XAPP1140 (v1.0) July 27, 2009

Embedded Platform Software and Hardware In-the-Field Upgrade Using Linux

Author: Brian Hill

Summary

This application note discusses an in-the-field upgrade of the Virtex®-5 FXT bitstream, Linux kernel, and loader flash images, using the presently running Linux kernel. Upgrade files are obtained from a USB mass storage device using the XPS USB Host core or over the network from an FTP server.

Included Systems

Included with this application note is one reference system built for the Xilinx ML507 Rev A board. The reference system is available in the following ZIP file available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=133141

Introduction

New features and bug fixes often necessitate upgrading flash images to replace the existing FPGA bitstream, bootloader, Linux kernel, or file system. This presents a challenge to provide a convenient mechanism for end users to perform this task. This application note provides a reference system and an example methodology to perform an in the field flash upgrade. New images may be retrieved from a USB Mass Storage device or from a network server. The running Linux image performs the flash upgrade.

Target Audience

This application note best serves users who are already adept at building and using Linux.

Hardware And Software Requirements

The hardware and software requirements for this reference system are:

- Xilinx ML507 Rev A board
- Xilinx Platform USB or Parallel IV programming cable
- RS232 serial cable and serial communication utility (HyperTerminal)
- Xilinx Platform Studio 11.2
- Xilinx Integrated Software Environment (ISE®) 11.2
- Xilinx Open Source Linux
- Suitable PowerPC® processor toolchain and Linux Root File System, such as DENX ELDK.
- (optional) GIT revision control software
- SMSC EVB-USB3300-XLX USB Daughter Card
- Ethernet Cable
- FTP server

Reference System Specifics

The supplied PowerPC processor reference system is configured to boot from on board parallel flash. The system also contains DDR2 Memory Controller, IIC master, Interrupt Controller, Tri-Mode Ethernet MAC, USB Host Controller, 16550 UART, and External Memory Controller (parallel flash) IP cores. This application note utilizes the SMSC EVB-USB-XLNX daughter card to provide USB connectivity.

Note: The xps_usb_host IP core version 1.00.a only supports high speed USB devices. Full speed and low speed devices **will not operate** with this version of the core.

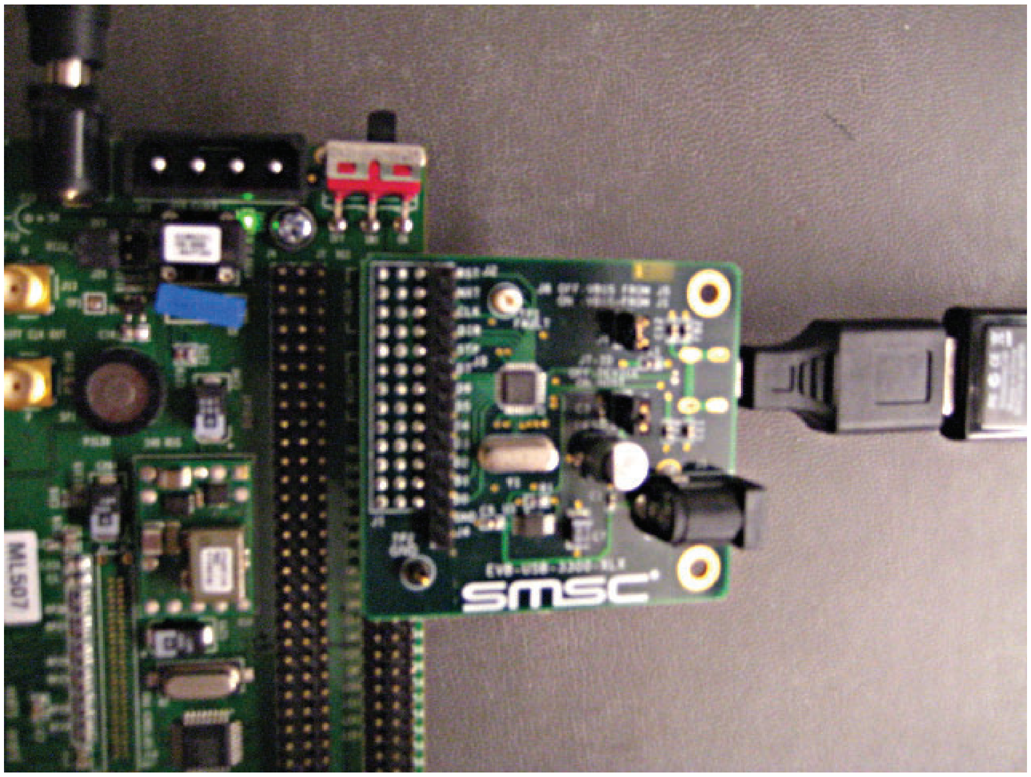


Figure 1: ML507 with SMSC EVB-USB3300-XLX USB Daughter Card

Address Map

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
ppc440mc_ddr2	DDR2_SDRAM	0x00000000	0x0FFFFFFF
xps_iic	IIC_EEPROM	0x81600000	0x8160FFFF
xps_intc	xps_intc_0	0x81800000	0x8180FFFF
xps_ll_temac	Hard_Ethernet_MAC	0x81C00000	0x81C0FFFF
xps_usb_host	xps_usb_host_0	0x82400000	0x824001FF
xps_uart16550	RS232_Uart_1	0x83E00000	0x83E0FFFF
xps_mch_emc	FLASH	0xFE000000	0xFFFFFFFF

Support Files

The reference system includes the following files which support this application note:

ready_for_download/

download.bit	FPGA bitstream
xapp1140.cmd	Instructions for iMPACT
xapp1140.opt	Commands for XMD
simpleImage.initrd.virtex440-ml507.elf	Bootable Linux image

linux/	
dotconfig	Linux kernel configuration
ramdisk.image.gz	Linux ramdisk image
virtex440-ml507.dts	Device tree hardware description
scripts/	
build_rom.pl	Generates a flash image suitable for use with the enclosed loader application.
mk_download.bin.sh	Converts download.bit to a file suitable for programming into flash with Linux.
upgrade.sh	Script which performs a flash upgrade.
upgrade-image/	
manifest	Upgrade description file
upgrade.tgz	Upgrade images
loader/	
loader.c	Simple boot loader
loader_linker_script.ld	Manually modified linker script. The loader .text is linked to the very end of flash memory.

