XILINX®

(v1.0) April 1, 2009

Using U-Boot with the Xilinx ML507 Evaluation Platform

Summary

Discusses the use of U-Boot to boot over the network and on-board flash, programming flash images with u-boot, generating JFFS2 flash file systems, and obtaining and building U-Boot. Building Linux kernel images suitable for use with U-Boot is also discussed.

THIS DOCUMENT IS NOT SUPPORTED BY XILINX. DO NOT CONTACT CUSTOMER SUPPORT WITH QUESTIONS REGARDING THIS HOWTO GUIDE.

Included Systems

Included with this howto guide is one reference system built for the Xilinx ML507 Rev A board

Introduction

The Universal Bootloader, formally known as Das U-Boot, is a very capable open source bootloader supporting several embedded platforms. This HOWTO guide is an introduction to using U-Boot on the Xilinx ML507 development board. Topics covered include obtaining Xilinx open source software, including U-Boot and the Linux kernel, building U-Boot, creation of JFFS2 Flash Filesystems, programming the necessary images into flash with the Xilinx FlashWriter utility, booting from flash, booting from the network, and using U-Boot to program flash images.

Target Audience

This HOWTO guide best serves users who are already comfortable with 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 11.1
- Xilinx Integrated Software Environment (ISE®) 11.1
- Xilinx Open Source Linux
- Xilinx Open Source U-Boot
- Suitable PowerPC processor toolchain and Linux Root File System, such as DENX ELDK.
- (optional) GIT revision control software
- (optional) Network cable and host PC for TFTP server functionality

Reference System Specifics

Address Map

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
ppc440mc_ddr2	DDR2_SDRAM	`0x00000000`	`0x0FFFFFFF`
xps_gpio	Push_Buttons_5Bit	`0x81400000`	`0x8140FFFF`
xps_iic	IIC_EEPROM	`0x81600000`	`0x8160FFFF`
xps_intc	xps_intc_0	`0x81800000`	`0x8180FFFF`
xps_ll_temac	Hard_Ethernet_MAC	`0x81C00000`	`0x81C0FFFF`
xps_sysace	SysACE_CompactFlash	`0x83600000`	`0x8360FFFF`
xps_uart16550	RS232_Uart_2	`0x83E00000`	`0x83E0FFFF`
xps_uart16550	RS232_Uart_1	`0x83E20000`	`0x83E2FFFF`
xps_mch_emc	FLASH	`0xFE000000`	`0xFFFFFFFF`

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 impact-batch.cmd
   ```
3. Proceed to the "Program the Bootloader into Flash with Xilinx FlashWriter" section, using the U-Boot image provided in the `ready_for_download` 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. Proceed to the "Program the Bootloader into Flash with Xilinx FlashWriter" section, using the U-Boot image provided in the `ready_for_download` area.

Obtaining the Software

The user will need to obtain source code for U-Boot, the Linux kernel, and the Linux kernel BSP generator in order to complete the tasks discussed in this HOWTO guide. These are all 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 HOWTO guide; 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 HOWTO guide. 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 09326d64
```

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 HOWTO guide. This version has been demonstrated to work as described in this document without modification. Perform after cloning the tree.

```
$ cd device-tree
$ git checkout 8fbfa99e
```

3. Obtain the latest u-boot

```
$ cd <project area>
$ git clone git://git.xilinx.com/u-boot-xlnx.git
```

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

```
$ cd u-boot-xlnx
$ git checkout 9abed00d
```

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 HOWTO guide can be obtained with the following links:

device-tree

linux-2.6-xlnx

u-boot

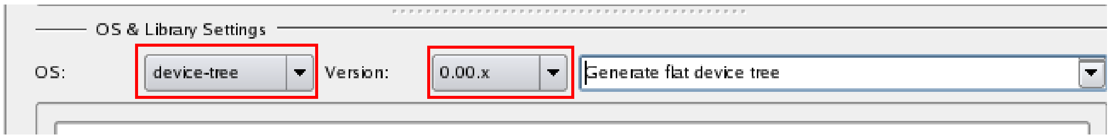
Note: In the future these direct links may not be available, and the user may need to navigate to the desired snapshot directly from the git.xilinx.com page.

Obtaining a toolchain

To build any of the software used in this HOWTO guide, the user will require a an appropriate PowerPC 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 HOWTO guide utilizes the ELDK, which can be found at http://www.denx.de/wiki/DULG/ELDK. Toolchain installation is beyond the scope of this HOWTO guide.

