NAND device it writes the 2048+64 page data into four 512+16 byte format. Due to the way it writes data to NAND page the standard algorithm (check specific offset such as 2048 contains non 0xFF data value) which detect factory bad blocks could falsely detect a block as bad which was written with proper data using HW ECC engine. To overcome this issue the factory bad block scan should only be done once per NAND device and stored in flash. Hence the u-boot patch does this scan on boot up and writes the NAND params and bad blocks list in block 0 (required by Boot ROM). But it will skip the scan if the params and bad block list are already written in block0. But the MTD sub-system in u-boot & kernel maintain bad block list (in MTD format) in some other blocks other than block 0 (required by bootROM). Due to this there are 2 bad block tables in flash. To aid in synchronization and maintenance of these two independent tables, three new commands "*nand_format*", "*nand_params*" and "*nand_bad_block0_dump*" are added to u-boot.

Bad Block Tables

LPC313x *u-boot* patch defines two types of bad block tables (BBTs) to satisfy MTD and boot ROM reuirements.

- **Block 0 based BBT**
  - To boot a device from NAND, the boot ROM code requires a Factory BBT map starting from Page 1 of Block 0.
  - This BBT is constructed by u-boot if the block 0 is nocontains Factory marked bad blocks and last four blocks which is used by MTD. It is typically only used by the boot ROM during boot. In u-boot, this table also provides an initial factory bad block record of the device before the factory bad block markers are overwritten. The map can be used to restore the NAND to the factory default condition.

- **MTD sub-system based BBT**
  - Last four blocks of NAND contains the MTD sub-system based BBT map which contains Factory BBT as well as blocks which become bad over the lifetime of the device. Initially, Block 0 based BBT is same as MTD based BBT, but over time, it will be different than MTD BBT.

New commands

Command 'nand_params'

This command will write NAND boot parameters in Page 0 and bad block list (based on factory markers) in Page 1 of block 0 respectively which requires by boot ROM code. See boot ROM chapter in LPC313x user manual for more details on the NAND boot parameters and bad block list requirements. At the time of very first boot, u-boot code will write this data into block 0. Then after, u-boot code never writes Block 0. To sync Block 0 BBT map with MTD BBT map, one has to manually write this command on u-boot command prompt. Use with care as this can alter the

original factory bad block marker history. Note, last four block of NAND device are also listed as bad in the Block 0 BBT since these blocks are used by MTD sub-system to store its own BBT.

Command 'nand_format'

This command supports two type of format methods:

- **"easy"** : This method will erase all the blocks of NAND device skipping block 0 and bad blocks present in the bad block map in Block 0. It does not erase the MTD sub-system based bad block map as those blocks are marked bad by MTD sub-system. To erase MTD based bad block map, use "nand_format hard" command.
- **"hard" :** This method will erase all the blocks of NAND device including block 0. To reconstruct bad blocks list it will perform erase/write/read/verify operation on each block and if verification fails, logs the block as bad. It also erase the MTD sub-system based bad block map which is stored in last four blocks of NAND. This command will take very long time to execute. At the end both Block 0 BBT and MTD BBT are updated with the newly detect bad blocks.

Command 'nand_bad_block0_dump'

This command displays bad blocks listed in bad block map stored into Block 0.

5.1.4.2 Programming NAND flash using u-boot

You can use *u-boot.bin* itself to program NAND flash device. For this you need to boot *u-boot* using UART or USB-DFU or other boot modes.