Executing the Reference System

Using HyperTerminal or a similar serial communications utility, map the operation of the utility to the physical COM port to be used. Then connect the UART of the board to this COM port. Set the HyperTerminal to the Bits per second to **9600**, Data Bits to **8**, Parity to **None,** and Flow Control to **None**.

Executing the Reference System using the Pre-Built Bitstream and the Compiled Software Application

To execute the system using files in the `ready_for_download/` directory in the project root directory, follow these steps:

1. Change directories to the `ready_for_download` directory.
2. Use iMPACT to download the bitstream by using the following command:
```
impact -batch xapp1140.cmd
```
3. Invoke XMD and connect to the processor using the following command:
```
xmd -opt xapp1140.opt
```
4. Download and run the Linux executable using the following commands:
```
dow simpleImage.initrd.virtex440-ml507.elf
run
```
5. Proceed to the "Programming the Flash with Linux" section, using the upgrade files provided in the `ready_for_download/upgrade-image/` area.

Executing the Reference System from XPS for Hardware

To execute the system for hardware using XPS, follow these steps:

1. Open `system.xmp` in XPS.
2. Select **Hardware→Generate Bitstream** to generate a bitstream for the system.
3. Select **Device Configuration→Download Bitstream** to download the bitstream.
4. Invoke XMD and connect to the processor using the following command:

   ```
   xmd -opt xapp1140.opt
   ```
5. Download and run the Linux executable using the following commands:

   ```
   dow simpleImage.initrd.virtex440-ml507.elf
   run
   ```
6. Proceed to the "Programming the Flash with Linux" section, using the upgrade files provided in the `ready_for_download/upgrade-image/` area.

Obtaining the Software

The user will need to obtain source code for the Linux kernel, and the Linux kernel BSP generator in order to complete the tasks discussed in this application note. These are available on the Xilinx public GIT server. GIT is a distributed revision control system. Installation and usage of GIT are beyond the scope of this application note; consult XAPP1107 for additional information.

Obtaining the Software with GIT

Users which do not have GIT installed, or who do not wish to use GIT should proceed to the "Obtaining a Snapshot of the Software Without GIT" section.

Users which already have GIT properly installed may obtain the latest versions of the required software with the following commands:

1. Obtain the latest Linux 2.6 kernel

   ```
   $ mkdir <project area>
   $ cd <project area>
   $ git clone git://git.xilinx.com/linux-2.6-xlnx.git
   ```

(OPTIONAL) Revert to the version used with this application note. This version has been demonstrated to work as described in this document without modification. Perform after cloning the tree.

```
$ cd linux-2.6-xlnx
$ git checkout 6b06f54c
```

2. Obtain the latest device tree generator

   ```
   $ cd <project area>
   $ git clone git://git.xilinx.com/device-tree.git
   ```

(OPTIONAL) Revert to the version used with this application note. This version has been demonstrated to work as described in this document without modification. Perform after cloning the tree.

```
$ cd device-tree
$ git checkout 33b0797b
```

Obtaining a Snapshot of the Software Without GIT

A snapshot of the source tree may be obtained from git.xilinx.com as a compressed tar file.

The exact revisions used to create this application note can be obtained with the following links:

device-tree

linux-2.6-xlnx

Note: In the future these direct links may not be available, and the user may need to navigate to the desired snapshot directly from the git.xilinx.com page.

Obtaining a Toolchain Compiler

To build any of the software used in this application note, the user will require an appropriate PowerPC processor toolchain (compiler, linker, etc...). Linux will also require a Root File System. If the user does not already have these resources available, the DENX ELDK 4.1 is one example implementation which is freely available. This application note utilizes the ELDK, which can be found at http://www.denx.de/wiki/DULG/ELDK. Toolchain installation is beyond the scope of this application note.

Flash Organization

The onboard flash must be logically divided into four separate areas to contain the various objects needed to boot Linux in a standalone fashion. Table 2 shows the division chosen in this application note.

Table 2: **Flash partitions**

	Start Address	Offset	Size
FPGA Bitstream	`0xFE000000`	`0x00000000`	0x00400000 (4M)
Linux Kernel	`0xFE400000`	`0x00400000`	`0x00500000` (5M)
JFFS2 Filesystem	`0xFE900000`	`0x00900000`	`0x01600000` (22M)
(unused)	`0xFFF00000`	`0x01FF0000`	`0x000E0000` (896K)
Loadert	`0xFFFE0000`	`0x01FE0000`	`0x00020000` (128K)

Linux requires an explicit definition of all flash sections. This explicit definition is represented by Linux as partitions of the flash device, much like fixed disk or any other mass storage partition. This configuration is presented in the "Prepare the Device Tree for Linux" section.

Generate the Linux BSP

The device tree is a single text file which describes the hardware devices present in the system. The device tree generator is used to create this BSP.

Note: The user may wish to begin with the provided xapp1140/ready_for_download/linux/virtex440-ml507.dts device tree rather than creating one with the device tree generator.

Open the EDK project in XPS. Choose Software → Software Platform Settings. Choose **device-tree** in the OS & Library Settings list box. Select version **0.00.x**.

Note: The device tree generator does not presently support the Xilinx SDK application. Instead, XPS must be used as shown in this application note.

