Automatic Generation of Linux 2.6 Board Support Packages

Summary

This document describes the automatic generation of a Linux 2.6 Board Support Package (BSP) through Xilinx Platform Studio (XPS). The document contains the following sections.

Overview

In a typical embedded development environment, one of the tasks is to create software to support the custom hardware on the embedded system for the target O/S. This software that supports embedded custom hardware is often called a Board Support Package (BSP). In an environment where hardware is defined in a programmable System-on-Chip (SoC), hardware changes can come about much more rapidly, making it difficult for the BSP to remain current with the revisions in hardware.

To ease this situation, Xilinx provides a process called Automatic BSP Generation that tailors a BSP according to the current hardware configuration of the FPGA.

Automatic generation of a BSP is done using Xilinx Platform Studio (XPS), which is available in the Xilinx Embedded Development Kit (EDK). XPS generates BSPs based on the defined hardware configuration. For Linux, XPS generates a sparse Linux kernel source tree containing just the hardware specific files for the BSP. For Linux 2.6, XPS supports both the MontaVista and Wind River Linux distributions.

In general, the flow of work for using Linux on an embedded system using FPGAs is as follows:

1. Define the hardware components in XPS
2. Select a Linux 2.6 distribution as the target operating system in XPS
3. Specify Operating System Parameters
4. Generate the BSP in XPS
5. Configure the kernel
6. Define the root file system
7. Build the kernel
8. Install the kernel and root file system
9. Develop and run application specific code

This guide will describe steps 2 through 5, and 7. The remaining steps are beyond the scope of this guide.

Getting Started with Linux 2.6

The Linux 2.6 distributions currently supported in XPS are from MontaVista and Wind River. These distributions can be purchased directly from those vendors. The MontaVista product is named Linux Professional Edition 4.0.1, which includes a kernel source tree, development tools, and technical support. The Wind River products are named General Purpose Platform, Linux Edition 1.3 and Linux Edition 2.0. These Linux products provide a PowerPC 4xx cross-development environment that runs on various host operating systems. See the vendor websites for a list of supported host operating systems.

To get started, first install the MontaVista or Wind River Linux distribution CDs for the PowerPC 4xx. Once the main distribution is installed, each vendor provides a Xilinx BSP CD or download image that can be installed on top of the main distribution. Please follow the vendor-specific installation instructions.

MontaVista uses the term LSP instead of BSP. LSP stands for Linux Support Package, but should be considered analogous to Board Support Package. The Linux 2.6 BSPs provided are for specific reference designs for the Xilinx ML403 and ML300 development boards. The reference designs can be found on the Xilinx website at www.xilinx.com/ml403 and www.xilinx.com/ml300 (ML300 Lounge) respectively. When developing a custom hardware design for these boards or for other boards, the user should use the automatically generated BSP from Platform Studio in conjunction with one of the aforementioned BSPs.

When building an EDK design on a custom board, the target FPGA needs to have the PowerPC processor. Either a serial port or some device that can be used as a console device would be very useful. Also, unless a ramdisk is going to be used, some device that will be used to access the root file system needs to be considered (e.g., Ethernet for an NFS root filesystem or System ACE for a CompactFlash root filesystem).

Creating a Working Kernel Tree

It is recommended to create a working copy of the Linux kernel tree that is installed with your MontaVista or Wind River distribution. This ensures that the installed copy is kept pristine.

MontaVista Linux

When installed on a Linux system, the default install directory for the MontaVista Linux kernel source, for MontaVista Linux Professional Edition, is here:

```
/opt/montavista/pro/devkit/lsp/<target board>/linux-2.6.10_mvl401
```

Some care must be taken to correctly copy the kernel source tree to preserve links and other file attributes. One method is to use *tar* to make a tarball of the source tree and then extract it to the target location. This tar method can even be done using a transitory temporary tar file by piping the output of the tar creation process into the tar extraction process like so:

```
tar cf - <source_dir> | tar xvf - -C <target_dir>
```

Here is a specific example:

```
tar cf - /opt/montavista/pro/devkit/lsp/xilinx-ml40x-ppc_405/linux-2.6.10_mvl401 | tar xvf - -C
my_linux-2.6.10
```

On Windows, the copy should be performed within a cygwin bash shell so that the Linux file attributes can be preserved, as the Linux kernel build depends on certain soft links to be present.

Wind River Linux

The steps for creating a working Linux kernel tree for Wind River Linux are different than those for MontaVista Linux. Refer to the Getting Started guide in your Wind River Linux distribution for details on creating a kernel and filesystem using the pre-built RPM method or the source build method, or using the Workbench IDE. The following steps describe creating a working kernel from the command line using the source build method. Note that these steps avoid building a filesystem.

- Create a working directory where you want the kernel tree to reside

- From the working directory, run the configure script to copy and configure a kernel tree for the specific BSP you're targeting. For example, for the ML403 BSP:

  <WINDRIVER_INSTALL_DIR>/wrlinux-1.3/wrlinux/configure --enable-kernel=cgl --enable-board=xilinx_ml403

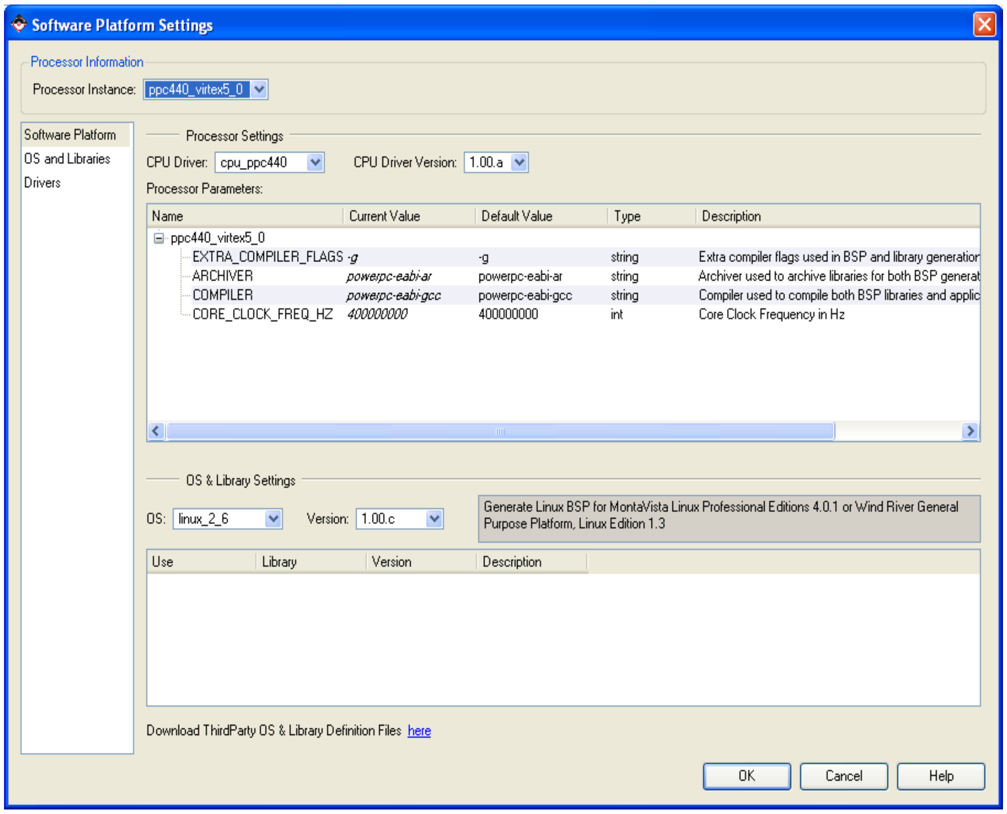
- Type "make -C dist linux.rebuild" to build the working kernel tree, which will reside in dist/linux-2.6.14-cgl under the working directory (for Wind River GPP LE 1.3).

How to Create a BSP from XPS

Before XPS can correctly generate the BSP, XPS needs to know which operating system will be used. XPS will also need to know a few things about the target board that are outside of what is defined by the hardware configuration in the FPGA.

Selecting the Target Operating System

Once the hardware components have been defined and configured in XPS, the target Operating System must be selected. Select the *Software Platform Settings* item from the *Software* menu in order to open the *Software Platform Settings* dialog box . See Figure 1, page 4.



Processor Instance: ppc440_virtex5_0

CPU Driver: cpu_ppc440 CPU Driver Version: 1.00.a

Processor Parameters:

Name	Current Value	Default Value	Type	Description
ppc440_virtex5_0				
EXTRA_COMPILER_FLAGS	-g	-g	string	Extra compiler flags used in BSP and library generation
ARCHIVER	powerpc-eabi-ar	powerpc-eabi-ar	string	Archiver used to archive libraries for both BSP generat
COMPILER	powerpc-eabi-gcc	powerpc-eabi-gcc	string	Compiler used to compile both BSP libraries and applic
CORE_CLOCK_FREQ_HZ	400000000	400000000	int	Core Clock Frequency in Hz

OS: linux_2_6 Version: 1.00.c

Generate Linux BSP for MontaVista Linux Professional Editions 4.0.1 or Wind River General Purpose Platform, Linux Edition 1.3