- Once you are at *u-boot* prompt you can load the image into SDRAM using one of following methods:
  - Load *u-boot.bin* from SD/MMC card. See section "5.2.1.1 Loading images from SD/MMC card using U-boot" for detailed steps.
  - Load *u-boot.bin* using UART (Y-MODEM) interface. See section "5.2.4.1 U-boot file transfer using Y-modem protocol over UART" for detailed steps.
  - Load *u-boot.bin* using TFTP file transfer. See section "5.2.3.1 U-boot file transfer using TFTP protocol" for detailed steps.
  - Load *u-boot.bin* using USB-DFU file transfer (EA313x board is USB gadget mode). See section "5.2.5.2 U-boot file transfer using USB-DFU class" for detailed steps.
  - Load *u-boot.bin* using USB flash drives (EA313x board is in USB host mode). See section "5.2.6.2 Loading images from USB flash drives using U-boot" for detailed steps.
- Once the target image is downloaded into SDRAM. Do the following to program NAND flash with the image data in SDRAM.

```
EA3131-NXP #
EA3131-NXP # usb start
(Re)start USB...
USB:   LPC31xx init hccr 19000100 and hcor 19000140 hc_length 64
Register 10011 NbrPorts 1
USB EHCI 1.00
scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
EA3131-NXP # fatload usb 0 31000000 u-boot.bin
```

```
reading u-boot.bin
...................

209408 bytes read
EA3131-NXP # nand write 31000000 20000 33800

NAND write: device 0 offset 0x20000, size 0x33800
 210944 bytes written: OK
EA3131-NXP #
EA3131-NXP #
```

- The *length* parameter in "*nand write"* command should be rounded-up to the nearest 2048 byte boundary (NAND device's page size used on EA board) for proper write operation. Note, in the above command sequence the rounded value is 0x33800.
- Now change the boot jumpers for NAND boot and reset the board.
- Note, if you haven't soldered the NAND fix board to the EA board then remove "ABUF_EN" and "DBUF_EN" jumpers for NAND boot to work.
- Similarly one can program kernel image into NAND flash and boot from it.

5.1.5 USB DFU

LPC313x/4x/5x boot ROM supports booting images via Device Firmware Upgrade (DFU) protocol over USB interface. For security reasons boot ROM expects the DFU download image to be TEA encrypted. See LPC313x/4x/5x user manual for more details. For DFU boot do the following:

- Create the DFU boot image as illustrated in section "3.3.2 Creating DFU bootable u-boot image".
- Now set the EA LPC3131 board in USB-DFU boot mode by setting jumpers – BOOT0 (L), BOOT1 (H) and BOOT2 (L).
- Power the board
- Connect USB cable between USB mini-B connector on board and the Linux PC on which "dfu-util" package is installed. See section "5.2.5.1 Installing DFU-UTIL package on host" for installation instructions.
- On Linux host PC issue the following commands.

```
nxp@nxp-demo:~/projects/lpc315x$ dfu-util -l
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Found Runtime: [0x0471:0xdf55] devnum=0, cfg=0, intf=0, alt=0,
name="UNDEFINED"
pdurgesh@pd-ubuntu:~/projects/lpc315x$ sudo dfu-util -R -t 2048 -D
test.rom
[sudo] password for nxp:
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Opening USB Device 0x0000:0x0000...
Claiming USB DFU Runtime Interface...
Determining device status: state = dfuIDLE, status = 0
WARNING: Runtime device already in DFU state ?!?
Found Runtime: [0x0471:0xdf55] devnum=38, cfg=0, intf=0, alt=0,
name="UNDEFINED"
Claiming USB DFU Interface...
Setting Alternate Setting ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
```

```
Transfer Size = 0x0800
bytes_per_hash=1597
Starting download: [#################################################]
finished!
state(8) = dfuMANIFEST-WAIT-RESET, status(0) = No error condition is
present
Done!
can't detach: error sending control message: Broken pipe
Resetting USB to switch back to runtime mode
nxp@nxp-demo:~/projects/lpc315x$ dfu-util -l
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Found Runtime: [0x1457:0x5119] devnum=0, cfg=0, intf=2, alt=0,
name="UNDEFINED"
pdurgesh@pd-ubuntu:~/projects/lpc315x$ dfu-util -D u-boot.bin
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Opening USB Device 0x0000:0x0000...
Claiming USB DFU Runtime Interface...
Cannot claim interface: could not claim interface 2: Operation not
permitted
nxp@nxp-demo:~/projects/lpc315x$ sudo dfu-util -D u-boot.bin
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Opening USB Device 0x0000:0x0000...
Claiming USB DFU Runtime Interface...
Determining device status: state = appIDLE, status = 0
Device really in Runtime Mode, send DFU detach request...
Resetting USB...
Opening USB Device...
Found Runtime: [0x1457:0x5119] devnum=40, cfg=0, intf=0, alt=0,
name="RAM memory"
Claiming USB DFU Interface...
Setting Alternate Setting ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
Transfer Size = 0x1000
bytes_per_hash=3720
Starting download: [#################################################]
finished!
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
nxp@nxp-demo:~/projects/lpc315x$
```