Generate the BSP

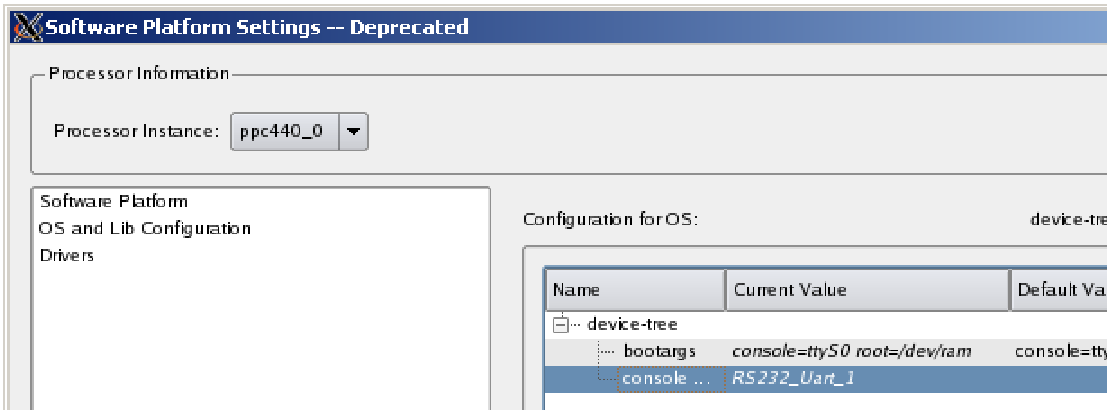
Open the EDK project in XPS. Choose Software → Software Platform Settings. Choose **device-tree** in the OS & Library Settings list box. Select version **0.00.x**.



OS & Library Settings
OS: device-tree
Version: 0.00.x
Generate flat device tree

Figure 1: **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-tre

Name	Current Value	Default Va
device-tree		
bootargs	console=ttyS0 root=/dev/ram	console=tt
console ...	RS232_Uart_1	

Figure 2: **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`

Copy `<edk system>/ppc440_0/include/xparameters.h` to `<project area>/u-boot-xlnx/board/xilinx/ml507/xparameters.h`

Edit the file `<project area>/u-boot-xlnx/board/xilinx/ml507/xparameters.h` to match the changes shown in **red**:

```
#define XPAR_CPU_PPC440_CORE_CLOCK_FREQ_HZ 400000000
#define XPAR_CORE_CLOCK_FREQ_HZ XPAR_CPU_PPC440_CORE_CLOCK_FREQ_HZ
```

Build U-Boot

Modify the u-boot source

The user must use the correct `xparameters.h` file which accurately describes the hardware u-boot will run on. This is described in the "Generate the BSP" section.

The source provided will generate a u-boot image intended to be run from RAM. This is useful for testing u-boot, but is not suitable to real world deployments. The configuration is edited so that the u-boot image will run from the on board flash.

Edit `include/configs/ml507.h` and make the changes or additions shown in **red**:

```
#define CONFIG_PPC              1
#define CONFIG_CMDLINE_TAG      1

#if ! (defined(CFG_ENV_IS_IN_FLASH) || defined(CFG_ENV_IS_IN_EEPROM))
#define CFG_ENV_IS_NOWHERE      1       /* no space to store environment */
```

```
#define CFG_ENV_SIZE            256


#define CFG_MONITOR_BASE       0xFFFC0000

#endif

/* following are used only if env is in EEPROM */
#ifdef  CFG_ENV_IS_IN_EEPROM
#define CFG_I2C_EEPROM_ADDR             (0xA0 >> 1)
#define CFG_I2C_EEPROM_ADDR_LEN         1
#define CFG_I2C_EEPROM_ADDR_OVERFLOW    0x3
#define CFD_I2C_EEPROM_SIZE             8192
#define CFG_ENV_OFFSET                  256
#define CFG_ENV_SIZE                    1024
#define CFG_EEPROM_PAGE_WRITE_BITS      4
#define CFG_EEPROM_PAGE_WRITE_DELAY_MS  5
#define CONFIG_ENV_OVERWRITE          1  /* writable ethaddr and serial# */

#define CFG_MONITOR_BASE       0xFFFC0000
```

Edit `board/xilinx/ml507/config.mk` and make the changes shown in red:

```
TEXT_BASE = 0xFFFC0000
#TEXT_BASE = 0x02000000
```

Build a U-Boot image

Indicate which toolchain is to be used. This below will work with a properly installed ELDK. In this case, `ppc_4xx-gcc`, etc... will be available in the user's PATH.

```
$ export CROSS_COMPILE ppc_4xx-
```