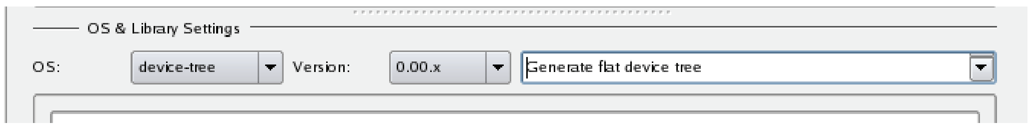
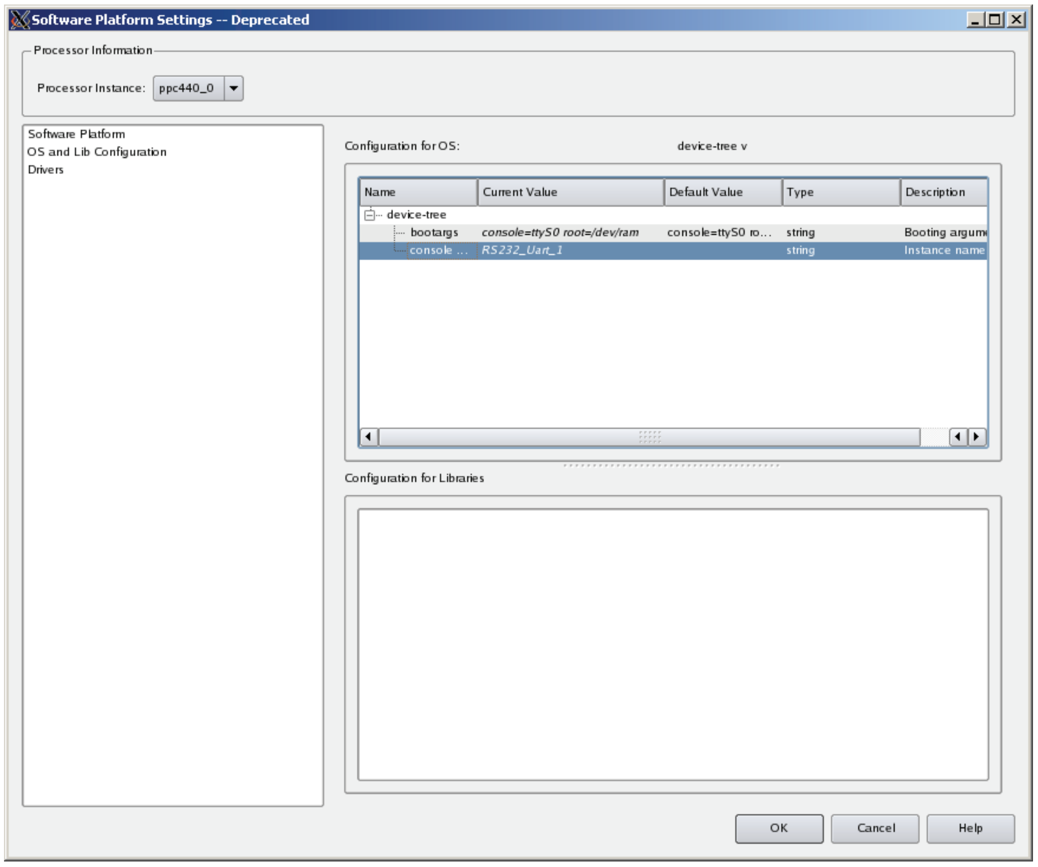


Figure 2: **OS & Library Settings**

Click OS and Lib Configuration. Expand the device-tree item and enter **RS232_Uart_1** in the console section. Click OK.


Software Platform Settings -- Deprecated
Processor Information
Processor Instance: ppc440_0
Software Platform
OS and Lib Configuration
Drivers
Configuration for OS: device-tree v
Configuration for Libraries
OK
Cancel
Help
XAPP1140_03_062209

Name	Current Value	Default Value	Type	Description
device-tree				
bootargs	console=ttyS0 root=/dev/ram	console=ttyS0 ro...	string	Booting argum
console ...	RS232_Uart_1		string	Instance name

Figure 3: **OS an Lib Configuration**

In XPS, select **Software** → **Generate Libraries and BSPs**.

Copy `<edk system>/ppc440_0/libsrc/device-tree/xilinx.dts` to `<project area>/linux-2.6-xlnx/arch/powerpc/boot/dts/virtex440-ml507.dts`

Prepare the Device Tree for Linux

The device tree file <project area>/linux-2.6-xlnx/arch/powerpc/boot/dts/virtex440-ml507.dts is edited to specify the proper kernel command line. The unique Ethernet MAC address is also specified in this file. The MAC address assigned to the user's board is found on a sticker on the bottom of the board.

The proper modifications are shown in **red**:

```
        chosen {
            bootargs = "console=ttyS0 ip=192.168.1.10 root=/dev/ram rw
    mtdparts=fe000000.flash:4M(bits),5M(zImage),22M(rootfs),896k(unused),128k
    (loader)";
            } ;
    ...
            Hard_Ethernet_MAC: xps-ll-temac@81c00000 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "xlnx,compound";
                ethernet@81c00000 {
                  compatible = "xlnx,xps-ll-temac-2.02.a", "xlnx,xps-ll-temac-
    1.00.a";
                    device_type = "network";
                    interrupt-parent = <&xps_intc_0>;
                    interrupts = < 3 2 >;
                    llink-connected = <&DMA0>;
                    local-mac-address = [ 00 0A 35 00 00 00 ];
                    reg = < 0x81c00000 0x40 >;
```

This specifies that the root file system is a ramdisk. The flash organization shown in "Flash Organization" is specified here. An ip address of 192.168.1.10 is statically assigned.

Build the Linux Kernel

Copy the Ramdisk Image

Copy the provided ramdisk image to the kernel tree:

```
$ cp <edk project>/ready_for_download/linux/ramdisk.image.gz <project
area>/linux-2.6-xlnx/arch/powerpc/boot
```

Configure the Kernel

The Linux kernel is configured to include the appropriate drivers needed to access the on board flash.