- Note, in above command sequence illustration, "*dfu-util –l*" commands are issued just for clarity. These commands could be skipped.
- Don't forget to specify "*-t 2048 –R*" options for initial DFU download. The LPC313x/4x/5x boot ROM expects the DFU payload length to be 2048 bytes and it will not transfer the control to downloaded image until it receives DFU reset command, hence the "*-R*" option.

5.2 Loading kernel & ramdisk using U-boot

5.2.1 SD/MMC card (FAT formatted)

U-boot with LPC313x patch supports loading images from FAT formatted SD/MMC cards.

5.2.1.1 Loading images from SD/MMC card using U-boot

To load images from FAT formatted SD/MMC cards do the following at u-boot prompt:

- Insert card into SD/MMC slot on EA I/O board.
- On terminal window at u-boot prompt type the followings command. The following instructions assume the file name as *u-boot.bin* and load/destination memory address as *0x31000000*.

```
EA3131-NXP #
EA3131-NXP # mmc init
mmc1 is available
EA3131-NXP # fatls mmc 0
        50   test.txt
    178232   u-boot.bin

2 file(s), 0 dir(s)

EA3131-NXP # fatload mmc 0 0x31000000 u-boot.bin
reading u-boot.bin

178232 bytes read
EA3131-NXP #
```

- Now the memory at address 0x31000000 is loaded with *u-boot.bin* file.

5.2.1.2 Configuring U-boot to boot kernel from SD/MMC

Once U-boot boots, to load the kernel and ramdisk from FAT formatted SD card, issue the following command at U-boot prompt. Note, the “sdmmcram_boot” script assumes that the kernel image (uImage) and uboot formatted ramdisk (rootfs.ext2.gz.uboot) files are present in root directory of the card.

```
EA3131-NXP # run sdmmcram_boot
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # run sdmmc_boot
```

5.2.2 On board NAND flash

The LPC313x patch adds NAND based block device support in u-boot.

To program kernel image into nand flash do the steps listed in section “5.1.4.2 Programming NAND flash using u-boot”. Always copy kernel image at **0x80000** offset so that enough block are left for u-boot and its environment.

Similarly program the ramdisk into nand at offset **0x2A0000**. Now issue the following command to boot kernel and ramdisk from NAND flash.

```
EA3131-NXP # run nandram_boot
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # run nand_boot
```

JFFS2 based root file system is not tested at the release of this package. But all the need drivers and tools are made available in this release.

5.2.3 Ethernet using TFTP boot

U-boot when enabled supports TFTP protocol for file transfers over ethernet. The file transfers are achieved using “tftpboot [loadAddress] [[hostIPaddr:]bootfilename]” command.

5.2.3.1 U-boot file transfer using TFTP protocol

Embedded Artist’s LPC313x version 2.0 IO boards have DM9000 Ethernet controller onboard. The board doesn’t have MAC-EEPROM populated by default; hence the MAC address has to be stored in on board NAND flash or SPI-flash. Current implementation of U-boot stores the MAC addresses as part of environment in SPI-NOR flash (as per default config). Hence users could modify the “ethaddr” environment variable to save different MAC address for different boards.