Use	Library	Version	Description

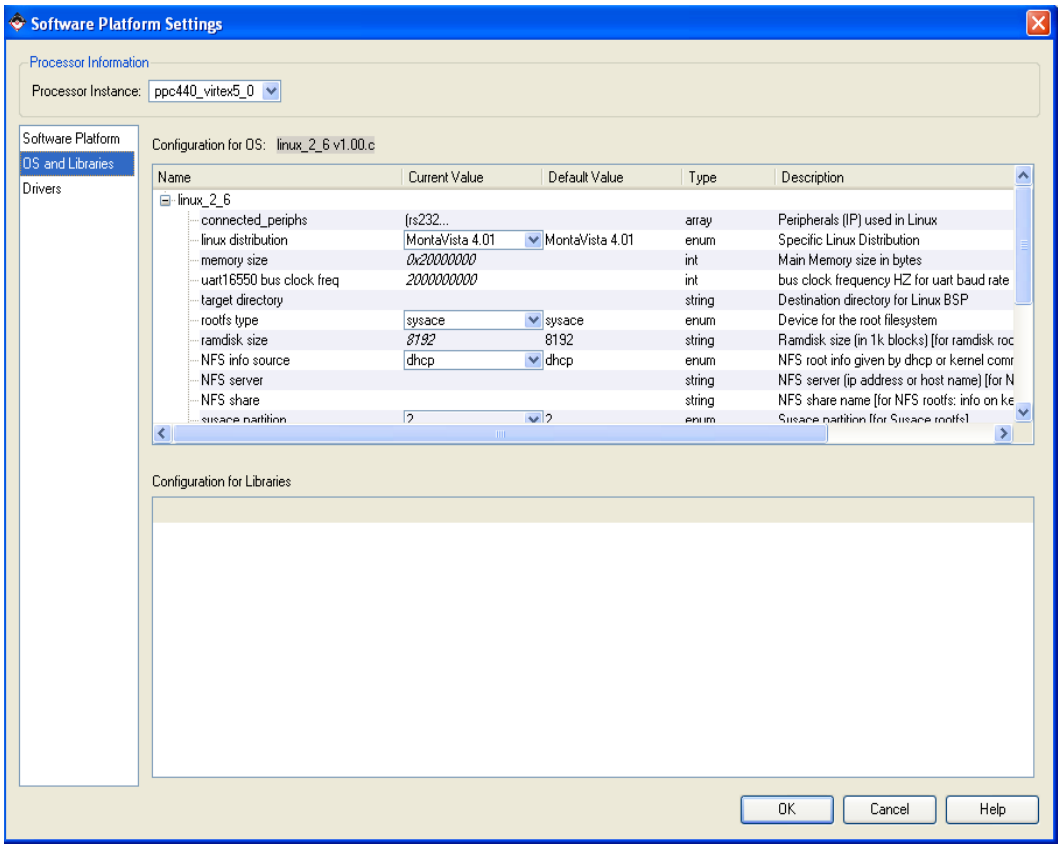
Download ThirdParty OS & Library Definition Files here

Figure 1: **Selecting Target Operating System**

Select *linux_2_6* from the drop down list in the *OS & Library Settings* section of the dialog box.

Setting Operating System Parameters

Now that a Linux 2.6 distribution has been selected, there are some configuration options available from within the *OS & Libraries* pane of the *Software Platform Settings* dialog box as shown in Figure 2, page 5.


Software Platform Settings

Processor Information

Processor Instance: ppc440_virtex5_0

Software Platform
OS and Libraries
Drivers

Configuration for OS: linux_2_6 v1.00.c

Name	Current Value	Default Value	Type	Description
linux_2_6				
connected_periphs	(rs232...		array	Peripherals (IP) used in Linux
linux distribution	MontaVista 4.01	MontaVista 4.01	enum	Specific Linux Distribution
memory size	0x20000000		int	Main Memory size in bytes
uart16550 bus clock freq	2000000000		int	bus clock frequency HZ for uart baud rate
target directory			string	Destination directory for Linux BSP
rootfs type	sysace	sysace	enum	Device for the root filesystem
ramdisk size	8192	8192	string	Ramdisk size (in 1k blocks) [for ramdisk roc
NFS info source	dhcp	dhcp	enum	NFS root info given by dhcp or kernel comr
NFS server			string	NFS server (ip address or host name) [for N
NFS share			string	NFS share name [for NFS rootfs: info on ke
sysace partition	2	2	enum	Sysace partition [for Sysace rootfs]

Configuration for Libraries

Figure 2: **Setting Library/OS Parameters**

The options, some of that are required and some that are optional, are described below.

`connected_periphs` (required)

This parameter specifies which hardware devices are to be supported in Linux through the generated BSP. See Table 1, page 24. Clicking in the Current Value column will bring up a dialog box in which you can specify which peripherals are to be *connected* to the OS.

`linux distribution` (required)

This parameter specifies which distribution you are using, either Wind River Linux or MontaVista Linux. The default is MontaVista Linux. Your Linux kernel may not build properly if this parameter is set incorrectly. This parameter determines the contents of a Kconfig file.

`memory size` (required)

This setting simply lets the OS know how much general-purpose RAM it can use in the system. Obviously this value should be less than or equal to the amount of physical general-purpose RAM available in the system. Though, it can be set to a value less than the amount of physical general-purpose RAM if simulating a smaller amount of RAM, or if some area of RAM is reserved for other purposes.

Note that the hardware configuration should be set so that memory starts at address 0x0.

`UART16550 bus clock freq` (optional)

This parameter specifies the frequency of the bus (in HZ) to which the console serial device is attached. The Linux kernel uses this value to program a 16550/16450 UART baud rate. Note that this setting is only required if a PLB or OPB UART 16550 is included in the hardware design.

`target directory` (optional)

The target directory where the BSP is created can be specified. Typically, the value points to a copy of the Linux kernel source tree so that the generated BSP will directly overlay a working kernel tree with the new drivers. For such an overlay to work correctly, *target_dir* should point to the top-most directory in the working kernel tree.

If this target directory is left blank, it defaults to

<project directory>/<processor name>/libsrc/linux_2_6_v1_00_a/linux

This directory will contain a sparse kernel tree with updated device drivers configured to match the hardware design. This directory should then be copied over the user's working Linux kernel tree. Please use forward slashes to delimit directory names.

`rootfs type` (optional)

The *rootfs type* parameter specifies which type of root filesystem will be used for this kernel tree. Note that this is an initial default type and it can always be changed in your kernel .config. The drop-down list in the Current Value column gives the user a selection of *nfs*, *ramdisk*, or *sysace*. The user's selection will be reflected in the default kernel command line. The default value is *sysace*. Note that as of this writing Wind River Linux does not officially support a ramdisk rootfs.

`ramdisk size` (optional)

This parameter specifies the size of the ramdisk if *ramdisk* was chosen for the *rootfs type*. The default value is 8192 1K byte blocks (8 MB). Note that this is an initial default value and it can be changed in your kernel .config.

`NFS info source` (optional)

This parameter specifies how the NFS root filesystem will be retrieved during boot if *nfs* was chosen for the *rootfs type*. The default value is *dhcp*, which means the NFS information will be taken from the DHCP server. The other option is to select *kernel command line* in order to get the NFS information directly from the kernel command string (e.g., nfsroot=).

`NFS server` (optional)

This parameter specifies the name of the NFS server from which the root filesystem will be mounted during boot if *nfs* is chosen for the *rootfs type* and *kernel command line* is chosen for the *NFS info source*.

`NFS share` (optional)

This parameter specifies the name of the NFS share on the NFS server. This parameter is only used if an NFS server name is provided and *nfs* is chosen for the *rootfs type* and *kernel command line* is chosen for the *NFS info source*..

`sysace partition` (optional)

This parameter specifies the CompactFlash card partition which contains the root filesystem. This parameter is only used if *sysace* was chosen for the *rootfs type*. The default value is 2.

`IP Address` (optional)

This parameter can be set to *on*, *off*, or a static IP address. If *on* is selected, DHCP will be used to retrieve the target's IP address during boot. If *off* is selected, the network is disabled during boot. If a static IP address is specified, this IP address is assigned to the primary Ethernet interface during boot. The default value is *on*.

`Additional kernel command line items` (optional)

This parameter can be used to specify additional command line options if not addressed by the options addressed above. For example, options for kgdb configuration or changing the console device could be specified here.

`powerdown` Parameters (optional)

The *powerdown* parameters are placeholders to aid in creating a soft power down feature in your board. The Xilinx ML300 supports a soft power down feature through a GPIO address. Note that this parameter applies only if you have a powerdown feature on the board and it is accessible through a memory-mapped address (e.g., GPIO).

These values are intended to support a power down method where the *powerdown value* is written to the *powerdown baseaddr* to initiate the hardware power down sequence. The *powerdown highaddr* is used to indicate a memory range used to map pages to a set of physical pages.

`IIC` Parameters (optional)

The IIC parameters are based around the hardware on the Xilinx ML300 and ML403 boards. The boards have an EEPROM attached to an I2C bus. In addition, Linux is set up to read the MAC address for the Ethernet driver from an address on this EEPROM. The IIC parameters specify which addresses in the EEPROM are to be used to read this MAC address as well as specify the device ID of the EEPROM on the I2C bus.

If your board does not have an EEPROM on an I2C bus, these parameters can be safely ignored.

The *IIC persistent baseaddr* value specifies the base address in the EEPROM where the MAC address is stored.

The *IIC persistent highaddr* value is not used for booting up. This value is used by other utilities available with the Xilinx boards that write to the EEPROM.

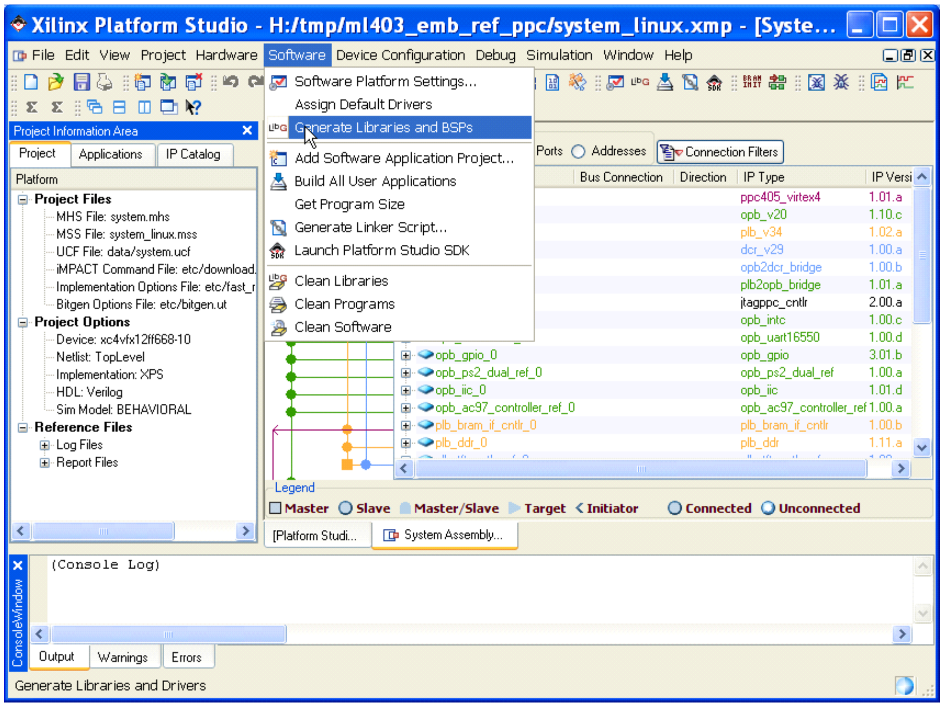
The *IIC persistent eepromaddr* value specifies the I2C bus device ID of the EEPROM.