Choose to use an in-flash configuration

```
$ make ml507_flash_config
```

Build the u-boot image

```
$ make ARCH=ppc
```

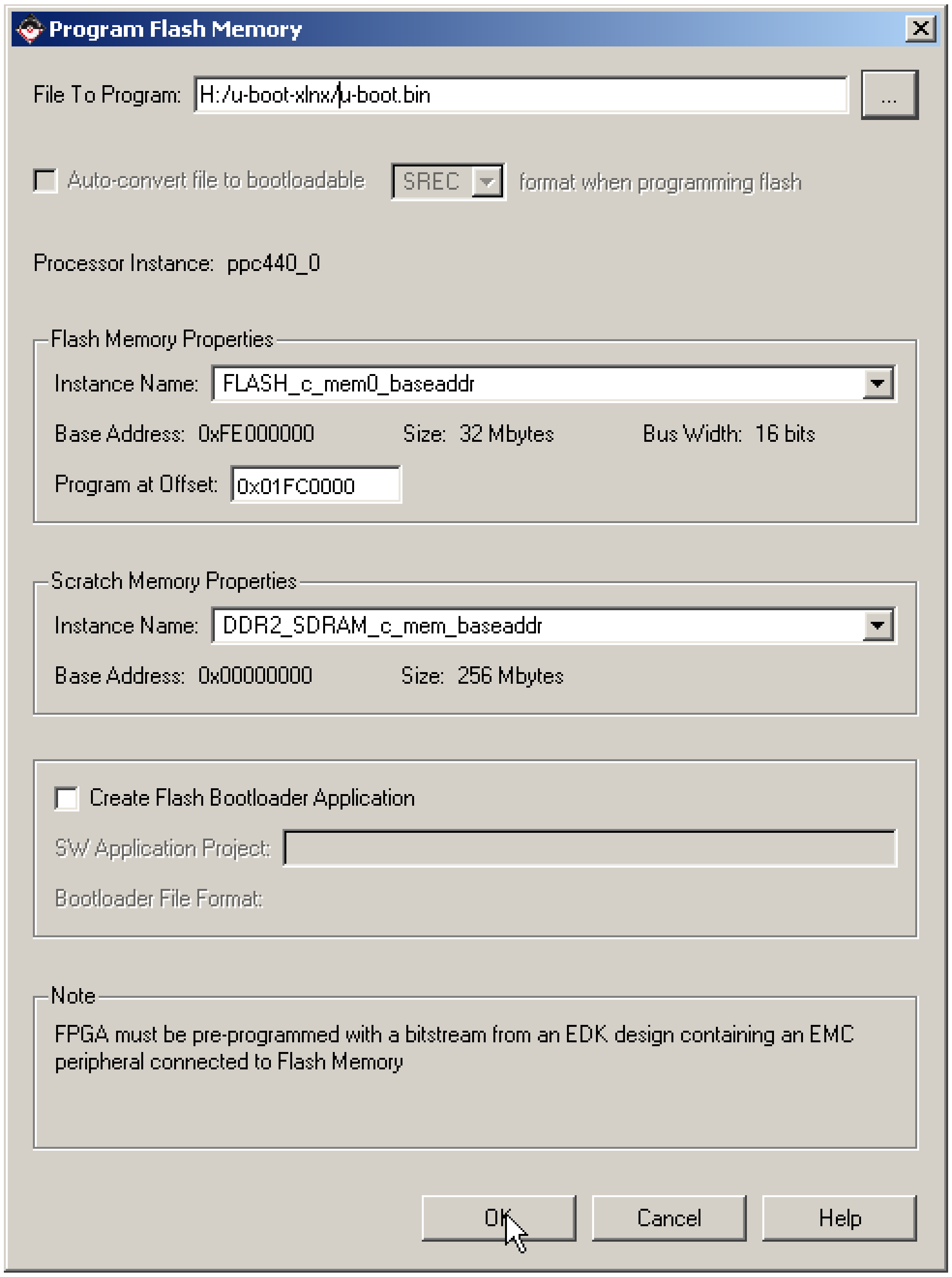
The image `u-boot.bin` is created.

Program the Bootloader into Flash with Xilinx FlashWriter

To use the u-boot image it must be programmed into the appropriate location in flash. This is performed with the Xilinx FlashWriter utility.

Open the hardware project with XPS. Choose **Device Configuration** → **Program Flash Memory**. For the **File To Program** field choose the `u-boot.bin` file which was created in the "Build U-Boot" section. Enter `0x01FC0000` in the Program at Offset field and click 'OK'. See Figure 3.

Note: A previously generate image file is available in the `ready_for_download` area.



Program Flash Memory

File To Program: H:/u-boot-xlnx/u-boot.bin

Auto-convert file to bootloadable SREC format when programming flash

Processor Instance: ppc440_0

Flash Memory Properties

Instance Name: FLASH_c_mem0_baseaddr

Base Address: 0xFE000000 Size: 32 Mbytes Bus Width: 16 bits

Program at Offset: 0x01FC0000

Scratch Memory Properties

Instance Name: DDR2_SDRAM_c_mem_baseaddr

Base Address: 0x00000000 Size: 256 Mbytes

Create Flash Bootloader Application

SW Application Project:

Bootloader File Format:

Note

FPGA must be pre-programmed with a bitstream from an EDK design containing an EMC peripheral connected to Flash Memory

Figure 3: **Program u-boot into flash with Flash Writer**

When completed, the u-boot image will be programmed at the very end of flash memory, where the processor boot vector is located. Press the 'CPU RST' button. The u-boot prompt **=>** is displayed.

Build the Linux kernel

U-Boot may boot a Linux system in a wide variety of ways. Initially, booting from flash will be discussed.