To aid customers in generating unique MAC address per board the u-boot code generates the 6 byte MAC address using the on-chip 32-bit “Random Number Generator” block. The MAC address is computed as “00:08:” + 4byte random number. The above described MAC address generation is done only when u-boot is not able find proper environment in onboard SPI or NAND flash devices. Note, users who want to capture Ethernet traffic for debug using MAC address filters should save environment, so that MAC address is fixed for all subsequent boots.

The following u-boot screen dump provides command sequence to download images using TFTP.

- First set the TFTP “serverip” address from the file will be downloaded.

```
EA3131-NXP #
EA3131-NXP # setenv serverip 192.168.1.48
EA3131-NXP #
```

- Now set the IP address, gateway address and netmask. This can be done using on of the following static method, dhcp method, bootp or rarp method depending on the type of servers available in your sub-network.
  - **Static method**

```
EA3131-NXP #
EA3131-NXP # setenv ipaddr 192.168.1.132
EA3131-NXP # setenv gatewayip 192.168.1.1
EA3131-NXP # setenv netmask 255.255.255.0
EA3131-NXP # setenv dnsip 216.136.95.2
EA3131-NXP #
```

  - **DHCP method:** In this method *u-boot* will try to download the bootfile immediately after getting the DHCP response.

```
EA3131-NXP #
EA3131-NXP # dhcp
dm9000 i/o: 0x20020000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:08:62:92:10:0c
operating at 100M full duplex mode
BOOTP broadcast 1
DHCP client bound to address 192.168.1.132
Using dm9000 device
TFTP from server 192.168.1.48; our IP address is 192.168.1.132
Filename 'uImage'.
Load address: 0x31000000
Loading:
#################################################################
         ##############################################
done
Bytes transferred = 1621548 (18be2c hex)
EA3131-NXP #
```

- **BOOTP method:** In this method *u-boot* will try to download the bootfile immediately after getting the bootp response.

```
EA3131-NXP #
EA3131-NXP # bootp
dm9000 i/o: 0x20020000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:08:62:92:10:0c
operating at 100M full duplex mode
BOOTP broadcast 1
DHCP client bound to address 192.168.1.132
Using dm9000 device
TFTP from server 192.168.1.48; our IP address is 192.168.1.132
Filename 'uImage'.
Load address: 0x31000000
Loading:
#################################################################
         ##############################################
done
Bytes transferred = 1621548 (18be2c hex)
EA3131-NXP #
```

- **RARP method:** In this method *u-boot* will try to download the bootfile immediately after getting the RARP response.

```
EA3131-NXP #
EA3131-NXP # setenv ethaddr 00:08:ee:00:80:44
EA3131-NXP # rarpboot
dm9000 i/o: 0x20020000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:08:ee:00:80:44
operating at 100M full duplex mode
RARP broadcast 1
Using dm9000 device
TFTP from server 192.168.1.48; our IP address is 192.168.1.78
Filename 'uImage'.
Load address: 0x31000000
Loading:
#################################################################
         ##############################################
done
Bytes transferred = 1621548 (18be2c hex)
EA3131-NXP #
```

Configuring TFTP, BOOTP and RARP server on Linux PC is outside the scope of this document.