`PCI Board` (optional)

This parameter specifies the target board for a PCI system. Currently, only the Xilinx ML410 board is supported. When this parameter is set to "ml410", the most common on-board PCI peripherals are enabled in the Linux kernel configuration through the automatic kernel configuration in Platform Studio (see Linux Kernel Configuration section).

Generating the BSP

Selecting *Generate Libraries and BSPs* from the *Software* menu can create the BSP as is seen in Figure 3, page 8.


Xilinx Platform Studio - H:/tmp/ml403_emb_ref_ppc/system_linux.xmp - [Syste...
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Software Platform Settings...
Assign Default Drivers
Generate Libraries and BSPs
Add Software Application Project...
Build All User Applications
Get Program Size
Generate Linker Script...
Launch Platform Studio SDK
Clean Libraries
Clean Programs
Clean Software
Project Information Area
Project
Applications
IP Catalog
Ports
Addresses
Connection Filters
Legend
Master
Slave
Master/Slave
Target
Initiator
Connected
Unconnected
[Platform Studi...
System Assembly...
ConsoleWindow
Output
Warnings
Errors
Generate Libraries and Drivers

Platform
- Project Files
  - MHS File: system.mhs
  - MSS File: system_linux.mss
  - UCF File: data/system.ucf
  - iMPACT Command File: etc/download.
  - Implementation Options File: etc/fast_r
  - Bitgen Options File: etc/bitgen.ut
- Project Options
  - Device: xc4vfx12ff668-10
  - Netlist: TopLevel
  - Implementation: XPS
  - HDL: Verilog
  - Sim Model: BEHAVIORAL
- Reference Files
  - Log Files
  - Report Files

	Bus Connection	Direction	IP Type	IP Versi
			ppc405_virtex4	1.01.a
			opb_v20	1.10.c
			plb_v34	1.02.a
			dcr_v29	1.00.a
			opb2dcr_bridge	1.00.b
			plb2opb_bridge	1.01.a
			jtagppc_cntlr	2.00.a
			opb_intc	1.00.c
			opb_uart16550	1.00.d
opb_gpio_0			opb_gpio	3.01.b
opb_ps2_dual_ref_0			opb_ps2_dual_ref	1.00.a
opb_iic_0			opb_iic	1.01.d
opb_ac97_controller_ref_0			opb_ac97_controller_ref	1.00.a
plb_bram_if_cntlr_0			plb_bram_if_cntlr	1.00.b
plb_ddr_0			plb_ddr	1.11.a

```
(Console Log)
```

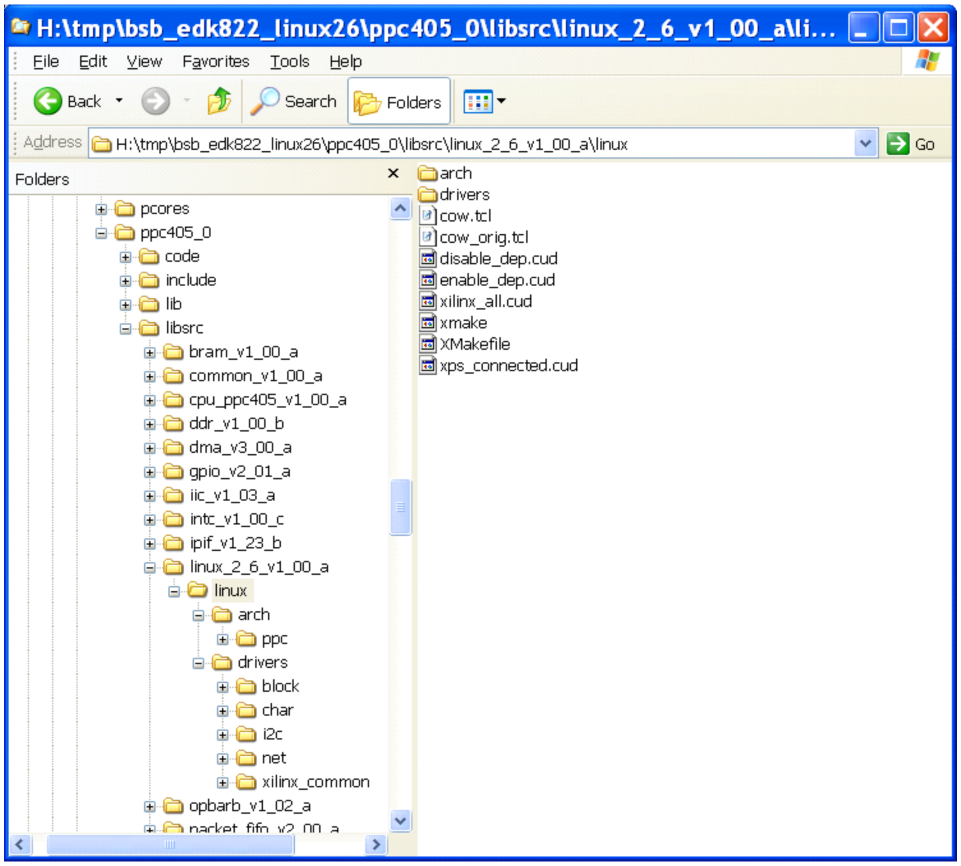
Figure 3: **Generate Libraries and BSPs Menu Item**

Directory Structures

If the target directory is left blank, the target directory defaults to

<project directory>\<processor name>\libsrc\linux_2_6_v1_00_c\linux.

The Linux directory shown in Figure 4, page 9, contains a directory tree of various Linux drivers for the currently configured hardware devices. If the specified target directory does not refer to the same directory as the working Linux kernel source tree, copy this directory into the working Linux kernel source tree.


H:\tmp\bsb_edk822_linux26\ppc405_0\libsrc\linux_2_6_v1_00_a\li...
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
Go
Folders

```
H:\tmp\bsb_edk822_linux26\ppc405_0\libsrc\linux_2_6_v1_00_a\linux

pcores
ppc405_0
    code
    include
    lib
    libsrc
        bram_v1_00_a
        common_v1_00_a
        cpu_ppc405_v1_00_a
        ddr_v1_00_b
        dma_v3_00_a
        gpio_v2_01_a
        iic_v1_03_a
        intc_v1_00_c
        ipif_v1_23_b
        linux_2_6_v1_00_a
            linux
                arch
                    ppc
                drivers
                    block
                    char
                    i2c
                    net
                    xilinx_common
        opbarb_v1_02_a
        packet_fifo_v2_00_a
```

```
arch
drivers
cow.tcl
cow_orig.tcl
disable_dep.cud
enable_dep.cud
xilinx_all.cud
xmake
XMakefile
xps_connected.cud
```

Figure 4: **Example Directory Structure for a Generated Linux BSP**

Copying the BSP to the Linux Kernel Source Tree

If the *target directory* option in the Library O/S Parameters settings refers to your working Linux kernel source tree, the generated BSP does not need to be copied. Otherwise the generated BSP needs to be copied over the top of a working kernel source tree. The top most directories in the generated BSP match some of those in the working Linux kernel directory structure. Copy the generated BSP files so that the files in the /arch directory and /drivers directory go into the /arch directory and /drivers directory in the working kernel source tree respectively.

Care should be taken when copying the BSP using the SMB (Windows Networking) protocol. When using this file sharing protocol, symbolic links to directories on the target can be overwritten as separate directories. The Linux kernel build seems to require the symbolic links to be present rather than having separate directories. In particular, pay attention to the *asm* directories.

Linux Kernel Configuration

The Linux 2.6 BSP generation process through XPS differs from that of Linux 2.4 in that better kernel customization is accomplished through the process. This results in the user needing to do very little, if any, kernel customization other than application-specific customization. There are three methods to configuring your kernel to match the XPS hardware design:

1. *target directory* points to a valid Linux 2.6 kernel tree

   This method is recommended if starting with unmodified Linux kernel source because it automatically updates the kernel .config file to match the Platform Studio hardware design. This method is not recommended if kernel modifications have been made which need to be preserved. Users do not need to manually configure the kernel (e.g., by running “make menuconfig”) to change individual menu options/setting hiding in different menus and sub-menus. Instead, simply generating the BSP from XPS followed by a kernel compilation will generate a kernel that will work on the specific XPS design. (Note this does not prevent users from running traditional “make menuconfig” to change individual kernel settings). A copy of the old .config is saved before updating it with the design-specific configuration. The saved .config is named .config.bak.*month_day_year_hour_min_sec*.

To compile the kernel tree in MontaVista Linux, change directory to the Linux kernel source tree, and type:

```
make oldconfig bzImage
```

To compile the kernel tree in Wind River Linux, change directory to the Linux working directory (i.e., containing the /dist subdirectory), and type:

```
make -C dist linux.rebuild
```

(or, from the root of the Linux kernel source tree, type:)

```
make ARCH=ppc CROSS_COMPILE=powerpc-wrs-linux-gnu- oldconfig bzImage
```

When first running "make oldconfig" or when first using xmake (described below), especially when there is a change in the board architecture from the base BSP, you may be prompted for configuration options on the command line. If this happens, just press enter to accept the default for all of the questions.

In order to accomplish this customization, more *OS & Libraries* parameter settings were made available in the *Software Platform Settings* dialog box within XPS, as described earlier in this document.

Note: If you want XPS to leave your kernel .config file alone, then please leave the *target_dir* parameter blank and copy the resulting sparse tree from the XPS project into your working kernel tree.

2. *target directory* is left blank and the config updater tool provided by Xilinx is used

   If the target directory is left blank or does not point to a valid Linux kernel source tree, the user must copy the sparse Linux tree from the project area to the working kernel tree. Xilinx provides a Tcl command script that can be run *after* the sparse tree copy to update the .config to match the XPS hardware design.

   In order to run the command, be sure an appropriate Tcl/TK interpreter is installed on the host system where the working Linux kernel tree resides. If this host system has the Xilinx EDK tools installed, then the appropriate interpreter is already installed and you can use the EDK/cygwin shell to perform the next step. Otherwise, be sure the host has a Tcl/TK interpreter 8.0 or newer installed.

   To update the .config file, change directory to the Linux kernel source tree, then type:

   ```
   $ tclsh cow.tcl
   ```

   Once the Tcl script is run once, there is no need to run it again unless a new BSP is generated from Platform Studio. Also, the user must compile the kernel after the script is run so configuration changes can take effect.

