Automatic Generation of Linux Support Packages Through Xilinx Platform Studio (XPS)

Summary

This document describes the development of the Board Support Structure Package (BSP). The document contains the following sections.

- “Overview”
- “Getting Started with MontaVista Linux”
- “How to Create a BSP from XPS”
- “Directory Structures”
- “Linux Kernel Configuration”
- “MontaVista Linux Devices Reference”
- “Getting More Information”

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 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 software drivers for Xilinx provided IP cores for their FPGA line of products. These drivers can be selected based upon the current hardware configuration defined in XPS using a process called Automatic BSP Generation.

Automatic generation of a BSP is done using the 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. XPS supports the MontaVista Linux distribution.

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 MontaVista Linux as the host operating system in XPS
3. Specify Operating System Parameters
4. Generate the BSP in XPS
5. Configure the kernel
6. Build the kernel
7. Define the root file system
8. Install the kernel and root filesystem
9. Develop and run application specific code

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

MontaVista Linux

MontaVista uses the term LSP instead of BSP. LSP stands for Linux Support Package. MontaVista distributes the entire Linux kernel source tree in the LSPs that they provide. MontaVista distributes fixed LSPs for reference designs on various development boards with Xilinx FPGAs for their Linux distribution for Power PC (PPC).

The provided LSP from MontaVista should be sufficient when using development boards along with the reference hardware design for the board. The user may want to use the automatically generated BSP when using a different hardware design.

Getting Started with MontaVista Linux

MontaVista Linux Professional Edition 3.1 can be purchased from MontaVista ([http://www.mvista.com](http://www.mvista.com)) which comes with technical support along with the MontaVista development tools. MontaVista also has a preview kit that can be obtained freely from their web site.

MontaVista provides a Linux cross development system that runs on a host x86 workstation using Linux or Windows. See the MontaVista web site for a list of supported host operating systems. The main install of the MontaVista development tools currently comes on 5 compact disks. A fixed LSP with which to start can be selected during the compact disk installation.

At the time of this writing, the fixed LSP for the Xilinx Virtex-II Pro ML300 is found on the lsps-ppc_405_mvl3.1.0 CD, while other LSPs need to be downloaded from the support pages on MontaVista’s web site. Currently there are two LSPs for boards that use the Xilinx Virtex-II Pro chip, the Xilinx Virtex-II Pro ML300 and the Insight/Memec Virtex-II Pro FG456. If the target board is a custom board, select a MontaVista LSP for the board that is closest in design to the target board.

Even if an LSP that is close cannot be found, some fixed LSP still needs to be chosen as it includes the Linux kernel source tree. In this case it is probably easiest to install the LSP for the Xilinx Virtex-II Pro found on the lsps-ppc_405_mvl3.1.0 CD.

At a minimum, the target board needs to have a PPC405 chip with a MMU. 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 filesystem system needs to be considered.

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. To open the **Software Platform Settings** dialog box, select **Software Platform Settings**… from the **Project** menu. See **Figure 1**.

![Software Platform Settings Dialog Box](image)

**Figure 1:** Selecting the Target Operating System

From the drop down list in the **Kernel and Operating Systems** section of the dialog box, select **linux_mvl31**, then click OK.

Setting Operating System Parameters

Some values in the **Library/OS Parameters** tab of the **Software Platform Settings** dialog box are required to be set properly. Others are optional. See **Figure 2**.

![Software Platform Settings Dialog Box](image)

**Figure 2:** Setting Operating System Parameters
MEM_SIZE (required)

The MEM_SIZE setting lets the OS know how much general-purpose RAM is available for use in the system. The MEM_SIZE value should be less than or equal to the amount of physical general-purpose RAM available in the system. 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.

TARGET_DIR (optional)

The target directory where the BSP is created can be specified in the Software Platforms Settings dialog box by setting a value for TARGET_DIR. Typically, the TARGET_DIR 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 Linux directory in the working kernel tree.

If this target directory is left blank, it defaults to

```
project directory\processor name\libs\linux_mvl31_v1_00_a\linux
```
Usually it is better to make a copy of the MontaVista Linux kernel source tree and then make modifications to the copy rather than risking the possibility of damaging the original source tree.

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-2.4.20_mvl31
```

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

```
C:\MontaVista\opt\montavista\pro\devkit\lsp\target board\linux-2.4.20_mvl31
```

For the preview kit, the default Linux kernel tree is here:

```
/opt/montavista/previewkit/lsp/target board/linux-2.4.20_mvl31
```

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-ml300-ppc_405/linux-2.4.20_mvl31 | tar xvf - C my_linux-2.4.20
```

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

**IIC Parameters (optional)**

The IIC parameters are based around the hardware on the Xilinx Virtex-II Pro ML300 board. This board has an EEPROM attached to an I2C bus. In addition, MontaVista 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 the development 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 Virtex-II Pro ML300 board that write to the EEPROM.