- Once the IP address configuration is done. Files can be downloaded from the TFTP server using the following commands.

```
EA3131-NXP #
EA3131-NXP # tftpboot 0x31000000 /tftpboot/uImage
dm9000 i/o: 0x20020000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:08:b1:5e:c2:7a
operating at 100M full duplex mode
Using dm9000 device
TFTP from server 192.168.1.48; our IP address is 192.168.1.133
Filename '/tftpboot/uImage'.
Load address: 0x31000000
Loading:
#################################################################
```

```
         ################################################
done
Bytes transferred = 1621548 (18be2c hex)
EA3131-NXP #
```

5.2.3.2 Configuring U-boot to boot kernel using TFTPboot

Once U-boot boots, to load the kernel and ramdisk from network using TFTP protocol, issue the following command at U-boot prompt. Note, the “netram_boot” script assumes that the kernel image (uImage) and uboot formatted ramdisk (rootfs.ext2.gz.uboot) files are in TFTP root directory on the server. If they are in sub-directories of TFTP root directory update “bootfile” and “ramfile” environment variables using “setenv” command in u-boot environment.

```
EA3131-NXP # run netram_boot
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # run sdmmc_boot
```

5.2.4 UART using Y-modem protocol

U-boot when enabled supports YMODEM protocol for serial file transfers. The file transfers are achieved using “loady [off] [baud]” command.

At the time of writing this document TeraTermPro application doesn’t support Y-modem protocol. As per TeraTerm project 4.66 release should support YMODEM. An unofficial release with YMODEM support is available at http://ttssh2.sourceforge.jp/snapshot/snapshot-20100325.zip location. Users could use this application or use any other terminal application (such as MS Hyperterm) which supports YMODEM protocol to proceed further.

5.2.4.1 U-boot file transfer using Y-modem protocol over UART

- To do Y-modem file transfer to board do the following at *u-boot* prompt:

```
EA3131-NXP #
EA3131-NXP # loady 0x31000000
## Ready for binary (ymodem) download to 0x31000000 at 115200 bps...
C
```

- Now on PC select Y-modem file transfer option. Following is a screen-shot showing the menu option in TeraTerm application.

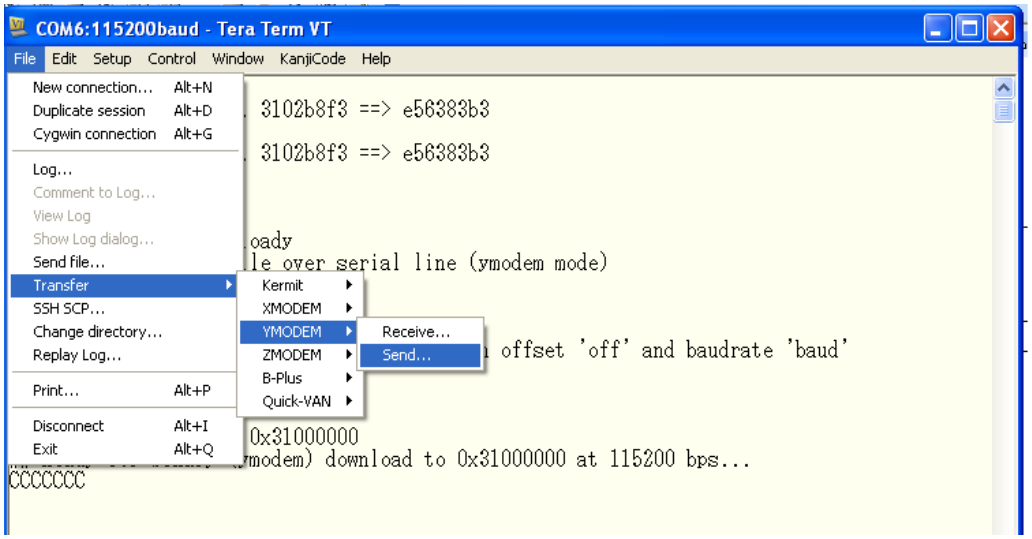


Fig 7. Y-modem transfer menu

a. Now select the file in “file selection” dialog box. After file is selected the file transfer will start and following progress dialog should appear.