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

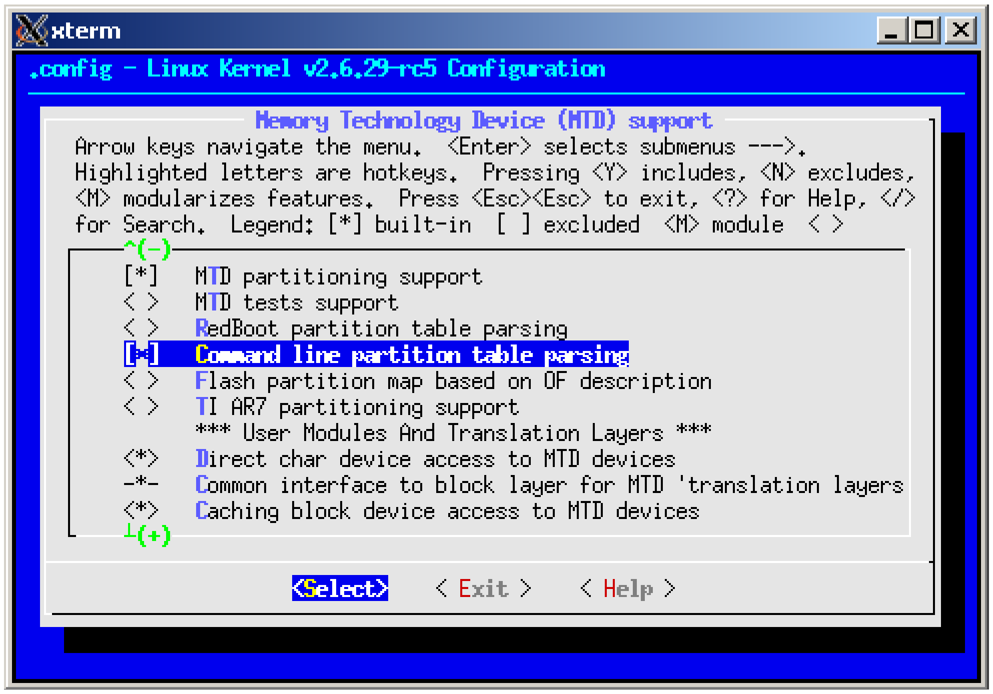
```
$ cp arch/powerpc/configs/44x/virtex5_defconfig .config
```

Build and run the kernel menu config application

```
make ARCH=powerpc menuconfig
```

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**

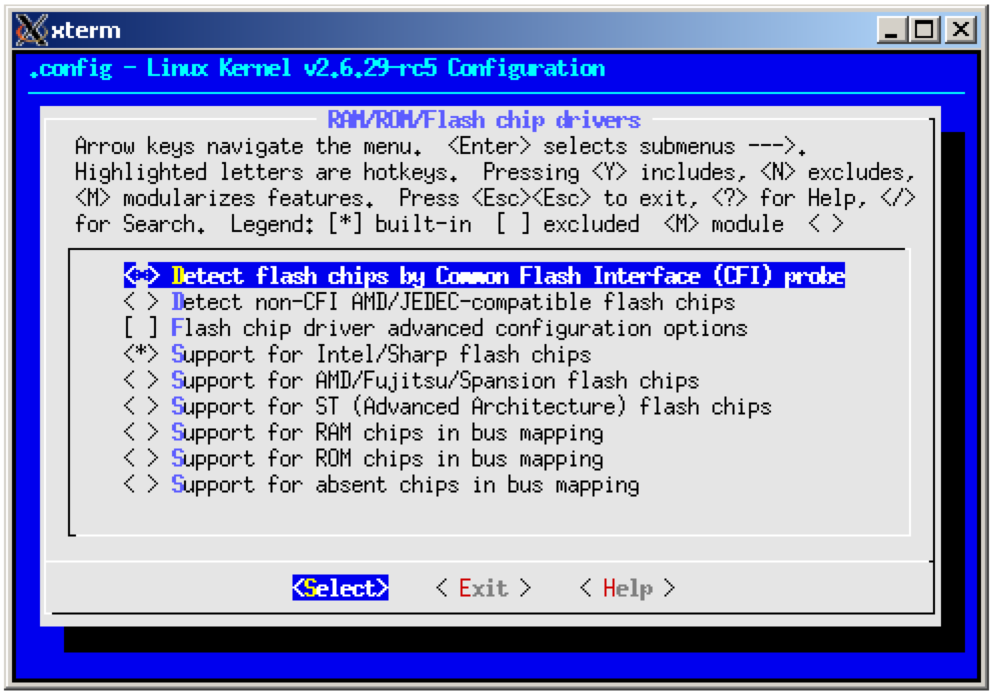


```
.config - Linux Kernel v2.6.29-rc5 Configuration
              Memory Technology Device (MTD) support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> 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 >
```