   Note that you can alternatively use *xmake* in place of the standard make command to update the .config file. You would, for example, type:

   ```
   ./xmake bzImage
   ```

   in the root of the Linux kernel source tree. The *xmake* tool will first invoke the above Tcl script and then invoke the standard Linux make command.

3. *target directory* is left blank and the kernel is configured manually

   If the target directory is left blank, the user must copy the sparse Linux tree from the project area to the working kernel tree. The user can then manually configure the kernel using “make menuconfig” or its equivalent. See the section below on Manual Kernel Configuration.

Manual Kernel Configuration

This section gives details on manually configuring the kernel for Xilinx-related IP. The user may not need to use these steps if the *target directory* specified points to a valid Linux kernel tree, which means the kernel .config was updated during the BSP generation process to match the XPS hardware design.

The default kernel configuration file that comes with a Linux 2.6 distribution contains a general set of kernel options. MontaVista Linux kernel source comes with predefined kernel configuration files for various development boards. One of these other configuration files may be a better starting point for your needs. These other configuration files can be found at

```
linux/arch/ppc/configs
```

in the Linux kernel source tree. To use one of these configuration files, simply copy the desired configuration file to

```
linux/.config
```

It is a good idea to first save a copy of the original configuration file, .config, in case the original configuration needs to be restored in the future.

One of the common methods for configuring the Linux kernel is to use the command

```
make menuconfig
```

There are several other methods for configuring the Linux kernel, but for this guide, we will describe configuration options using the *make menuconfig* method.

Covering every kernel configuration option for the various versions of the Linux kernel is beyond the scope of this guide. However, we include information using MontaVista Linux 4.0.1 as an example on how to accomplish some of the initial development tasks that will likely be encountered in your project. Configuration options for other Linux distributions and other kernel versions may vary from the examples.

Booting From a Compact Flash Card (using SystemACE)

There are various boot loaders that can be selected for booting Linux from a Compact Flash (CF) card. However, Xilinx and related boards often provide an alternative boot method called Booting from SystemACE. Booting from SystemACE is different from other boot loaders in that it also loads a hardware bitstream into the FPGA.

Booting from SystemACE involves the following steps:

1. Generate the hardware bitstream
2. Build the Linux kernel
3. Create the SystemACE file
4. Partition the CF card
5. Copy the SystemACE file to the CF card

Boards that provide booting from SystemACE have a chip, called a SystemACE chip, which will read an inserted CF card and look for a file with a .ace extension. This ace file will contain the hardware bitstream along with possibly an executable program. The SystemACE chip then loads the FPGA with the hardware bitstream, and if there is an executable program, it will load the program into memory and begin executing it.

Keep in mind that the hardware bitstream can also have an application that runs in BRAM in the FPGA. When booting from SystemACE, it is recommended to have such an application in the hardware bitstream such as a bootloader or bootloop application. This will ensure that the processor doesn't execute random instructions in the timing window between when the FPGA is programmed and when the application in the ace file is loaded and run. When in doubt, just use the processor bootloop program in XPS.

To create the bitstream, select *Update Bitstream* from the *Device Configuration* menu of XPS. This will make sure the hardware system bitstream is up-to-date and will merge in the BRAM application into a file called *download.bit*. After the Linux kernel has been built, create the ace file with the *genace.tcl* script. This *genace.tcl* script will merge the hardware bitstream with an *elf* executable into a single file with a *.ace* extension. Information on this script can be found in the *Platform Studio User's Guide*.

In order to boot from SystemACE, the ace file needs to be in the first partition on the CF card. This partition needs to be formatted to have the DOS file system on it. This DOS partition may need to be created on the CF card and should be large enough to hold the ace file. Extra space for additional ace files may be desired as well. 10 megabytes should be sufficient for most situations. If multiple ace files are being managed on the CF card, refer to the *SystemACE Compact Flash Solution* data sheet (DS080), which can be found on the Xilinx web site (http://www.xilinx.com).

> **Note**: The Xilinx Virtex-II Pro ML300 comes with a 1GB Microdrive. Instead of repartitioning a CF card, this Microdrive, which also has partitions for swap and a Linux root filesystem, may be used instead. See the Xilinx web site at http://www.xilinx.com/products/boards/ml300/index.htm if you need to restore the image on your Microdrive. A CF card image can be found for the Virtex-4 ML403 board at http://www.xilinx.com/ml403 under the Demos and Reference Designs link.

Finally the ace file needs to be copied to the DOS partition on the CF card. Refer to the Xilinx System ACE datasheet for information on where in the DOS partition the ace file needs to reside. To boot the system, just make sure the CF card is in the proper card reader slot before powering on the board. If everything is in proper order, the SystemACE chip will take care of the rest.

Setting Up Ethernet

This section will specifically describe the steps needed to get Ethernet working on the Xilinx boards. Other development boards may be set up similarly, so this section may still be useful for other boards.

In the Xilinx Virtex-II Pro ML300 and Virtex-4 ML403, a unique Ethernet MAC address for the board is stored in an EEPROM. The EEPROM is accessed from the FPGA using an I2C bus, and therefore an I2C controller within the FPGA. The Linux BSPs for these Xilinx boards attempt to read the MAC address over I2C during initialization. In Linux, if the MAC address cannot be found for one reason or another, a default MAC address is used. This default MAC address is fine to use. Though, it is not very convenient if your network will have many of these development boards attached, as each board will need different kernel software each with a different default MAC address. The default MAC address is defined in *arch/ppc/boot/simple/embedded_config.c.*

In many cases there is a need to use Ethernet without an I2C bus, thus the I2C driver will not be present. In other cases, other methods of retrieving the MAC address are desired. If, for whatever reason, there is a need to use Ethernet without the I2C driver, in *arch/ppc/boot/simple/embed_config.c*, the line

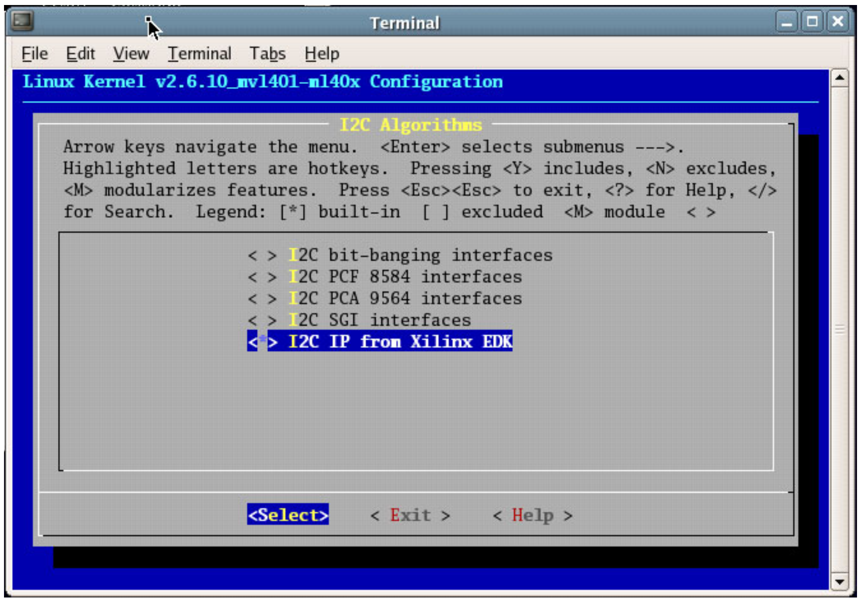
```
#error I2C needed for obtaining the Ethernet MAC address
```

may need to be removed or changed into a warning or comment line to avoid a compilation error.

In order to retrieve this MAC address, the I2C bus is used to read the EEPROM containing the MAC address. To use Ethernet with the stored MAC address, the following steps need to be done to configure the Linux kernel:

1. Enable the I2C driver in the kernel
2. Enable the Ethernet driver in the kernel

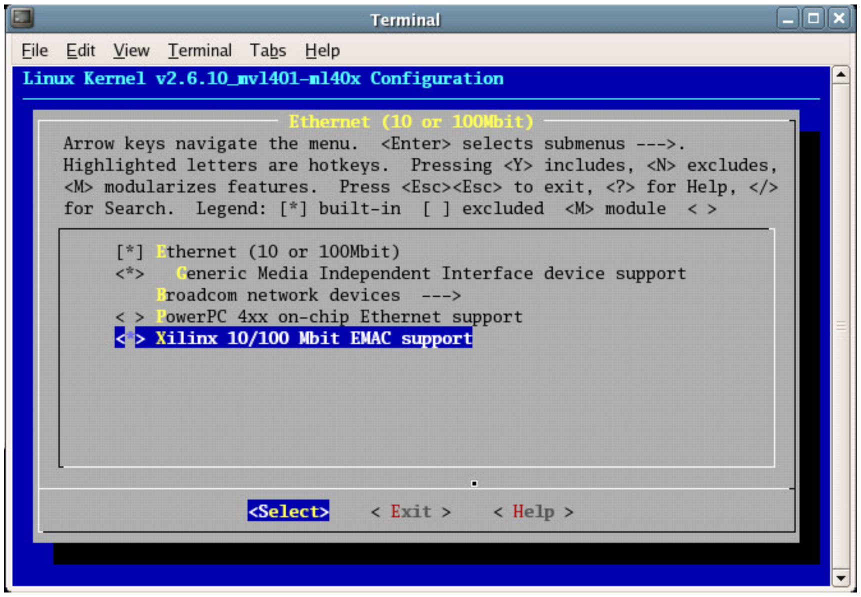
Note: A common mistake is to get to this section of the document, and try to enable the I2C driver while forgetting that the I2C IP core is not included in the hardware design of the FPGA. Remember to make sure the I2C IP core is included in the hardware configuration.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration

                              I2C Algorithms
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

                   < > I2C bit-banging interfaces
                   < > I2C PCF 8584 interfaces
                   < > I2C PCA 9564 interfaces
                   < > I2C SGI interfaces
                   < > I2C IP from Xilinx EDK

                   <Select>    < Exit >    < Help >
```

Figure 5: **Enabling I2C in the Kernel**

The I2C driver can be enabled in the kernel by selecting *Device Drivers->I2C Support->I2C Algorithms*, then *I2C IP from Xilinx EDK* in the *make menuconfig* menus. If the root filesystem will reside on NFS, it will be best to have the I2C driver included in the kernel. Otherwise the I2C driver can be built as a module..



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration

                          Ethernet (10 or 100Mbit)
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

       [*] Ethernet (10 or 100Mbit)
       <*>   Generic Media Independent Interface device support
           Broadcom network devices  --->
       < > PowerPC 4xx on-chip Ethernet support
       < > Xilinx 10/100 Mbit EMAC support

                   <Select>    < Exit >    < Help >
```

Figure 6: **Enabling 10/100 Ethernet in the Kernel**

The Ethernet driver can be enabled in the kernel from *make menuconfig* by selecting *Device Drivers->Networking support->Ethernet (10 or 100Mbit)* then *Xilinx 10/100 Mbit EMAC support* or *Ethernet (1000 Mbit)* then *Xilinx 10/100/1000 Mbit TEMAC support*. If the root file system will reside on NFS, this driver should be built into the kernel. Otherwise it can be built as a module. By default, the BSPs for the Xilinx boards in Linux have 10/100 Ethernet enabled in the kernel (not as a module).