Indicate which toolchain is to be used. This below will work with a properly installed ELDK.

```
$ export CROSS_COMPILE ppc_4xx-
$ cd <project area>/linux-2.6-xlnx
```

Copy the default ML507 kernel configuration to use as a starting point

```
$ cp arch/powerpc/configs/44x/virtex5_defconfig .config
```

Build and run the kernel menu config application

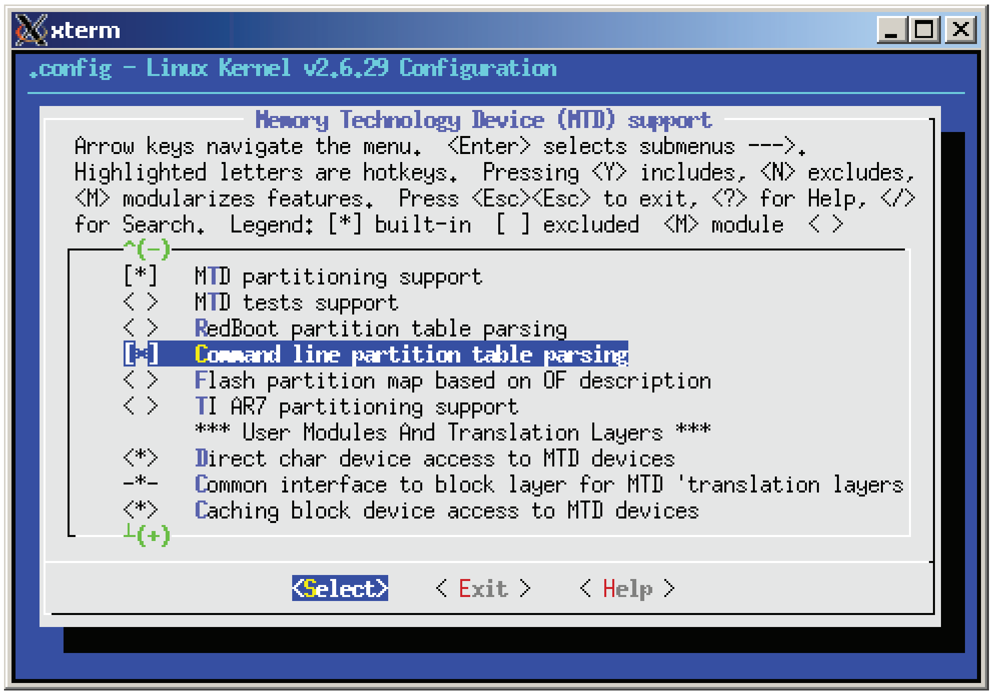
```
$ make ARCH=powerpc menuconfig
```

Note: The user may choose to begin with the provided `xapp1140/ready_for_download/linux/dotconfig` configuration file instead of performing the configuration process.

Submenus are chosen with <enter>, options are modified with <space>.

1. Enable **Device Drivers→ Memory Technology Device (MTD)** support (with the space bar, making an asterisk (*) appear).
2. Choose **Device Drivers→ Memory Technology Device (MTD)** support (enter)
   a. Enable **MTD partitioning support**
   b. Enable **Command line partition table parsing**
   c. Enable **Direct char device access to MTD devices**