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**



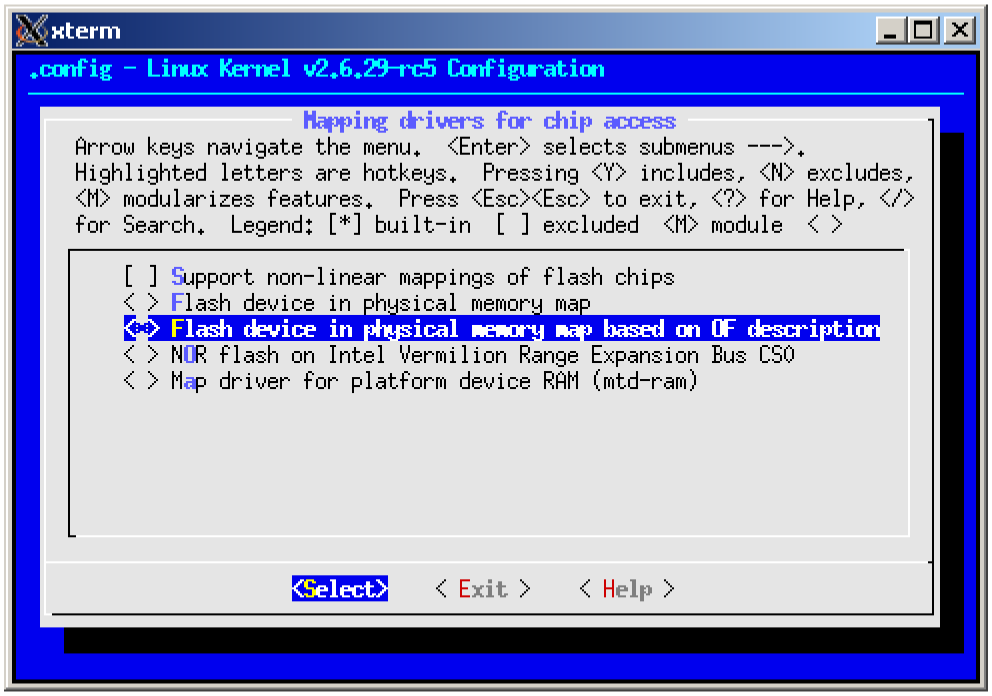
```
.config - Linux Kernel v2.6.29-rc5 Configuration
                    RAM/ROM/Flash chip drivers
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  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
      <*> Support for Intel/Sharp flash chips
      < > Support for AMD/Fujitsu/Spansion flash chips
      < > Support for ST (Advanced Architecture) flash chips
      < > Support for RAM chips in bus mapping
      < > Support for ROM chips in bus mapping
      < > Support for absent chips in bus mapping

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

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-rc5 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
      <*> 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)

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

Figure 6: **MTD mapping driver**

5. Enable **File Systems** → **Miscellaneous filesystems** → **Journalling Flash File System v2**

6. Exit and save the configuration.

Compile an image suitable for use with u-boot:

```
make ARCH=powerpc uImage
```

Note: A prebuilt image `uImage` is provided in the `ready_for_download` area.

The new image is created in `linux-2.6-xlnx/arch/powerpc/boot/uImage.`

Note: Building a `uImage` requires that the `mkimage` application be found in the users PATH. If the user does not have `mkimage` installed elsewhere, an executable is built with U-Boot. This executable can be found at `<project area>/u-boot-xlnx/tools/mkimage`.

Create a JFFS2 flash filesystem

A JFFS2 flash file system is used. JFFS2 images are created with the `mkfs.jffs2` command. The user must have this command installed locally to complete this procedure. Most Linux distributions will already have this application available. For additional information the user may look at http://sources.redhat.com/jffs2

A previously generated filesystem image `jffs2-image.bin` is provided in the `ready_for_download` area.

A root filesystem `ramdisk.image.gz` is provided in the `ready_for_download` area. This file is a compressed image of a native Linux EXT2 filesystem. The Linux `mkfs.jffs2` command will copy a directory structure from the local filesystem into a JFFS2 image file. The provided ramdisk image is mounted and used as source material for the JFFS2 filesystem.

Note: `mkfs.jffs2` is not provided with the Xilinx EDK and must be installed separately by the user. It is commonly available for all major Linux distributions.

Uncompress the image in a temporary location

```
# cd <edk project>/ready_for_download
# mkdir tmp
# cp ramdisk.image.gz tmp
# gunzip ramdisk.image.gz
```

Mount the EXT2 filesystem image into the local filesystem. The user must have root privileges to perform this task.

```
# mkdir ext2image
# mount -o loop ramdisk.image ext2image
```

Create a JFFS2 filesystem image from the files in the `ext2image` directory

```
# mkfs.jffs2 --big-endian --eraseblock=131072 --root=ext2image -o jffs2-
image.bin
```

The JFFS2 image <edk project>/ready_for_download/tmp/jffs2-image.bin is created. Unmount the ext2 filesystem.

```
# umount ext2image
```

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 with U-Boot. Table 2 shows the division chosen in this HOWTO guide.

Table 2: **Flash partitions**

	Start Address	Offset	Size
FPGA Bitstream	0xFE000000	0x00000000	0x00400000 (4M)
Linux Kernel	0xFE400000	0x00400000	0x00200000 (2M)
Root Filesystem	0xFE600000	0x00600000	0x01900000 (25M)
device tree	0xFFF00000	0x01F00000	0x00020000 (128K)
(unused)	0xFFF20000	0x01F20000	0x000A0000 (640K)
U-Boot	0xFFFC0000	0x01FC0000	0x00040000 (256K)

U-Boot will use images at any flash location and does not require that the user create any logical structure describing the flash organization. Linux, however, 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 U-Boot" section.

Prepare the device tree for U-Boot

The steps in "Generate the BSP" created a text file known as a device tree, which the user then copied to the appropriate kernel tree location. This file describes the hardware system on which Linux will run. U-Boot will provide the location of a device tree in memory to the Linux kernel upon boot. One of the entries within this file is the Linux bootargs. These are parameters passed to the Linux kernel which direct startup configuration and behavior.

Modify the kernel command line to use the flash

The previously generated `linux-2.6-xlnx/arch/powerpc/boot/dts/virtex440-ml507.dts` is edited so that the kernel command line reflects the text shown in **red**.