Linux Root Filesystem Setup

The location of the root filesystem can reside in a number of different places irrespective of how the system boots. The root filesystem may reside on an NFS network share, on a CF card, or even get loaded into RAM, among other choices.

There are various methods for creating the root filesystem contents including using vendor tools. Describing how to create a root filesystem or even describing what executables are needed on the root filesystem is beyond the scope of this guide. Some good resources for getting help in this area are:

Building Embedded Linux Systems (http://www.oreilly.com/catalog/belinuxsys/index.html)

Linux From Scratch Project (http://www.linuxfromscratch.org)

Filesystem Hierarchy Standard (http://www.pathname.com/fhs)

Note: If the Xilinx ML300 or ML403 is being used and you're using MontaVista Linux, you can start with the root filesystem that is on the Microdrive or CompactFlash card that comes with the board. Note that the root filesystem on the CF card was built for MontaVista Linux and is not guaranteed to work with Wind River Linux.

In an embedded system, there is often a need or desire to separate the static root filesystem (used for boot-up processes) from an area that is storing transient, field use data or end user data. This separation can prevent dynamic data from accidentally overwriting system files or simply filling up the root filesystem preventing the system from booting.

Linux supports a wide range of filesystems such as Ext2, Ext3, ReiserFS, JFS, XFS, and others. The root filesystem for many embedded systems will remain mostly static. In this case a good filesystem to use is Ext2, which is widely used and is well tested.

If the embedded system will be writing many files, in particular large files, the XFS file system is a good choice. Ext3 is also commonly used. Note that using Ext3 or XFS on a CF card is not particularly useful, as CF cards are relatively slow and have a relative low capacity. If all that is being written or modified is configuration data written through well-defined methods, using a single root partition should be sufficient. Use of Ext3, XFS, or some other filesystems is more beneficial when there is a fairly fast or large-capacity storage medium in the embedded system such as, say, a hard disk.

Using the Root Filesystem on a Compact Flash Card

This section explains how to use a root filesystem on a CF card, accessed through System ACE. If the root file system is to reside on the CF card, the following steps are needed:

1. Partition the CF card
2. Create file systems on the CF card
3. Copy the root filesystem files and directories
4. Configure the kernel to compile in the CF card drivers
5. Configure the kernel boot parameters to use the root file system on the CF card

If a CF card needs to be repartitioned to hold the root filesystem, an easy way to partition it is to attach a CF card reader to a Linux workstation and use the Linux tools for partitioning drives. Linux *fdisk* seems to work well. On a system here, the CF card could be accessed through */dev/sda*, though this may be different on your system.

Note: If the CF card is also going to be used to boot from System ACE, remember to leave the first partition as a DOS partition.

Technically, a swap partition is not needed. However, having a swap partition will increase the amount of virtual memory available. The recommended swap partition size is usually twice the size of RAM. Though, more or less swap space may be specified depending on the system needs. When creating a swap partition, remember to set the partition's type to Linux Swap (type 82).

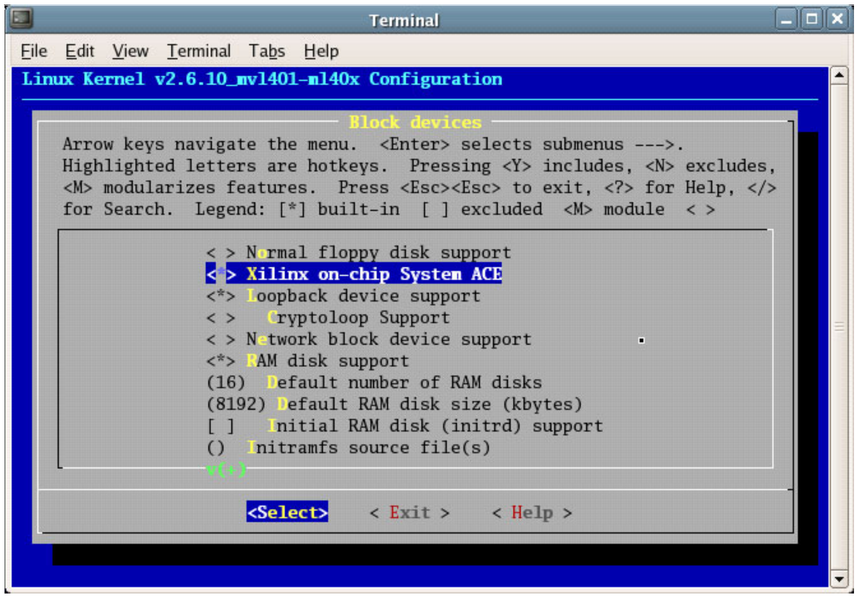
The partitions for the root filesystem should be large enough to hold the files being placed in the root filesystem. A helpful utility for determining the root filesystem size, if it's in a staging area, is *du*. When creating the root partition, remember to set the partition's type to Linux (type 83).

Once the partitions have been created, the file system on the Linux partitions will need to be created. Some Linux tools such as *parted* are capable of creating the empty filesystem at the time each partition is created. Otherwise, a tool such as *mkfs* will be needed.

Often the root filesystem contents are placed in a staging area allowing a whole directory tree to be copied over at once. When copying such a directory tree, it is a good idea to get the file and directory attributes set correctly before performing the copy. Make sure when performing the copy to use a command that will preserve the attributes such as *cp -a*, or *tar*.

After the root filesystem files have been copied over to the CF card, the kernel will need to be configured to use that root filesystem. In order for the kernel to be able to read the CF card, the SystemACE kernel driver will need to be enabled.

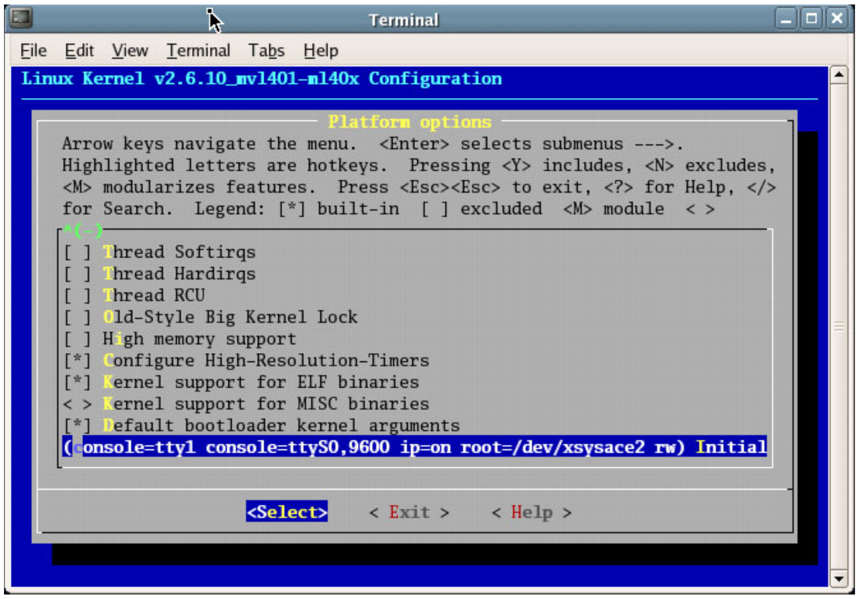
> **Note**: A common mistake at this point might be to have forgotten to include the SystemACE IP core in your hardware project in XPS.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration

                    Block devices
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

              < > Normal floppy disk support
              <*> Xilinx on-chip System ACE
              <*> Loopback device support
              < >   Cryptoloop Support
              < > Network block device support
              <*> RAM disk support
              (16)  Default number of RAM disks
              (8192) Default RAM disk size (kbytes)
              [ ]   Initial RAM disk (initrd) support
              ()  Initramfs source file(s)
              v(+)

                 <Select>    < Exit >    < Help >
```

Figure 7: **Selecting SystemACE kernel support**

The SystemACE driver can be enabled by selecting *Device Drivers->Block Devices, then Xilinx on-chip SystemACE* in the menu from *make menuconfig*. Make sure support for this device is included in the kernel, not as a module. See Figure 5, page 13. By default in the BSP for the Xilinx boards in Linux, this option is enabled.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                    Platform options
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
 ^(-)
 [ ] Thread Softirqs
 [ ] Thread Hardirqs
 [ ] Thread RCU
 [ ] Old-Style Big Kernel Lock
 [ ] High memory support
 [*] Configure High-Resolution-Timers
 [*] Kernel support for ELF binaries
 < > Kernel support for MISC binaries
 [*] Default bootloader kernel arguments
 (console=tty1 console=ttyS0,9600 ip=on root=/dev/xsysace2 rw) Initial
```

Figure 8: **Initial Kernel Command String Option**

Next, edit the *initial kernel command string* option. This option can be found by selecting *Platform options* in the main menu of *make menuconfig* , to tell the kernel where the root filesystem resides. *Default bootloader kernel arguments* must also be selected for this option to appear. The *root*= item of this initial kernel command string should, among other options, contain

root=/dev/xsysace2 rw

where *N* is the partition number of the root file system on the CF card.

By default in the BSPs for the Xilinx boards, this option is set up to use the root filesystem on a CF card.

Using the Root Filesystem in a RAM disk

When using a RAM disk for the root filesystem, the RAM disk image gets linked in with the kernel image. The process for using a root filesystem is nearly identical to using an initial RAM disk (initrd). The only difference is that in the boot sequence instead of performing a pivot root to the root filesystem on a different medium, the kernel performs a pivot root back to the RAM disk file system. Note that as of this writing Wind River Linux does not officially support a ramdisk rootfs.