d. Enable **Caching block device access to MTD devices**

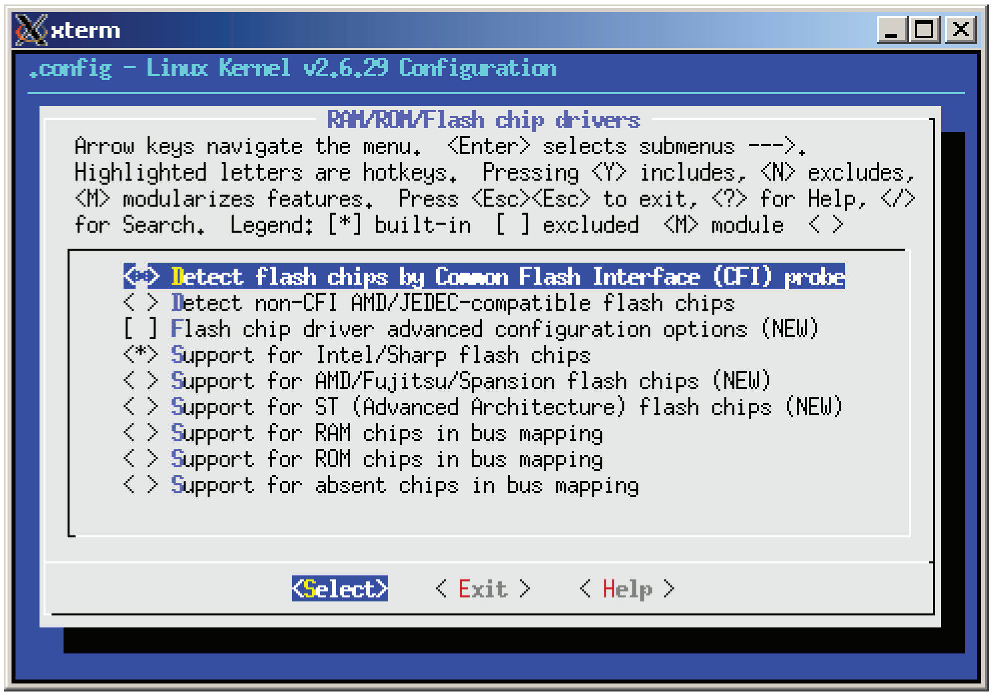


```
.config - Linux Kernel v2.6.29 Configuration
            Memory Technology Device (MTD) support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
      ^(-)
      [*]   MTD partitioning support
      < >   MTD tests support
      < >   RedBoot partition table parsing
      [*]   Command line partition table parsing
      < >   Flash partition map based on OF description
      < >   TI AR7 partitioning support
            *** User Modules And Translation Layers ***
      <*>   Direct char device access to MTD devices
      -*-   Common interface to block layer for MTD 'translation layers
      <*>   Caching block device access to MTD devices
      v(+)

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

XAPP1140_04_062209

Figure 4: **Memory Technology Device (MTD) Support**

3. Choose **Device Drivers**→ **MTD Support** → **RAM/ROM/Flash chip drivers**
   a. Enable **Detect flash chips by Common Flash Interface (CFI) probe**
   b. Enable **Support for Intel/Sharp flash chips**

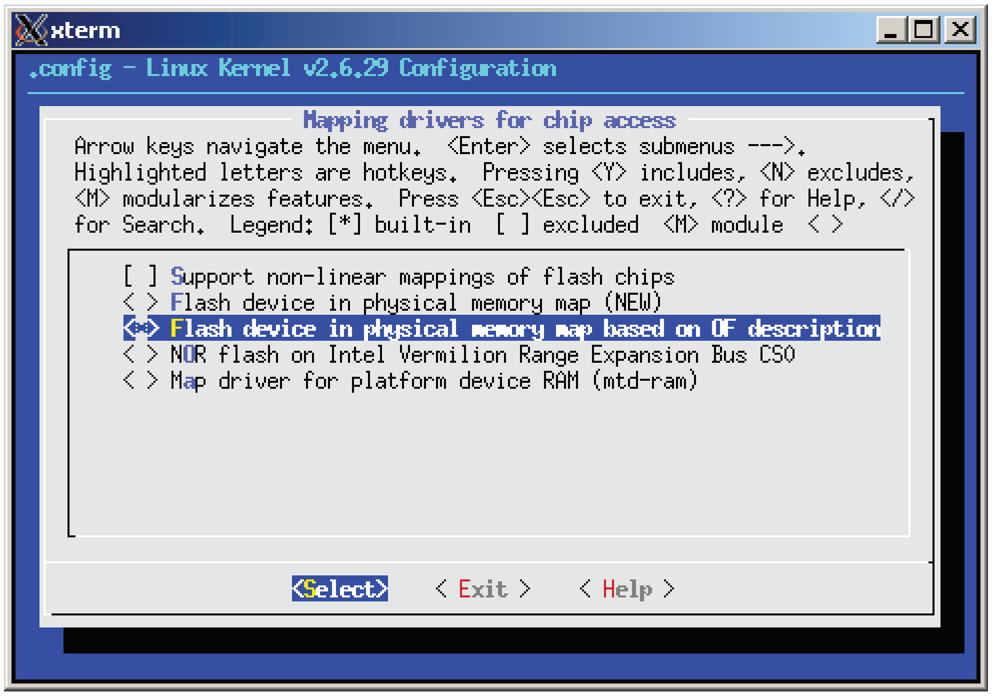


```
.config - Linux Kernel v2.6.29 Configuration
                 RAM/ROM/Flash chip drivers
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      <*> Detect flash chips by Common Flash Interface (CFI) probe
      < > Detect non-CFI AMD/JEDEC-compatible flash chips
      [ ] Flash chip driver advanced configuration options (NEW)
      <*> Support for Intel/Sharp flash chips
      < > Support for AMD/Fujitsu/Spansion flash chips (NEW)
      < > Support for ST (Advanced Architecture) flash chips (NEW)
      < > Support for RAM chips in bus mapping
      < > Support for ROM chips in bus mapping
      < > Support for absent chips in bus mapping

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

XAPP1140_05_062209

Figure 5: **MTD Flash Chip Drivers**

4. Choose **Device Drivers**→ **MTD Support** → **Mapping drivers for chip access**
   a. Enable **Flash device in physical memory map based on OF description**



```
.config - Linux Kernel v2.6.29 Configuration
                    Mapping drivers for chip access
  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 non-linear mappings of flash chips
      < > Flash device in physical memory map (NEW)
      <*> Flash device in physical memory map based on OF description
      < > NOR flash on Intel Vermilion Range Expansion Bus CS0
      < > Map driver for platform device RAM (mtd-ram)
```

XAPP1140_06_062209

Figure 6: **MTD mapping driver**

5. Choose **Device Drivers** → **SCSI device support**
   a. Enable **SCSI device support**
   b. Enable **SCSI disk support**
6. Enable **Device Drivers** → **USB support** (space)
7. Choose **Device Drivers**→ **USB support** (enter)
   a. Enable **Support of Host-side USB**
   b. Enable **USB device filesystem**
   c. Enable **EHCI HCD (USB 2.0) support**
   d. Enable **Use Xilinx usb host EHCI controller core**
   e. Enable **USB Mass Storage support**
8. Choose **File Systems** → **Native language support**
   a. Enable **Codepage 850**
   b. Enable **NLS ISO 8859-1**

Note: Users in different geographic locations may need to enable Native language support for their region (non Western European languages) to successfully mount vfat filesystems created locally.

9. Set **File Systems** → **DOS/FAT/NT Filesystems** → **Default codepage for FAT** to `850`
10. Enable **File Systems** → **Miscellaneous filesystems** → **Journalling Flash File System v2**

11. Exit and save the configuration.

Compile the kernel:

```
$ make ARCH=powerpc simpleImage.initrd.virtex440-ml507
```

Note: A prebuilt image `simpleImage.initrd.virtex440-ml507.elf` is provided in the `ready_for_download` area.

The new image is created in `linux-2.6-xlnx/arch/powerpc/boot/simpleImage.initrd.virtex440-ml507.elf.`

Note: Other Linux distributions and other hardware architectures often refeer to this target as a zImage.initrd.

The Loader

The simpleImage created in “Build the Linux Kernel” can not be executed directly from flash. A small loader is required to copy the simpleImage from flash to DRAM. Rather than a loader which parses the ELF headers of the simpleImage directly, the simpleImage is converted to an ordinary binary, and a header is prepended to indicate where this binary blob should be copied. This allows the loader to be very small and simple.