```
chosen {
        bootargs = "console=ttyS0 rootfstype=jffs2 root=/dev/mtdblock2 rw
mtdparts=fe000000.flash:4M(bits)ro,2M(kernel)ro,25M(rootfs),128k(dev-
tree),640K(unused),256K(uboot)";
        linux,stdout-path = "/plb@0/serial@83e20000";
    } ;
```

The bootargs, which are entered all as **one single line**, indicate the following:

1. The first serial port, `ttyS0`, is to be used as the console
2. The root filesystem is on the third partition of the flash device. This is referenced as `mtdblock2`.
3. The flash partitions are defined with `mtdparts`. This mapping implements that specified in "Flash Organization".

The flash device name `fe000000.flash` is determined by the presence of the xps-mch-emc peripheral in the `virtex440-ml507.dts` device-tree system description.

```
FLASH: flash@fe000000 {
            compatible = "xlnx,xps-mch-emc-3.00.a", "cfi-flash";
            reg = < 0xfe000000 0x2000000 >;
```

The use of this cfi-flash device was enabled by the user with the "Enable Flash device in physical memory map based on OF description" kernel configuration option.

Compile the device tree

The device tree is maintained as a human readable text file. The kernel expects the device tree in binary form. The device tree compiler is used to create a binary file

```
$ cd <project area>/linux-2.6-xlnx
$ arch/powerpc/boot/dtc -b 0 -V 17 -R 4 -S 0x3000 -I dts -O dtb -o ml507-
jffs2.dtb -f arch/powerpc/boot/dts/virtex440-ml507.dts
```

The file `<project area>/linux-2.6-xlnx/ml507-jffs2.dtb` is created.

Note: The executable `arch/powerpc/boot/dtc` was created in "Build the Linux kernel".

Program Linux into flash with FlashWriter

The user has generated a Linux kernel, a JFFS2 root file system, and a device tree. These items are now programmed into the chosen locations within the flash using Xilinx FlashWriter. The user will follow the same procedure which was used to program u-boot in "Program the Bootloader into Flash with Xilinx FlashWriter".

Note: Prebuilt images are supplied in the ready_for_download area which may be utilized if the user has not generated these images as instructed in this HOWTO guide.

1. Program `<project area>/linux-2.6-xlnx/arch/powerpc/boot/uImage` at offset `0x00400000`.
2. Program `<EDK Project>/ready_for_download/tmp/jffs2-image.bin` at offset `0x00600000`.
3. Program `<project area>/linux-2.6-xlnx/ml507-jffs2.dtb` at offset `0x01F00000`
4. If the user has not already completed the steps in "Program the Bootloader into Flash with Xilinx FlashWriter" program `<project area>/u-boot-xlnx/u-boot.bin` at offset `0x01FC0000` at this time.

At this time, the user should press the "CPU RST" button on the board. The u-boot prompt is displayed.

=>

Configure the U-Boot environment

U-Boot is very flexible. It can boot the target from a wide variety of sources with several different configurations. The user configures U-Boot to boot the already programmed in-flash Linux at this time.

U-Boot configuration is maintained in its environment. U-Boot should have the default environment at this time. The U-Boot image built in this HOWTO guide is configured to use the on board IIC EEPROM as nonvolatile storage of the U-Boot environment.

The **printenv** command will display the present environment

```
=> printenv
baudrate=9600
loads_echo=1
ethaddr=00:0A:35:01:CF:70
stdin=serial
stdout=serial
stderr=serial
...
```

The **bootm** command will boot Linux from an already in-memory image. The onboard NOR flash is CPU addressable; the images programmed are in memory which U-Boot can boot from directly.

The user manually boots Linux with the **bootm** command, specifying the address of the uImage and device tree in memory.

```
=> bootm 0xFE400000 - 0xFFF00000
```

After a few moments, the Linux command prompt appears. Reboot the system to return to the U-Boot prompt.

```
root:~> reboot
```

The **bootcmd** environment variable is set to contain whatever U-Boot commands are necessary to boot the desired image.

```
=> setenv bootcmd 'bootm 0xFE400000 - 0xFFF00000'
```

The environment is saved in nonvolatile storage