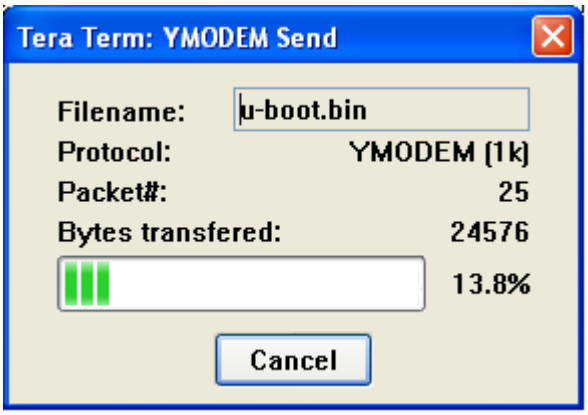


Fig 8. Y-modem file transfer progress dialog

- The following screen shows the terminal output after file transfer.

```
EA3131-NXP # loady 31000000
## Ready for binary (ymodem) download to 0x31000000 at 115200 bps...
CCxyzModem - CRC mode, 0(SOH)/176(STX)/0(CAN) packets, 4 retries
## Total Size      = 0x0002b8f4 = 178420 Bytes
EA3131-NXP #
```

- After both files are downloaded enter ‘boot’ command to boot linux.

5.2.4.2 Configuring U-boot to boot kernel using Y-modem protocol over UART

Once U-boot boots, to load the kernel and ramdisk from host PC using Y-modem protocol over UART. Issue the following command at U-boot prompt. Note, the following command sequence assumes that the kernel image (uImage) is downloaded first and then uboot formatted ramdisk (rootfs.ext2.gz.uboot) file is downloaded next.

```
EA3131-NXP # loady $(loadaddr)
EA3131-NXP # loady $(rd_addr)
EA3131-NXP # run ramargs
EA3131-NXP # bootm $(loadaddr) $(rd_addr)
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # loady $(loadaddr)
EA3131-NXP # run nfsargs
EA3131-NXP # bootm $(loadaddr)
```

5.2.5 USB DFU class

The LPC313x patch includes Opemoko's USB gadget DFU class support for u-boot. USB Device Firmware Upgrade (DFU) is an official USB device class specification of the USB Implementers Forum. It specifies a vendor and device independent way of updating the firmware of a USB device. The idea is to have only one vendor-independent firmware update tool as part of the operating system, which can then (given a particular firmware image) be downloaded into the device.

The LPC313x U-boot provides USB-DFU mechanism to download kernel and ramdisk images to the EA3131/EA3152 boards. Customer boards which don't have Ethernet controller on-board could use this method for faster download during development.

5.2.5.1 Installing DFU-UTIL package on host

Openmoko project has developed "dfu-util" package which can used to transfer files to devices supporting DFU class specification. To install the dfu-util package on your host distributions follow the instructions provided on http://wiki.openmoko.org/wiki/Dfu-util site. At the time of writing this document most Linux distribution have "dfu-util" package available as an optional install package.

5.2.5.2 U-boot file transfer using USB-DFU class

- To do USB-DFU file transfer to board do the following at *u-boot* prompt:

```
EA3131-NXP # usbpoll 31000000
```

- Now connect the USB cable between mini-B connector on EA3131 board and Linux host PC which has "dfu-util" package installed.
- Once the cable is connected on PC issue the following command to transfer the file. The following command assumes that "uImage" file is in current working directory.

```
nxp@nxp-demo:~/projects/lpc315x$ sudo dfu-util -D uImage
```

- After "dfu-util" command is issued on PC the file transfer should start. Following are the screen shots of board and PC.
  - On EA3131 board (terminal screen)

```
EA3131-NXP # usbpoll 31000000
USB get RESET interrupt
USB get RESET interrupt
USB get RESET interrupt
set_addr24
USB get RESET interrupt
DFU: Switching to DFU Mode
USB get RESET interrupt
set_addr24
```

```
USB get suspend interrupt
USB get RESET interrupt
USB get RESET interrupt
set_addr25
new_state = 2
Starting DFU DOWNLOAD to RAM (0x31000000)
bytes transfered: 1621548
EA3131-NXP #
```