Generate a Binary Image of the ELF file

An absolute memory image of the Linux simpleImage is used in the flash, not the ELF file output by the linker. The Object Copy utility is used to copy segments from the ELF file to a binary image.

```
$ powerpc-eabi-objcopy -O binary simpleImage.initrd.virtex440-ml507.elf
linux.bin
```

The generated file `linux.bin` has no relocation information - the loader will not know where it should be copied from flash.

The readelf Utility

The readelf utility is used to display the ELF headers of an executable in a textual format. This data shows how the simpleImage should be relocated to DRAM. The data needed for the flash loader are shown in **red**:

```
$ powerpc-eabi-readelf -e simpleImage.initrd.virtex440-ml507.elf
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           PowerPC
  Version:                           0x1
  Entry point address:               0x4008bc
  Start of program headers:          52 (bytes into file)
  Start of section headers:          3351540 (bytes into file)
  Flags:                             0x8000, relocatable-lib
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         2
  Size of section headers:           40 (bytes)
  Number of section headers:         22
  Section header string table index: 19

Section Headers:
  [Nr] Name              Type      Addr     Off     Size    ES Flg Lk Inf Al
  [ 0]                   NULL     00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS 00400000 010000 008d98 00  AX  0   0  4
  [ 2] .data             PROGBITS 00409000 019000 001c64 00  WA  0   0  4
  [ 3] __builtin_cmdline PROGBITS 0040ac64 01ac64 000200 00  WA  0   0  4
  [ 4] .kernel:dtb       PROGBITS 0040ae68 01ae68 002bfd 00   A  0   0  1
  [ 5] .kernel:vmlinux.s PROGBITS 0040e000 01e000 1853ca 00   A  0   0  1
  [ 6] .kernel:initrd    PROGBITS 00594000 1a4000 1739e5 00   A  0   0  1
  [ 7] .bss              NOBITS   00708000 3179e5 00cddc 00  WA  0   0  4
  [ 8] .debug_abbrev     PROGBITS 00000000 3179e5 002747 00      0   0  1
  [ 9] .debug_info       PROGBITS 00000000 31a12c 00a3d6 00      0   0  1
  [10] .debug_line       PROGBITS 00000000 324502 001c91 00      0   0  1
  [11] .debug_frame      PROGBITS 00000000 326194 0016fc 00      0   0  4
  [12] .debug_loc        PROGBITS 00000000 327890 007f89 00      0   0  1
  [13] .debug_pubnames   PROGBITS 00000000 32f819 000863 00      0   0  1
  [14] .debug_aranges    PROGBITS 00000000 33007c 0002e0 00      0   0  1
  [15] .debug_str        PROGBITS 00000000 33035c 00190a 01  MS  0   0  1
  [16] .comment          PROGBITS 00000000 331c66 0003c0 00      0   0  1
  [17] .note.GNU-stack   PROGBITS 00000000 332026 000000 00      0   0  1
  [18] .debug_ranges     PROGBITS 00000000 332026 0002d0 00      0   0  1
  [19] .shstrtab         STRTAB   00000000 3322f6 0000fe 00      0   0  1
  [20] .symtab           SYMTAB   00000000 332764 001330 10     21 183  4
  [21] .strtab           STRTAB   00000000 333a94 000ec6 00      0   0  1

Program Headers:
  Type      Offset   VirtAddr   PhysAddr   FileSiz  MemSiz   Flg Align
  LOAD      0x010000 0x00400000 0x00400000 0x3079e5 0x314ddc RWE 0x10000
  GNU_STACK 0x000000 0x00000000 0x00000000 0x00000  0x00000  RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00 .text .data __builtin_cmdline .kernel:dtb .kernel:vmlinux.strip
.kernel:initrd .bss
   01
```

The data provided by readelf needed by the loader is

LOAD	The address where the executable begins and it's size. The simpleImage begins at `0x00400000`
Entry Point	The address of the first instruction of the executable.
.bss	The BSS, or Block Started by Symbol is not present within the ELF file. This segment is the location of uninitialized global data. The loader should zero this memory.

The build_rom.pl Script

The script build_rom.pl provided with this application note generates a binary image of the ELF file using objdump, parses the output of readelf, and prepends a header suitable for use with a simple loader to the binary image. The file format is shown in Table 3.

Table 3: **Loader image header format**

0	"XLNX"
1	Entry point address
2	BSS address
3	BSS size
4	Load address
5	Load size

Generate the Flash Image

The flash image for the Linux kernel is generated with the build_rom script:

```
$ <edk project>/ready_for_download/scripts/build_rom.pl
simpleImage.initrd.virtex440-ml507.elf
Parsing readelf output for simpleImage.initrd.virtex440-ml507.elf
Entry: 0x4008bc
BSS:   0x00708000 52700
LOAD:  0x00400000 0x3079e5
Generating image:
Appending header:
```

It is seen that the first six (6) words of the generated binary file contain the expected header information:

```
$ hexdump -C simpleImage.initrd.virtex440-ml507.elf.bin |head
00000000  58 4c 4e 58 00 40 08 bc  00 70 80 00 00 00 cd dc
00000010  00 40 00 00 00 30 79 e5  00 01 28 20 94 21 ff f0
00000020  7c 08 02 a6 42 9f 00 05  bf c1 00 08 7f c8 02 a6
00000030  90 01 00 14 80 1e ff f0  7f c0 f2 14 81 3e 80 00 |
00000040  80 09 00 14 2f 80 00 00  41 9e 00 0c 7c 08 03 a6
00000050  4e 80 00 21 48 00 00 00  00 01 27 e0 94 21 ff d0
00000060  7c 08 02 a6 42 9f 00 05  bf 61 00 1c 7f c8 02 a6
00000070  90 01 00 34 80 1e ff f0  7f c0 f2 14 83 fe 80 04
00000080  7f e3 fb 78 48 00 1a 89  2f 83 00 00 41 be 00 10
00000090  80 7e 80 08 48 00 14 35  4b ff ff 85 80 9e 80 0c
```

Generate the Loader

The loader provided with this application note is a XIlinx standalone BSP application. Generate a linker script for the application specifying that all segments apart from the heap and the stack should be in FLASH. The heap and the stack are assigned to DDR memory.