```
=> saveenv
```

Press the ‘CPU RST’ button on the ML507. Linux is automatically booted from flash.

Booting over the network

U-Boot can fetch images over the network using TFTP. IP configuration and TFTP server administration are beyond the scope of this HOWTO guide. All material provided in this section pertaining to configuration of the TFTP server is intended as a quick reference only. Users which are unfamiliar with these concepts may need additional reference material.

Configure the TFTP server

Linux

Typical Linux server installations will provide an already configured TFTP server. Files placed in the `/tftpboot` directory are available to TFTP clients.

Note: Obtaining, installing, and configuring a Linux TFTP server is beyond the scope of this HOWTO guide. If the user wishes to use a Linux TFTP server they should consult the pertinent documentation. **All further examples regarding TFTP boot in this HOWTO guide refer to Microsoft Windows.**

Microsoft Windows

Windows has no traditional TFTP server. TFTPD32, which is freely available from http://tftpd32.jounin.net is used in this HOWTO guide. It requires no installation and can be run directly from the directory in which it has been downloaded.

Statically configure the Host PC IP address to `192.168.0.1`. Existing version of Windows are similar but differ in the exact steps to perform this task. Consult the applicable Microsoft documentation.

Run the tftpd32 executable. A window similar to Figure 7 appears.

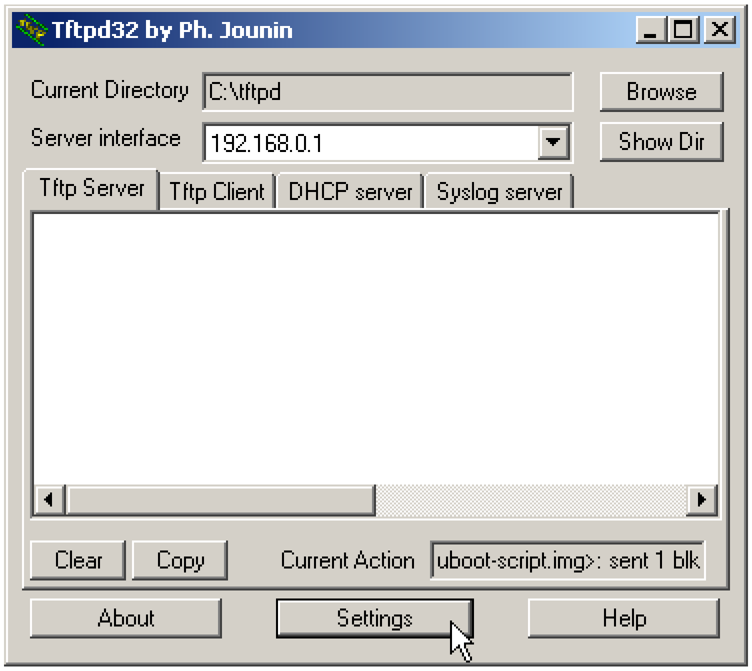
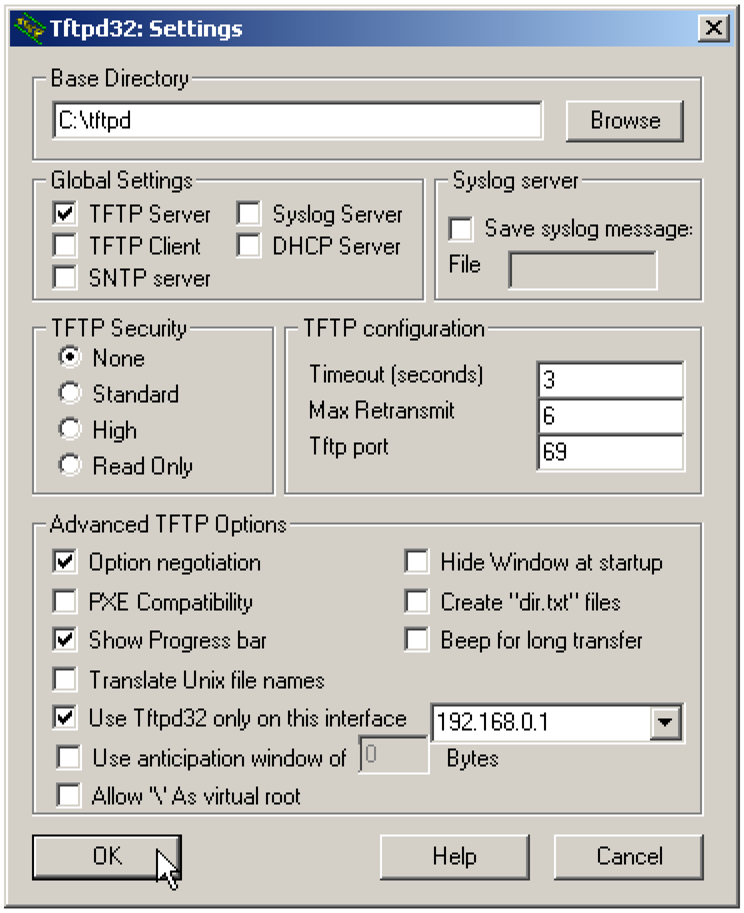


Figure 7: **TFTPD32**

Click **Settings** to configure the TFTP server. The user should choose a Base Directory (the location of files the server makes available to TFTP clients). `C:\tftpd` is used. The user should ONLY enable the TFTP Server. Enabling other services (especially DHCP) may adversely affect the network the user is connected to and should only be done by knowledgeable users. See Figure 8.



Tftpd32: Settings

Base Directory: C:\tftpd [Browse]

Global Settings
- [✓] TFTP Server
- [ ] Syslog Server
- [ ] TFTP Client
- [ ] DHCP Server
- [ ] SNTP server

Syslog server
- [ ] Save syslog message:
- File:

TFTP Security
- (●) None
- ( ) Standard
- ( ) High
- ( ) Read Only

TFTP configuration
- Timeout (seconds): 3
- Max Retransmit: 6
- Tftp port: 69

Advanced TFTP Options
- [✓] Option negotiation
- [ ] Hide Window at startup
- [ ] PXE Compatibility
- [ ] Create "dir.txt" files
- [✓] Show Progress bar
- [ ] Beep for long transfer
- [ ] Translate Unix file names
- [✓] Use Tftpd32 only on this interface: 192.168.0.1
- [ ] Use anticipation window of 0 Bytes
- [ ] Allow '\' As virtual root

Figure 8: **TFTPD32 settings**

Click OK. Restart TFTPD32. At this time, any files present in `C:\tftpd` may be fetched from the server by TFTP clients.

Connect the ML507 to the server network. They may either be directly cabled together or connected through an ethernet switch.

Prepare the ramdisk image for use with U-Boot

The ramdisk image provided with this HOWTO guide, `ramdisk.image.gz` is found in the `ready_for_download` area. It is a compressed EXT2 filesystem. This image must be prepared for use with U-Boot.