- On Linux host PC

```
nxp@nxp-demo:~/projects/lpc315x$ sudo dfu-util -D uImage
dfu-util - (C) 2007-2008 by OpenMoko Inc.
This program is Free Software and has ABSOLUTELY NO WARRANTY

Opening USB Device 0x0000:0x0000...
Found Runtime: [0x1457:0x5119] devnum=34, cfg=0, intf=0, alt=0,
name="RAM memory"
Claiming USB DFU Interface...
Setting Alternate Setting ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
Transfer Size = 0x1000
bytes_per_hash=32430
Starting download: [##################################################]
finished!
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

5.2.5.3 Configuring U-boot to boot kernel using USB-DFU class

Once U-boot boots, to load the kernel and ramdisk from host PC using USB-DFU class (EA313x in USB gadget mode). Issue the following command at U-boot prompt. Note, the “usbdfuram_boot” script assumes that the kernel image (uImage) is downloaded first and then uboot formatted ramdisk (rootfs.ext2.gz.uboot) file is downloaded next.

```
EA3131-NXP # run usbdfuram_boot
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # run usbdfu_boot
```

5.2.6 USB flash drives

The LPC313x patch includes support for USB EHCI host controller present on LPC313x/4x/5x SoCs. As part of USB host support u-boot has mass storage class support to read USB flash drives (a.k.a USB pen drives).

Customer boards which have USB type-A connector on-board could use this method for faster download method during development.

5.2.6.1 Enabling USB EHCI and Mass-storage class support in U-boot

To enable USB EHCI host support in u-boot, following CONFIG defines should be defined in work_2009.11\include\configs\ ea31xx.h file.

- As shown in the following snippet of code from ea31xx.h file,

```
/* Uncomment if you want USB host support and
 * disable USB gadget supoprt. Please note if
```

```
 * you disable USB gadget support, USB DFU boot
 * mode will not work.
 */
/*
 * USB configuration as EHCI HOST
 */
#define CONFIG_CMD_USB
#define CONFIG_USB_EHCI              /* Enable EHCI USB support */
#define CONFIG_USB_EHCI_LPC313X           /* on LPC313X platform */
#define CONFIG_EHCI_IS_TDI
#define CONFIG_USB_STORAGE
#define CONFIG_SUPPORT_VFAT
```

- After changing ea31xx.h file always do 'make clean' before re-building the u-boot image.

5.2.6.2 Loading images from USB flash drives using U-boot

To load images from FAT formatted USB flash drives do the following at u-boot prompt:

- Insert USB flash drive into USB type-A connector on EA I/O board.
- On terminal window at u-boot prompt type the followings command. The following instructions assume the file name as *uimage* and load/destination memory address as *0x31000000*.

```
EA3131-NXP #
EA3131-NXP # usb start
(Re)start USB...
USB:   LPC31xx init hccr 19000100 and hcor 19000140 hc_length 64
Register 10011 NbrPorts 1
USB EHCI 1.00
scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
EA3131-NXP # fatls usb 0
  1465248   uimage

1 file(s), 0 dir(s)

EA3131-NXP # fatload usb 0 31000000 uimage
reading uimage
.................................................................................
.................................................................

1465248 bytes read
EA3131-NXP #
```

- Now the memory at address 0x31000000 is loaded with *uimage* file.

5.2.6.3 Configuring U-boot to boot kernel from USB flash drives

Once U-boot boots, to load the kernel and ramdisk from FAT formatted USB flash drives, issue the following command at U-boot prompt. Note, the following command sequence assumes that the kernel image (uImage) and uboot formatted ramdisk (rootfs.ext2.gz.uboot) files are present in root directory of the card. If they are in sub-directories update "bootfile" and "ramfile" environment variables using "setenv" command in u-boot environment.