Here are the steps needed to use a RAM disk root filesystem:

1. Create the RAM disk file
2. Configure the kernel to have the RAM disk driver
3. Configure the kernel initial command string to use the root from RAM disk.
4. Build the kernel so it includes the ram disk.

MontaVista Linux Professional Edition 4.0 includes a pre-built RAM disk image. This pre-built image is about 6MB uncompressed and most likely provides the contents sufficient for your initial development. If this pre-built RAM disk image can be used, the first step of building a RAM disk file can be skipped. Directions for how to make use of this pre-built image are described below. If for some reason this RAM disk image will not work for your project, a different RAM disk image will have to be created. The Wind River Linux distribution does not contain a pre-built RAM disk image.

The RAM disk starts out as an image file containing the file system that will be loaded into memory at boot time. This RAM disk image should hold a standard ext2 filesystem. The easiest way to create the image file, is to use an available Linux workstation and to start with the following commands:

```
dd if=/dev/zero of=initrd.img bs=1k count=<kbytes size>
mke2fs -F -v -m0 initrd.img
```

With the above commands, an empty RAM disk image is created. The next step is to mount that image file, and copy the root filesystem files to it. To mount the image file to, say, */mnt/tmp*, the following command can be used:

```
mount -o loop initrd.img /mnt/tmp
```

Now, copy the files and directories, preserving their attributes, to the RAM disk root filesystem, and then unmount it. Remember to use *cp -a* or *tar* when performing the copy so that file and directory attributes can be preserved. The *umount* program is used to unmount a filesystem. If your image is mounted on */mnt/tmp* as in the example above, the command

```
umount /mnt/tmp
```

will unmount the file system in the image file.

Finally compress the image file using the following command:

```
gzip -9 < initrd.img > ramdisk.image.gz
```

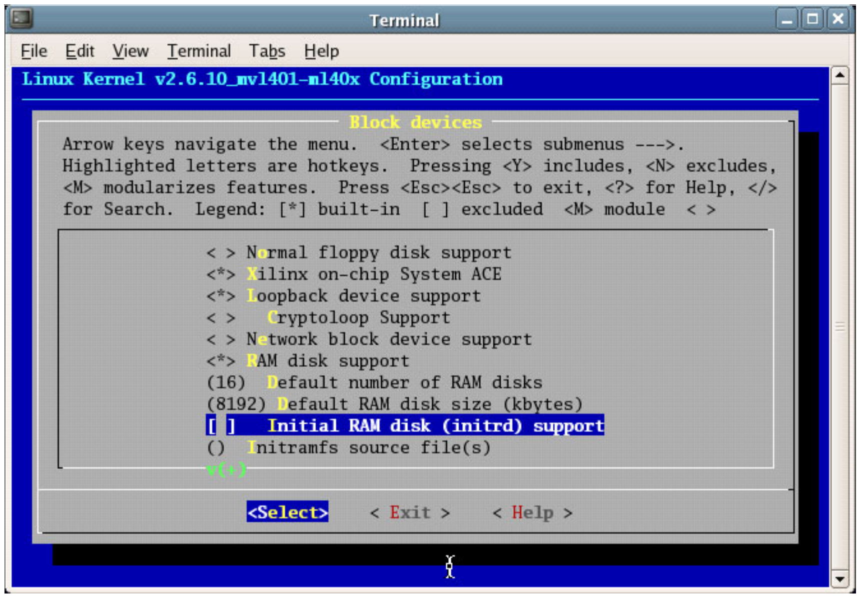
And then copy *ramdisk.image.gz* to the following directory in your working Linux kernel source tree:

```
arch/ppc/boot/images
```

If you are using the pre-built RAM disk image from MontaVista, simply copy the pre-built image to the Linux kernel source tree. Here is an example, assuming you have the ppc405 tools from MontaVista installed in the default location. From the top of the linux working kernel tree, the following should work:

```
cp /opt/montavista/pro/devkit/ppc/405/images/ramdisk.gz arch/ppc/boot/images/ramdisk.image.gz
```

Now that the RAM disk image file has been created and is in the right place, the kernel must be configured to use it. The RAM disk driver must be enabled and the kernel initial command string must be set so it uses the RAM disk as the root filesystem.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                          Block devices
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

              < > Normal floppy disk support
              <*> Xilinx on-chip System ACE
              <*> Loopback device support
              < >   Cryptoloop Support
              < > Network block device support
              <*> RAM disk support
              (16)  Default number of RAM disks
              (8192) Default RAM disk size (kbytes)
              [ ]   Initial RAM disk (initrd) support
              ()  Initramfs source file(s)
              v(+)

                 <Select>    < Exit >    < Help >
```

Figure 9: **RAM Disk Options**

The options for enabling RAM disk support in the kernel can be found in *make menuconfig* by selecting *Device Drivers->Block devices* in the top-level menu as shown in Figure 7, page 15. To set up the kernel for a RAM disk root, *RAM disk support* as well as *Initial RAM disk (initrd) support* must both be enabled in the kernel (not as a module). The *Default RAM disk size* should be set to a value a little larger than the actual RAM disk image uncompressed size so that there is room for temporary files used during boot up. Making the RAM disk size in the kernel 8K larger than the image uncompressed size has been observed to work well.

To configure the kernel to use the RAM disk as it's root file system, the *Initial kernel command string* must be modified. This option can be found in *make menuconfig* by selecting *Platform options* in the main menu as shown in Figure 6, page 13. Note that *Default bootloader kernel arguments* must also be selected for this option to appear.

To continue using the initial RAM disk as your root file system, you will have to set the *root=* item of the *initial kernel command string* to have

    root=/dev/ram rw

Note there should be no other *root=* options on this command string. Wind River Linux may use

    root=/dev/ram0 rw

instead.

By default in the BSPs Xilinx boards in Linux, this option is not set up to use a RAM disk root filesystem. The part of the line that reads

    root =/dev/xsysace2

needs to be replaced with the correct text, as described above.

Configuring an NFS Root Filesystem

Most embedded systems won't be using NFS to store the root filesystem in the final product. However, using NFS for the root filesystem during development can be useful. With an NFS root filesystem, you don't need to worry about size requirements. This is especially useful when working with temporary debug files. It is also much easier to update an NFS root filesystem, as opposed to CF cards or RAM disk images, when, during development, new programs are discovered to be necessary.

To use an NFS share for the root file system, there are three steps:

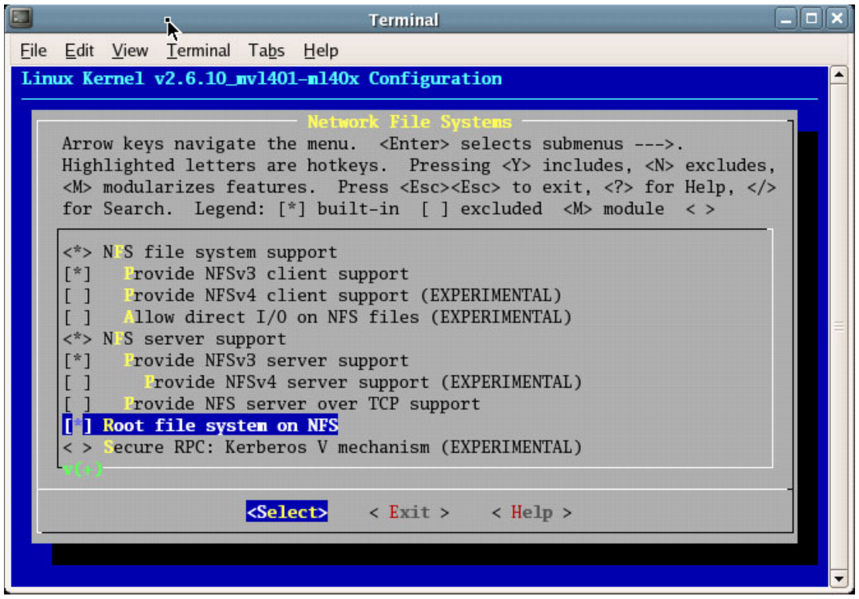
1. Create the root filesystem on an NFS share
2. Setup Ethernet (see above)
3. Configure the kernel boot parameters to use an NFS root

To create the root filesystem on an NFS share, put the target root file system files in a directory tree that will be exported through NFS. Keep in mind that the files for this share are not necessarily the same as those that run on the host system. In fact, most of the time, the system hosting the NFS will not have the same processor architecture as the target system. There are many resources available that describe how to share a directory through NFS. NFS will not be fully covered here. A basic NFS share can be created on a Linux host system by adding

    <directory path> *(rw)

to the file */etc/exports*, and then restart the NFS daemon. Keep in mind that you have to be logged in as root in order to edit that file and to restart the NFS daemon.

To set up Ethernet in the kernel see the section, "Setting Up Ethernet," page 12.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration

                     Network File Systems
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

 <*> NFS file system support
 [*]   Provide NFSv3 client support
 [ ]   Provide NFSv4 client support (EXPERIMENTAL)
 [ ]   Allow direct I/O on NFS files (EXPERIMENTAL)
 <*> NFS server support
 [*]   Provide NFSv3 server support
 [ ]     Provide NFSv4 server support (EXPERIMENTAL)
 [ ]   Provide NFS server over TCP support
 [*] Root file system on NFS
 < > Secure RPC: Kerberos V mechanism (EXPERIMENTAL)
 v(+)

            <Select>    < Exit >    < Help >
```

Figure 10: **Root File System on NFS**

Lastly, the kernel must be told to use NFS for the root filesystem along with the location on the network of the NFS to use. In *File systems->Network File Systems*, enable *Root file system on NFS*. See Figure 8, page 16. The *Initial kernel command string* is again modified to complete the settings to use the root filesystem over NFS. This option can be found in *make menuconfig* by selecting *Platform options* in the main menu as shown in Figure 6, page 13.

If using an NFS server and share, the *root=* item of the *initial kernel command string* should be replaced with:

```
ip=on nfsroot=<nfs share> rw
```

Here is an example

```
ip=on nfsroot=192.168.1.10:/export/ml300_root rw
```

If using DHCP to retrieve the NFS server and share name, the root= item should contain:

```
ip=on root=/dev/nfs
```

By default in the BSPs for the Xilinx boards in Linux, this option is not set up to use the root filesystem over NFS. The part of the line that reads

```
root =/dev/xsysace2
```

needs to be replaced with the correct text as described above.