The `IIC_PERSISTENT_EEPROMADDR` value specifies the I2C bus device ID of the EEPROM.

**POWERDOWN Parameters (optional)**

The `POWERDOWN` parameters are placeholders to aid in creating a soft power down feature in the board. The Xilinx Virtex-II Pro ML300 supports a soft power down feature through a GPIO address.
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.

connected_periphs (required)

The connected_periphs parameter specifies which hardware devices are to be supported in Linux through the generated BSP. See Table 1.

Generating the BSP

Selecting Generate Libraries and BSPs from the Tools menu to create the BSP as shown in Figure 3.

Directory Structures

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

```
project directory\processor name\libsrc\linux_mvl31_v1_00_a\linux
```
This Linux directory 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, this directory needs to be copied into the working Linux kernel source tree.

Copy the BSP to the Linux Kernel Source Tree

Once the working Linux kernel source tree is in place, if necessary, copy the BSP generated by XPS into the working Linux kernel directory structure. Notice that 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 directory tree respectively.

Note that if the TARGET_DIR option in the Library O/S Parameters settings refers to the working Linux kernel source tree, the BSP does not need to be copied.

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 default kernel configuration file that comes from MontaVista contains a general set of kernel options. The 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 the board being used. 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, copy the desired configuration file to

    linux/.config

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

Regardless of which configuration file is used, every hardware platform and Linux kernel environment cannot be anticipated, so it is likely the kernel configuration will need to be modified.

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, the configuration options will be described using the `make menuconfig` method.

Covering every kernel configuration option is beyond the scope of this guide. However, this document includes information on how to accomplish some of the initial development tasks that will likely be encountered in the project.

**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 on board 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, use the processor bootloop program in XPS.

To create the bitstream, select **Update Bitstream** from the **Tools** 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. For the Xilinx Virtex-II Pro ML300, start a Xygwin shell from the XPS tools menu and use the following command:

```
Xmd genace.tcl -jprog -board ml300 -hw implementation/download.bit -elf
../linux/arch/ppc/boot/images/zImage.embedded -ace system.ace
```

For other boards, different options will have to be given to the **genace.tcl** script. 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/ise/embedded/ml300_v2pdk/members](http://www.xilinx.com/ise/embedded/ml300_v2pdk/members), if the image on the Microdrive needs to be restored.

Finally the ace file needs to be copied to the DOS partition on the CF card. Note that there should be only one ace file in this root directory. 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**

In the Xilinx Virtex-II Pro ML300, the MAC address for the Ethernet core is stored in an EEPROM. These Xilinx boards are each shipped with unique MAC addresses in this EEPROM. This section will specifically describe special steps needed to get Ethernet working on the Xilinx Virtex-II Pro ML300. Other development boards may be set up similarly, so this section may still be useful for other boards.

In MontaVista Linux, if the MAC address cannot be found for one reason or another, a default MAC address is used. It is acceptable to use this default MAC address, although, it is not very convenient if the network is to have many of these development boards attached, because each board will need different kernel software each with a different default MAC address.

Some information on how to begin changing the default MAC address can be found in the ML300 fixed BSP from the MontaVista Linux installation in a file called, **mv1-xilinx-ml300-GigE.readme**.
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, around line 719, in arch/ppc/boot/simple/embedded_config.c, the line,

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

needs to be removed or changed into a comment line.

In order to retrieve this MAC address, the IIC (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 frequent oversight occurs when the user reaches this section of the document. They may attempt to enable the I2C driver, while forgetting that the I2C IP core is not included in the hardware design of the FPGA. Remember to ensure that the I2C IP core is included in the hardware configuration.

The I2C driver can be enabled in the kernel by selecting Character Devices, then I2C, then Xilinx on-chip I2C 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. By default, the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux has this item enabled as a module.

The Ethernet driver can be enabled in the kernel from make menuconfig by selecting Network device support, then Ethernet (10 or 100Mbit) or Ethernet (1000 Mbit), then Xilinx on-chip ethernet. Again, if the root filesystem will reside on NFS, this driver should be built into the kernel. Otherwise, it can be built as a module. By default, the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux has 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 get loaded into RAM, among other choices.

The contents required for the root filesystem might vary greatly. Root filesystems can be as small as 5MB or as large as 10GB, depending on the type of system being built and the required software programs. The Linux root filesystem on the Microdrive that is included with the Xilinx Virtex-II Pro ML300 is about 700MB.

There are various methods for creating the root filesystem contents including using MontaVista’s DevRocket tool. Describing how to create a root filesystem or describing what executable files 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 Virtex-II Pro ML300 is being used, the user can start with the root filesystem that is on the Microdrive that comes with the board.