Note: The BSS, if used, should also be assigned to DDR. The loader application has no data in the BSS.

The EDK linker file generator will link items at the beginning of the selected memory. The flash has been partitioned for various uses, and the loader can not reside at the beginning of flash. The PowerPC processor boot vector is `0xFFFFFFFC`, which requires that the bootloader be at the end of flash. The generated linker script is edited to that the loader is placed at the end of flash.

The flash base address is set to match the loader location shown in "Flash Organization".

```
MEMORY
{
   DDR2_SDRAM_C_MEM_BASEADDR : ORIGIN = 0x00000000, LENGTH = 0x10000000
   FLASH_C_MEM0_BASEADDR : ORIGIN = 0xFFFE0000, LENGTH = 0x00020000
}
```

The application and the standalone BSP are configured to compile optimized for size -Os. Build the loader.

An image of the loader suitable for programming into flash is generated with the objcopy utility.

```
$ powerpc-eabi-objcopy -O binary executable.elf loader.bin
```

The image file loader.bin is generated.

The FPGA Bitstream

The Virtex-5 FPGA can be configured with a parallel flash. The same flash which holds Linux, the Linux loader and the Linux file system is used for the purpose of configuring the FPGA. This will allow the design to be entirely standalone, eliminating the need to configure the FPGA with impact.

1. Set the ML507 configuration switches so that the FPGA will be configured with BPI_UP configuration 0. SW3 is set to `00001000`.

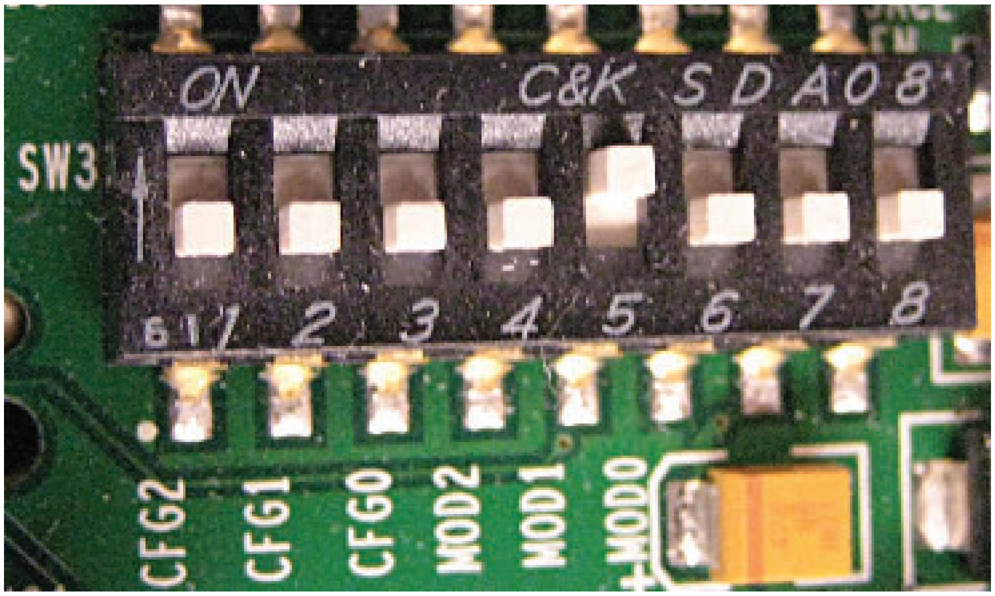


Figure 7: **ML507 SW3 Settings for BPI UP Configuration 0**

2. An image file of suitable format is prepared from the `download.bit` file generated in "Executing the Reference System from XPS for Hardware".

```
$ cd <edk project>/implementation
$ promgen -w -p bin -c FF -o download.bin -u 0 download.bit
```

Note: A previously generated `download.bin` is available in the `ready_for_download/upgrade-image/upgrade.tgz` archive.

Note: The bitstream used must use the Configuration Clock as the Startup Clock. This has already been specified in the EDK project file `etc/bitgen.ut` as shown:

```
-g StartUpClk:CCLK
```

Programming the Flash with Linux

The previously generated `download.bin`, `loader.bin`, and `simpleImage.initrd.virtex440-ml507.elf.bin` files are ready to be programmed into flash at the offsets indicated in "Flash Organization". The Xilinx flashwriter utility could be used for this task, but this application note only discusses using Linux to upgrade the flash. Consult UG111 for information on the Xilinx Flashwriter utility.

Manual Flash Programming

In order to program the flash with new flash images the files must be made available to the running Linux image. There are numerous ways this can be accomplished, such as FTP the files over the network, the System ACE, or USB mass storage. This application note only discusses files on a USB mass storage device and files retrieved over the network.

Copy the files to be upgraded to a USB mass storage device. Connect the mass storage device to the ML507. At this time, the mass storage partition should be visible to Linux:

```
root:/> cat /proc/partitions
major minor  #blocks  name

  31     0       4096 mtdblock0
  31     1       5120 mtdblock1
  31     2      22528 mtdblock2
  31     3        896 mtdblock3
  31     4        128 mtdblock4
   8     0    7872511 sda
```

Note: The display will vary depending on how the mass storage device used has been partitioned.

Mount the mass storage device which appears in the partition list:

```
root:/> mount -t vfat /dev/sda /mnt/usb
```

The files on the mass storage device are now available in `/mnt/usb`.