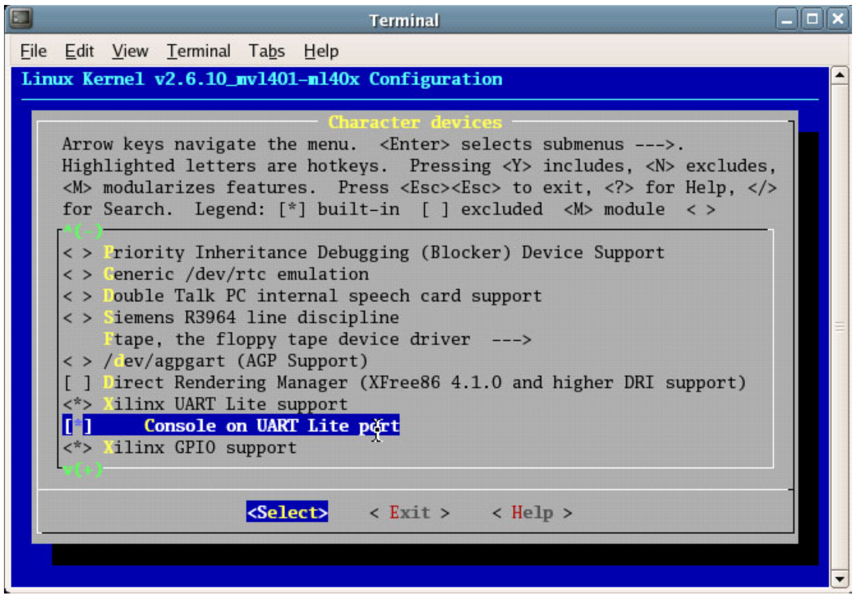
Configuring a Serial/UART Main Console

Linux provides for various consoles to be connected. Some may be over a serial port while others are through a keyboard and monitor. The main console is the console on which the kernel displays messages. Other consoles merely provide additional login sessions.

To have the main console use a serial port, there are three steps:

1. Configure the kernel to include a UART driver
2. Configure the UART driver to include support for the main console
3. Configure the kernel boot parameters to use UART as primary console

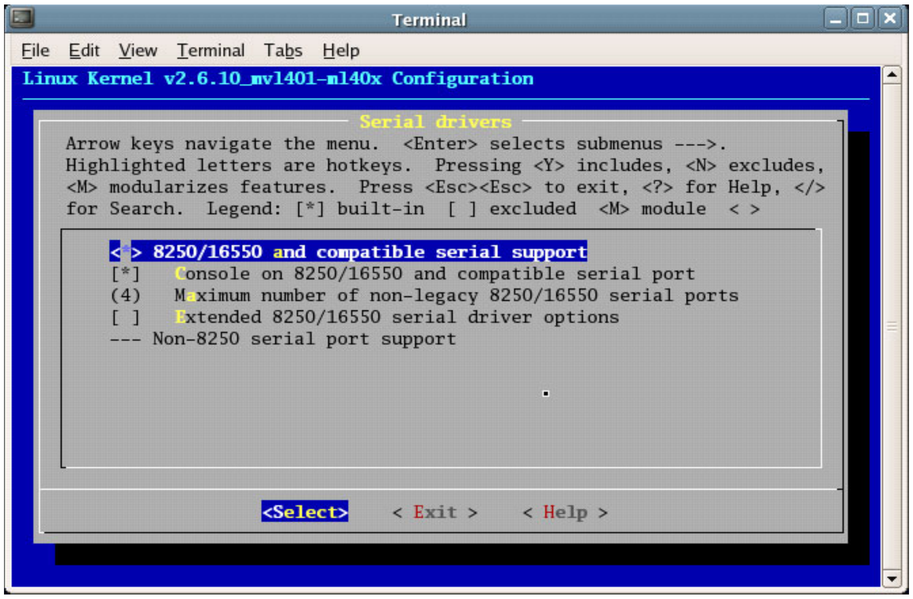
To set up the kernel so that it uses the serial port for the main console, the correct UART driver needs to be enabled according to the hardware configuration. The UART Lite driver is for the UART Lite IP core that may be present in your hardware configuration. Otherwise the standard Linux UART driver is used.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                     Character devices
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
 ^(-)
 < > Priority Inheritance Debugging (Blocker) Device Support
 < > Generic /dev/rtc emulation
 < > Double Talk PC internal speech card support
 < > Siemens R3964 line discipline
     Ftape, the floppy tape device driver  --->
 < > /dev/agpgart (AGP Support)
 [ ] Direct Rendering Manager (XFree86 4.1.0 and higher DRI support)
 <*> Xilinx UART Lite support
 [*]     Console on UART Lite port
 <*> Xilinx GPIO support
 v(+)
              <Select>    < Exit >    < Help >
```

Figure 11: **Kernel Configuration for UART Lite**

The option to enable the *Xilinx UART Lite* driver can be found under the *Device Drivers->Character devices* menu the main menu in *make menuconfig*. If using the UART Lite driver for the main console over a serial port, this driver must be enabled in the kernel, as opposed to being a module. Once *Xilinx UART Lite* has been properly enabled, *Console on UART Lite port* must also be selected.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                     Serial drivers
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    <*> 8250/16550 and compatible serial support
    [*]   Console on 8250/16550 and compatible serial port
    (4)   Maximum number of non-legacy 8250/16550 serial ports
    [ ]   Extended 8250/16550 serial driver options
    --- Non-8250 serial port support

              <Select>    < Exit >    < Help >
```

Figure 12: **Kernel Configuration for UART 16550**

If instead of the UART Lite IP core, the UART16550 IP core is used, the standard Linux 16550 serial driver should be used. This option can be found under the *Device Drivers->Character devices->Serial drivers* menu. Just as with the UART Lite, this driver should be included in the kernel instead of being a module, and the *Console on 8250/16550 and compatible serial port* option should be enabled as well.

Once a UART driver has been enabled and it has been configured to allow a serial console, the *Initial kernel command string* option needs to be modified. See Figure 6, page 13.

Make sure the *Initial kernel command string* option contains the following text:

for UART Lite (WR GPP LE 1.3 and MVL 4.0.1):

console=ttl<port number>

for UART Lite (WR GPP LE 2.0):

console=ttyUL<port number>

for UART 16550:

console=ttyS<port number>,<baud rate>

where the port number refers to which serial port is used and the baud rate is the speed of the serial port. For example, for a 16550 serial console on the first serial port with a baud rate of 9600, the string looks like

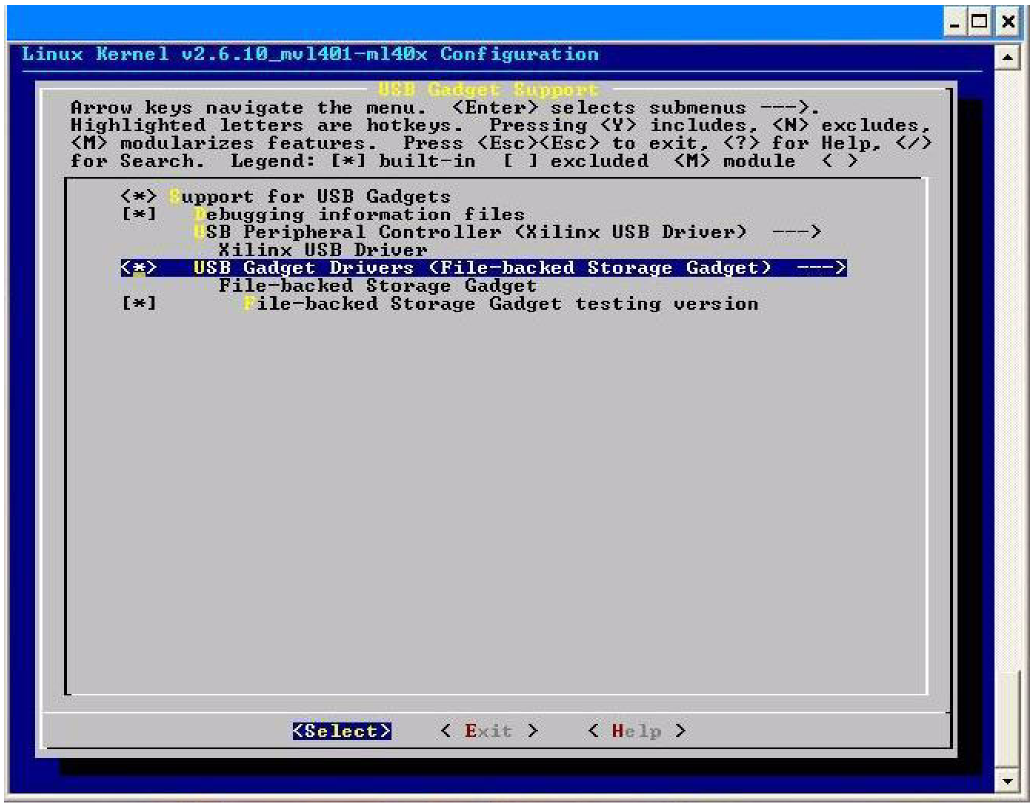
console=ttyS0,9600

When using a string, as in the example above, a client terminal application should connect using 9600 bps, 8 data bits, no parity, and 1 stop bit. Note that the UART Lite baud rate is fixed at hardware build time, so there is no need to specify it in software.

Sometimes there may be uncertainty about how the main console is set up and how subsequent consoles are set up. With the main console on the serial port, all of the boot messages will be sent to the serial port. This is generally the desired outcome for embedded development. This way any error messages presented there can be seen during the boot up sequence. The method described here to set up the main serial console should not be confused with methods for attaching additional console through the file /etc/inittab.