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 filling up the root filesystem, thereby preventing the system from booting.

Linux supports a wide range of file systems 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 relatively 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 file systems 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. If the root filesystem 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 filesystem 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`, although accessing may be different on the user’s system.
**Note:** If the CF card is also going to be used to boot from SystemACE, 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 that 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 the entire directory tree to be copied over at once. When copying such a directory tree, ensure that the file and directory attributes are set correctly before performing the copy. To perform the copy, 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 must be configured to use that root filesystem. To enable the kernel to read the CF card, the SystemACE kernel driver must be enabled.

**Note:** At this point, the user may have forgotten that the I2C IP core is not included in the hardware design of the FPGA. The user must ensure that the SystemACE IP core has been included in the hardware project in XPS.

The SystemACE driver can be enabled by selecting **Xilinx on-chip SystemACE** in the **Block devices** menu from **make menuconfig**. Ensure that support for this device is included in the kernel, not as a module. See **Figure 6**. By default in the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux, this option is enabled correctly.
Next, edit the initial kernel command string option as shown in Figure 7. This option can be found by selecting General Setup 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

```bash
geroot=/dev/xsysace/disc0/partN rw
```

where `N` is the partition number of the root filesystem on the CF card.

By default in the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux, this option is not set up to use the root filesystem on a CF card. The part of the line that reads, `root=/dev/nfs`, needs to be replaced with the correct text, as described above.
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.

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 3.1 includes a pre-built RAM disk image. This pre-built image is about 6MB, uncompressed, and most likely provides sufficient contents for 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 the project, a different RAM disk image will have to be created.

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 file system. 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,
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 the image is mounted on `/mnt/tmp` as in the example above, the command, `umount /mnt/tmp`, will unmount the filesystem 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 the working Linux kernel source tree:

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

If the pre-built RAM disk image from MontaVista is being used, copy the pre-built image to the Linux kernel source tree. From the top of the linux working kernel tree, the following should work, assuming the MontaVista tools are installed in the default location:

```
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 correct location, 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.

The options for enabling RAM disk support in the kernel can be found in `make menuconfig` by selecting `Block devices` in the top-level menu. See Figure 8. 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 its root filesystem, the `Initial kernel command string` must be modified. This option can be found in `make menuconfig` by selecting `General Setup` in the main menu. See Figure 7, Initial Kernel Command String Option. Note that Default bootloader kernel arguments must also be selected for this option to appear.

To continue using the initial RAM disk as the root filesystem, the user must set the `root=` item of the `initial kernel command string` to have `root=/dev/ram rw`.

There may be no other `root=` options on this command string.

By default in the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux, this option is not set up to use a RAM disk root filesystem. The part of the line that reads, `root=/dev/nfs`, must 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, the user need not be concerned 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 filesystem, 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 filesystem 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`, then restarting the NFS daemon. Keep in mind that the user must 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 above, Setting up Ethernet.

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. The Initial kernel command string is again modified to accomplish this. This option can be found in `make menuconfig` by selecting General Setup in the main menu. See Figure 7, Initial Kernel Command String Option.
The root= item of the initial kernel command string should contain ip=on nfsroot=<nfs share> rw.

Here is an example: ip=on nfsroot=192.168.1.10:/ml300_root rw

By default in the BSP for the Xilinx Virtex-II Pro ML300 in MontaVista Linux, this option is not set up to use the root filesystem over NFS. The part of the line that reads, root =/dev/nfs, must 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 the hardware configuration. Otherwise the standard Linux UART driver is used.

The option to enable the Xilinx UART Lite driver, can be found under the Character devices menu of the main menu in make menuconfig. See Figure 9. 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.
If instead of the UART Lite IP core, the UART16550 IP core is used, the standard Linux generic serial driver should be used. Just as with the UART Lite, this driver should be included in the kernel instead of being a module, and the Support for console on serial port option should be enabled as well. See Figure 10.

Once a UART driver has been enabled and configured to allow a serial console, the Initial kernel command string option must be modified. See Figure 8.

Make sure the Initial kernel command string option contains the text, Console=ttyS$_\text{port number baud rate}$, 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 serial console on the first serial port with a baud rate of 9600, the string will appear as:

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.

On occasion, the user may become confused about how the main console is set up and how subsequent consoles are set up. For console setups, note that:

1. 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. With this arrangement, any error messages presented there can be seen during the boot up sequence.

2. the method described to set up the main serial console should not be confused with methods for attaching additional console through the file, /etc/inittab.

Configuring an LCD Panel for the Main Console

Depending on the project, the user may want support for a LCD panel or other display device that can present more information than just text.
To get LCD support, the user must ensure that LCD support is set correctly in the kernel configuration. Support for frame buffer devices and Xilinx LCD display support can be selected in the console drivers menu as shown in Figure 11.

![Figure 11: LCD Display Support Option](image1)

In addition to setting LCD support, enable the Virtual terminal option. The Virtual terminal option can be found under the Character devices option, in the main menu as shown in Figure 12.

![Figure 12: Virtual Terminal Option](image2)
MontaVista Linux Devices Reference

If the user is new to MontaVista Linux, or Linux in general, there is much material to be learned, which may seem overwhelming at times. To reduce the time spent searching for files and settings, this document includes Table 1 to show 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

<table>
<thead>
<tr>
<th>Xilinx Driver</th>
<th>IP Core</th>
<th>Driver Location in LSP</th>
<th>Linux Kernel menuconfig Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>opb_uart16550</td>
<td>linux/drivers/8250.c</td>
<td>Character devices/Standard/generic (8250/16550 and compatible UARTs) serial support</td>
</tr>
<tr>
<td></td>
<td>plb_uart16550</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uartlite</td>
<td>opb_uartlite</td>
<td>linux/drivers/char/xilinx_uartlite</td>
<td>Character devices/Xilinx UART Lite</td>
</tr>
<tr>
<td>emac</td>
<td>opb_ethernet, plb_ethernet</td>
<td>linux/drivers/net/xilinx_enet</td>
<td>Network device support/Ethernet (10 or 100Mbit)/Xilinx on-chip ethernet</td>
</tr>
<tr>
<td>gemac (1)</td>
<td>plb_gemac</td>
<td>linux/drivers/net/xilinx_gige</td>
<td>Network device support/Ethernet (1000Mbit)/Xilinx on-chip ethernet</td>
</tr>
<tr>
<td>temac (1)</td>
<td>plb_temac</td>
<td>linux/drivers/net/xilinx_gige</td>
<td>Network device support/Ethernet (1000Mbit)/Xilinx on-chip ethernet</td>
</tr>
<tr>
<td>N/A</td>
<td>opb_intc, dcr_intc</td>
<td>arch/ppc/kernel/xilinx_pic.c</td>
<td>N/A</td>
</tr>
<tr>
<td>iic</td>
<td>opb_iic</td>
<td>linux/drivers/ide/xilinx_iic</td>
<td>Character devices/I2C support/Xilinx on-chip I2C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/A</td>
<td>plb_tft_cntlr_ref</td>
<td>linux/drivers/video/xilinxfb.c</td>
<td>Console drivers/Frame-buffer support/Xilinx LCD display support</td>
</tr>
<tr>
<td>touchscreen_ref</td>
<td>opb_tsd_ref</td>
<td>linux/drivers/char/xilinx_ts</td>
<td>Character devices/Xilinx touchscreen</td>
</tr>
<tr>
<td>ps2_ref</td>
<td>opb_ps2_dual_ref</td>
<td>linux/drivers/char/xilinx_keyb</td>
<td>General setup/PC PS/2 style Keyboard</td>
</tr>
<tr>
<td>spi</td>
<td>opb_spi</td>
<td>linux/drivers/char/xilinx_spi</td>
<td>Character devices/Xilins SPI</td>
</tr>
<tr>
<td>sysace</td>
<td>opb_sysace</td>
<td>linux/drivers/block/xilinxsysace</td>
<td>Block devices/Xilins on-chip System ACE</td>
</tr>
<tr>
<td>gpio</td>
<td>opb_gpio</td>
<td>linux/drivers/char/xilinx_gpio</td>
<td>Character devices/Xilins on-chip GPIO</td>
</tr>
</tbody>
</table>

Note: 1. temac and gemac cannot co-exist in the same EDK system for using the current BSP.

At this time, the following drivers are not supported in MontaVista Linux:
1. gpio 2.00.a
Instead, use the previous version of these drivers.

Getting More Information

To resolve a question or problem associated with the Xilinx IP or drivers associated with such IP, contact Xilinx support at: [http://support.xilinx.com/support/mysupport.htm](http://support.xilinx.com/support/mysupport.htm)

Or call the main US technical support hotline at: 1 800-255-7778

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

Otherwise, if the user has purchased MontaVista Linux Professional Edition, they are entitled to support from MontaVista.

Considerable information on Linux can be found on the internet, such as at:

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


In addition to just using web searches, there are also various email lists the user can search or join. At the time of this writing, one such list called *linuxppc-embedded*, can be found at: [https://ozlabs.org/mailman/listinfo/linuxppc-embedded](https://ozlabs.org/mailman/listinfo/linuxppc-embedded)
Getting More Information

The archives for this list can be found at:

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

Several books on using and developing software for Linux are published by O'Reilly Media, Inc., such as:


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

Understanding the Linux Kernel, 2nd Edition
(http://www.oreilly.com/catalog/linuxkernel2/index.html)

Linux Device Drivers, 3rd Edition