Erase the Partition

The FPGA bitstream is programmed first. Before programming the new image, it is necessary to erase the appropriate flash region. The first flash partition corresponds to the bitstream:

```
root:/> cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00020000 "bits"
mtd1: 00500000 00020000 "zImage"
mtd2: 01600000 00020000 "rootfs"
mtd3: 000e0000 00020000 "unused"
mtd4: 00020000 00008000 "loader"
```

Erase MTD0:

```
root:/> flash_eraseall /dev/mtd0
Erasing 128 Kibyte @ 3e0000 -- 96 % complete.
```

Program Download.bin into the Flash:

```
root:/> cd /mnt/usb
root:/mnt/usb> cp download.bin /dev/mtd0
```

Unmount the USB mass storage device

```
root:/mnt/usb> cd /
root:/> umount /mnt/usb
```

Automated Flash Upgrade

The script `upgrade.sh` provided with this application note automates the upgrade procedure. It can use upgrade images from either a USB mass storage device or over the network. When executed with no arguments, the script will automatically mount the USB mass storage device. If there is more than one partition on this mass storage device, only the last one is mounted (the user must place their files on this partition). Once mounted, the script will look for a file named `manifest`. If a URL is provided, the manifest file is retrieved over the network from the specified location. The sample manifest provided with this application note is shown below:

```
version: 1.0
tarball: upgrade.tgz
image: mtd0 download.bin
image: mtd1 simpleImage.initrd.virtex440-ml507.elf.bin
image: mtd4 loader.bin
```

The manifest file specifies a version number (1.0). This version coincides with the file /version in the Linux file system:

```
root:/> cat /version
version: 1.0
```

The tarball field indicates which compressed tar file on the USB mass storage device (or on the network server) contains the upgrade images. In this instance, `upgrade.tgz` is used.

The image: fields denote which flash partition is programmed with which image file. The image files are located within the compressed tar image.

Generate the Tarball:

Place all the image files in a subdirectory `images`:

```
$ ls images/
download.bin  loader.bin  simpleImage.initrd.virtex440-ml507.elf.bin
```

Create a compressed tar file from the images:

```
$ cd images
$ tar -czvf ../upgrade.tgz *
download.bin
loader.bin
simpleImage.initrd.virtex440-ml507.elf.bin
```

Note: Previously generated manifest and upgrade.tgz files are provided in the <EDK project>/ready_for_download/upgrade-image/ directory.

Upgrade the Images with USB

Place the tarball and the manifest files on a USB mass storage device at the top level directory.

Connect the USB mass storage device to the ML507 and run the upgrade.sh script.

```
root:/> upgrade.sh
Mounting: sda
Upgrade manifest version 1.0 found
Currently installed version: 1.0
Proceed? (y/n)
y
Extracting: /mnt/usb/upgrade.tgz
Upgrading bitstream
Erasing MTD0
Erasing 128 Kibyte @ 3e0000 -- 96 % complete.
Programming MTD0
Upgrading Linux kernel
Erasing MTD1
Erasing 128 Kibyte @ 4e0000 -- 97 % complete.
Programming MTD1
Upgrading loader
Erasing MTD4
Erasing 32 Kibyte @ 18000 -- 75 % complete.
Programming MTD4
root:/>
```

Power Cycle the ML507.

```
Xilinx Loader:
Flash header at: 0xFE400000
Entry:     0x004008BC
BSS:       0x00708000
BSS Size:  0x0000CDDC
Load Addr: 0x00400000
Load Size: 0x003079E5
```

```
Zero BSS:
Copy text:
Launch:
<kernel boot messages follow>
```

Upgrade the Images Over the Network

The upgrade images can also be fetched over the network. As seen in "Prepare the Device Tree for Linux" a static IP address of 192.168.1.10 is assigned to the ML507.

IP configuration and addressing are beyond the scope of this application note. While more complex configurations are possible, the user should directly connect the ML507 to a FTP server which has been manually configured with the IP address of 192.168.1.1 to successfully perform the tasks outlined in this application note.

The upgrade.sh script will use the wget utility to obtain the manifest and tarball files. Any URL supported by wget (FTP, HTTP) should function.

Place the manifest and tarball files on the FTP server and run the upgrade.sh script on the ML507:

```
root:/> upgrade.sh ftp://192.168.1.1
Network upgrade from ftp://192.168.1.1
Connecting to 192.168.1.1[192.168.1.1]:21
manifest             100% |*****************************|   170    --:--:-
- ETA
Upgrade manifest version 1.0 found
Currently installed version: 1.0
Proceed? (y/n)
y
Connecting to 192.168.1.1[192.168.1.1]:21
upgrade.tgz          100% |*****************************|  3440 KB --:--:-
- ETA
Extracting: /tmp/upgrade.tgz
IMAGES: download.bin simpleImage.initrd.virtex440-ml507.elf.bin
loader.bin

Upgrading bitstream
Erasing MTD0
Erasing 128 Kibyte @ 3e0000 -- 96 % complete.
Programming MTD0

Upgrading Linux kernel
Erasing MTD1
Erasing 128 Kibyte @ 4e0000 -- 97 % complete.
Programming MTD1

Upgrading loader
Erasing MTD4
Erasing 32 Kibyte @ 18000 -- 75 % complete.
Programming MTD4

root:/>
```

Power Cycle the ML507.

```
Xilinx Loader:
Flash header at: 0xFE400000
Entry:      0x004008BC
BSS:        0x00708000
BSS Size:   0x0000CDDC
Load Addr:  0x00400000
Load Size:  0x003079E5
Zero BSS:
Copy text:
Launch:
<kernel boot messages follow>
```

References

1. UG347 ML505/506/507 Evaluation Platform
2. UG111 Embedded System Tools Reference Guide
3. XAPP1107 Getting Started Using Git
4. http://www.denx.de/wiki/DULG/ELDK DENX Embedded Linux Development Kit
5. http://git.xilinx.com Xilinx GIT server and access portal
6. http://xilinx.wikidot.com Xilinx Open Source documentation

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
07/27/09	1.0	Initial Xilinx release.

Notice of Disclaimer