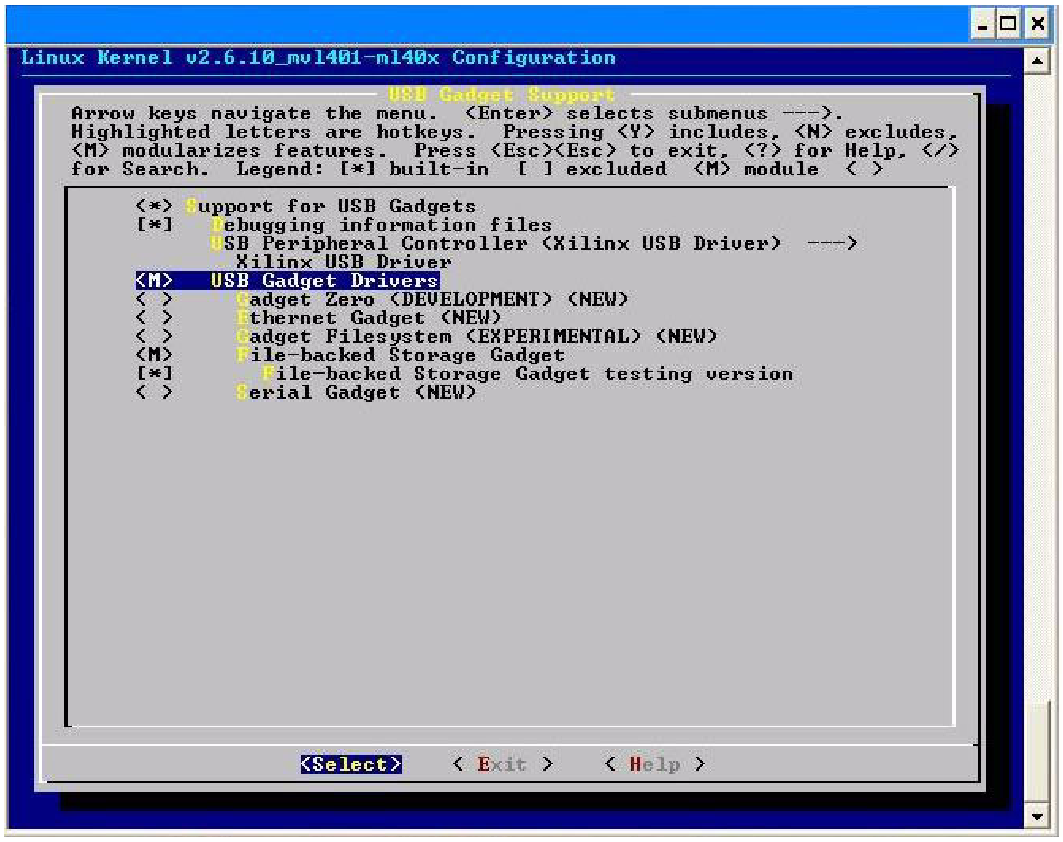
Configuring the USB Peripheral Controller Driver and the Gadget Driver

The Xilinx USB Peripheral Controller driver supports only the File-backed Storage Gadget Driver. In the generated BSP the File-backed Storage Gadget is enabled by default as shown in Figure 13, page 22. The File-backed Storage Gadget should be selected as a module in *Device Drivers -> USB Support-> USB Gadget Support* in the menu from *make menuconfig* as shown in Figure 14, page 23.



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                    USB Gadget Support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      <*> Support for USB Gadgets
      [*]   Debugging information files
            USB Peripheral Controller (Xilinx USB Driver)  --->
              Xilinx USB Driver
      <*>   USB Gadget Drivers (File-backed Storage Gadget)  --->
              File-backed Storage Gadget
      [*]       File-backed Storage Gadget testing version

                   <Select>    < Exit >    < Help >
```

Figure 13: **Default USB Gadget Driver Support**



```
Linux Kernel v2.6.10_mvl401-ml40x Configuration
                         USB Gadget Support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

       <*> Support for USB Gadgets
       [*]   Debugging information files
             USB Peripheral Controller (Xilinx USB Driver)  --->
               Xilinx USB Driver
       <M>   USB Gadget Drivers
       < >     Gadget Zero (DEVELOPMENT) (NEW)
       < >     Ethernet Gadget (NEW)
       < >     Gadget Filesystem (EXPERIMENTAL) (NEW)
       <M>     File-backed Storage Gadget
       [*]       File-backed Storage Gadget testing version
       < >     Serial Gadget (NEW)

                    <Select>    < Exit >    < Help >
```

Figure 14: **Configuring USB Gadget Driver Support as a Module**

Configuring PCI and related peripherals

This section describes the configuration options for the ML410 board. If you are using the automatic configuration update feature in Platform Studio (cow.tcl or xmake), you can ignore this section.

The following is a quick list of the "make menuconfig" options for each of the PCI devices on the ML410 using MVL 4.0.1.

- Enable Bus Options -> PCI Support
  - Enable PCI device name database
- Enable Device Drivers -> ATA/ATAPI/MFM/RLL support -> ATA/ATAPI/MFM/RLL support
  - Enable Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy/support
    - Enable Include IDE/ATA-2 DISK support
    - Enable Include IDE/ATAPI CDROM support
    - Enable Include IDE/ATAPI TAPE support
    - Enable PCI IDE chipset support
      - Enable Sharing PCI IDE interrupts source
      - Enable Boot off-board chipsets first support
      - Enable Generic PCI IDE chipset support
      - Enable Generic PCI bus-master DMA support

            - Enable ALI M15x3 chipset support
            - Enable PROMIS PDC202{46|62|65|
- Enable Device Driver -> SCSI device support -> SCSI device support
    - Enable SCSI generic support
- If using a 3COM Ethernet card:
    - Enable Device Driver -> Networking support -> Ethernet (10 or 100Mbit) -> 3COM cards
        - Enable 3c590/3c900 series (592/595/597) Vortex/Bommerang support
- Enable Device Driver -> USB support -> Support for Host-side USB
    - Enable USB device filesystem

Note that the above options may vary depending on your needs.

Linux Devices Reference

To reduce the amount of time spent searching for files and settings, Table 1 provides the relationship between driver modules, Xilinx provided IP cores, location of driver source files, and kernel config items.

Table 1: **Drivers and IP cores supported in Linux**

Xilinx Driver	Xilinx IP Core	Driver Location in LSP	Linux Kernel Config Item
n/a	opb_uart16550 plb_uart16550 xps_uart16550	linux/drivers/serial/8250.c	Device Drivers/Character devices/Serial drivers/ 8250/16550 and compatible serial support
uartlite	opb_uartlite xps_uartlite	linux/drivers/char/xilinx_uartlite	Device Drivers/Character devices/Xilinx UART Lite support
emac	opb_ethernet plb_ethernet	linux/drivers/net/xilinx_emac	Device Drivers/Networking support/Network device support/Ethernet (10 or 100Mbit)/Xilinx 10/100 Mbit EMAC support
temac	plb_temac	linux/drivers/net/xilinx_temac	Device Drivers/Networking support/Network device support/Ethernet (1000 Mbit)/Xilinx 10/100/1000 Mbit TEMAC support
lltemac	xps_ll_temac	linux/drivers/net/xilinx_temac	Device Drivers/Networking support/Network device support/Ethernet (1000 Mbit)/Xilinx 10/100/1000 Mbit TEMAC support
llfifo	xps_ll_fifo	linux/drivers/xilinx_common	n/a
lldma	mpmc w/ sdma ppc440 dma	linux/drivers/xilinx_common	n/a
n/a	opb_intc dcr_intc xps_intc	linux/arch/ppc/syslib/xilinx_pic.c	n/a
iic	opb_iic xps_iic	linux/drivers/i2c/algos/xilinx_iic	Device Drivers/I2C support/I2C Algorithms/I2C IP from Xilinx EDK
n/a	plb_tft_cntlr_ref	linux/drivers/video/xilinxfb.c	n/a
touchscreen_ref	opb_tsd_ref	linux/drivers/char/xilinx_ts	n/a

Table 1: **Drivers and IP cores supported in Linux** ***(Contd)***

Xilinx Driver	Xilinx IP Core	Driver Location in LSP	Linux Kernel Config Item
ps2_ref	opb_ps2_dual_ref opb_ps2_ref	linux/drivers/input/serio/xilinx_ps2	Device Drivers/Input device support/Xilinx PS/2 Controller Support
spi	opb_spi xps_spi	linux/drivers/char/xilinx_spi	n/a
sysace	opb_sysace xps_sysace	linux/drivers/block/xilinx_sysace	Device Drivers/Block devices/Xilinx on-chip System ACE
gpio	opb_gpio plb_gpio xps_gpio	linux/drivers/char/xilinx_gpio	Device Drivers/Character devices/Xilinx GPIO support
usb	opb_usb2_device xps_usb2_device	linux/drivers/usb/gadget/xilinx_usb	Device Drivers/USB support/USB Gadget Support/Support for USB Gadget/USB Peripheral Controller (Xilinx USB Driver) Device Drivers/USB support/USB Gadget Support/Support for USB Gadget/USB Gadget Drivers
common	n/a	linux/drivers/xilinx_common	n/a

Driver Configuration and the Platform Bus

The Linux 2.6 distributions and Xilinx device drivers have begun to move toward the platform bus model for driver initialization. This means that (almost) all Xilinx drivers no longer depend on xparameters.h, nor do they use the _g.c file for driver configuration. Instead, the files *virtex.c* and *virtex.h* in arch/ppc/platforms/4xx and *xilinx_devices.h* in include/linux specify the Xilinx driver configurations. Device drivers get their configuration from the platform bus structures populated in the *virtex.c* file. The *virtex.** files currently depend on xparameters.h, but in the future could be made to depend instead on non-static configuration data, such as data passed to the kernel from a bootloader. The ultimate intent being that the kernel tree need not be recompiled in order to reconfigure Xilinx device drivers.

Getting More Information

If you have a question or problem associated with the Xilinx IP or drivers associated with such IP, then you should contact Xilinx Support. The Xilinx support web site is here:

http://support.xilinx.com

Or you can call the main US technical support hotline at one of the following numbers:

1 800-255-7778

For other regions, see the Xilinx support web site for additional phone numbers.

Otherwise, if you have purchased MontaVista or Wind River Linux, you are entitled to support from those vendors.

The Internet also contains a wealth of information on Linux. A few Internet resources can be found here:

Linux From Scratch Project (http://www.linuxfromscratch.org)

Filesystem Hierarchy Standard (http://www.pathname.com/fhs)

In addition to just using web searches, there are also various email lists you can search or join. At the time of this writing, one such list called *linuxppc-embedded*, can be found here:

https://ozlabs.org/mailman/listinfo/linuxppc-embedded

The archives for this list can be found here:

http://ozlabs.org/pipermail/linuxppc-embedded/

O'Reilly also publishes several good books on using and developing software for Linux. Here are some such books that you might find useful:

Running Linux, 4th Edition (http://www.oreilly.com/catalog/runux4/)

Building Embedded Linux Systems (http://www.oreilly.com/catalog/belinuxsys/index.html)

Understanding the Linux Kernel, 3rd Edition (http://www.oreilly.com/catalog/understandlk/index.html)

Linux Device Drivers, 3rd Edition (http://www.oreilly.com/catalog/linuxdrive3/index.html)

Linux Network Administrator's Guide (http://www.oreilly.com/catalog/linag3/index.html)