```
EA3131-NXP # usb start
EA3131-NXP # fatload usb 0 $(loadaddr) $(bootfile)
```

```
EA3131-NXP # fatload usb 0 $(rd_addr) $(ramfile)
EA3131-NXP # run ramargs
EA3131-NXP # bootm $(loadaddr) $(rd_addr)
```

To boot kernel with NFS based root file system issue the following command at U-boot prompt.

```
EA3131-NXP # usb start
EA3131-NXP # fatload usb 0 $(loadaddr) $(bootfile)
EA3131-NXP # run nfsargs
EA3131-NXP # bootm $(loadaddr)
```

4. Porting Apex to LPC313x custom board

To port Apex to custom boards (using LPC313x/4x/5x chips) the main rule of thumb is to change & implement code specific to the new board in all places where you find the machine specific macros. In Apex code search for CONFIG_MACH_VAL3153 or CONFIG_MACH_EA313x_V1 defines. Below is the list containing filenames and list of changes to be done in those files.

1. Create your own board specific CONFIG_MACH_xxx define in work_1.6.8\src\mach-lpc313x\kconfig file.
2. In file work_1.6.8\src\mach-lpc313x\initialize.c:
   - Modify initialize_bootstrap (void) function with proper SDRAM and static memory (memory mapped peripherals) timings. Use the CONFIG_MACH_xxx define added in Kconfig to add your board specific code.
3. File work_1.6.8\src\mach-lpc313x\env.c:
   - Modify the default boot parameters for your board. ie., change .default_value. Or create your own config file and manipulate the boot parameter in config file.

7. Porting Linux 2.6.x to LPC313x custom board

To port 2.6.x Linux kernel to custom boards (using LPC313x/4x/5x chips) the main rule of thumb is to change & implement code specific to the new board in all places where you find the machine specific macros. In LPC313x Linux patch code search for CONFIG_MACH_VAL3153 or CONFIG_MACH_EA313X defines. Below is the list containing filenames and list of changes to be done in those files.

1. Create your own board specific define in work_2.6.x.x\arch\arm\mach-lpc313x\kconfig file.
2. Copy work_2.6.x.x\arch\arm\mach-lpc313x\ea313x.c to work_2.6.x.x\arch\arm\mach-lpc313x\board-mybrd.c file. Modify all structures and function according to your board.
3. File work_2.6.28.2\arch\arm\mach-lpc313x\include\mach\irqs.h:
   - Add board specific BOARD_IRQ_EVENT_MAP.
4. File work_2.6.28.2\arch\arm\mach-lpc313x\Makefile:
   - Include board file in the build
   - obj-$(CONFIG_MACH_MYBRD) += board-mybrd.o leds.o
5. If LEDs are connected to GPIO0/1/2 on your board then you could re-use leds.c or else create new file according to your board if LED functionality is desired. OR else remove LED feature in your kernel config.
6. Create your own custom kernel config file for your board. You could start with ea313x_defconfig file and remove/add kernel feature per you board and create mybrd_defconfig file by copying the work_2.6.x.x\.config file to work_2.6.x.x\ arch\arm\configs\mybrd_defconfig.

8. Build differences between LTIB and ELDK

LTIB and ELDK take different approaches to providing a complete Linux system. The ELDK approach provides pre-built binary packages from the root filesystem image. LTIB builds all necessary packages and the root filesystem from the configuration options selected by the user.

For the current version of the ELDK and LTIB releases, the following differences apply.

- ELDK and LTIB use similar, but different toolchains
- The generated LTIB ramdisk image is bigger than the pre-packaged ELDK ramdisk image
- The Apex default configuration has a change to the Linux kernel command line to allow for increased ramdisk size
    - If packages are added to the system via the LTIB package menu, this ramdisk size may need to be increased. LTIB will give a recommended value to use for the new ramdisk size.
    - The ramdisk size can be edited in the Apex menu using the “Configure Apex” option and then the environment menu in Apex.