```
$ cd <EDK Project>
$ mkimage -A ppc -O linux -T ramdisk -C gzip -d
ready_for_download/ramdisk.image.gz uramdisk.image
```

The file `<EDK Project>/uramdisk.image` is created. If the user does not have `mkimage` installed elsewhere, an executable is built with U-Boot. This executable can be found at `<project area>/u-boot-xlnx/tools/mkimage`.

Configure the U-Boot environment

Assign a static IP address to the target.

```
=> setenv ipaddr 192.168.0.5
```

Assign a default server address.

```
=> setenv serverip 192.168.0.1
```

Verify connectivity by pinging the server.

```
=> ping 192.168.0.1
host 192.168.0.1 is alive
```

Save this configuration

```
=> saveenv
```

Modify the kernel command line to use a ramdisk

The previously generated `linux-2.6-xlnx/arch/powerpc/boot/dts/virtex440-ml507.dts` is edited so that the kernel command line reflects the text shown in **red**.

```
chosen {
        bootargs = "console=ttyS0 root=/dev/ram rw
mtdparts=fe000000.flash:4M(bits)ro,2M(kernel)ro,25M(rootfs),128k(dev-
tree),640K(unused),256K(uboot)";
        linux,stdout-path = "/plb@0/serial@83e20000";
    } ;
```

The bootargs, which are entered all as **one single line**, indicate the following:

1. The first serial port, `ttyS0`, is to be used as the console
2. The root filesystem is a ramdisk
3. The flash partitions are defined with `mtdparts`. This mapping reflects that specified in "Flash Organization"

Compile the device tree

The device tree is maintained as a human readable text file. The kernel expect the device tree in binary form. The device tree compiler is used to create a binary file

```
$ cd <project area>/linux-2.6-xlnx
$ arch/powerpc/boot/dtc -b 0 -V 17 -R 4 -S 0x3000 -I dts -O dtb -o ml507-
ramdisk.dtb -f arch/powerpc/boot/dts/virtex440-ml507.dts
```

The file `<project area>/linux-2.6-xlnx/ml507-ramdisk.dtb` is created.

Note: The executable `arch/powerpc/boot/dtc` was created in "Build the Linux kernel".

Fetch and boot Linux

At this time, the user will copy the `uImage`, `uramdisk.image`, and `ml507-ramdisk.dtb` images to the `C:\tftpd` directory.

Fetch the images with TFTP, specifying the image to fetch and the memory location to store it.

```
=> tftp 0x1000000 ml507-ramdisk.dtb
=> tftp 0x1C00000 uImage
=> tftp 0x1800000 uramdisk.image
```

Boot the images which are now in memory

```
=> bootm 0x1c00000 0x1800000 0x1000000
```

Configure U-Boot to autoboot over the network

As seen in "Configure the U-Boot environment", the bootcmd environment variable can be set to contain U-Boot commands to automatically boot the board at startup.

Configure U-Boot to autoboot over the network:

```
=> setenv bootcmd 'tftp 0x1000000 ml507-ramdisk.dtb; tftp 0x1C00000 uImage;
tftp 0x1800000 uramdisk.image; bootm 0x1c00000 0x1800000 0x1000000'
=> saveenv
```

Boot the board

```
=> boot
```

Programming the flash with U-Boot

With the TFTP server and U-Boot configured as shown in "Booting over the network" the user can use U-Boot to fetch images over the network and program them into flash.

Copy the flash file system image `jffs2-image.bin` to `C:\tftpd`. Direct U-Boot to fetch the flash file system image with TFTP.

```
=> tftp 0x1000000 jffs2-image.bin
```

Turn off write protection for the sectors to be programmed.

```
=> protect off FE600000 +1900000
```

Erase the flash area which will be programmed with the image

```
=> erase FE600000 +${filesize}
```

Program the flash with the in-memory image

```
=> cp.b 0x1000000 0xFE600000 ${filesize}
```

U-Boot Scripts

U-Boot can fetch script files containing U-Boot commands and execute them. The user begins with an ordinary text file which contains U-Boot commands.

EXAMPLE: Create the text file `upgrade-all.txt`

```
echo 'Upgrading JFFS2 image'
tftp 0x1100000 jffs2-image.bin
protect off FE600000 +1900000
erase FE600000 +${filesize}
cp.b 0x1100000 0xFE600000 ${filesize}
echo 'Upgrading device tree'
tftp 0x1100000 ml507-jffs2.dtb
protect off FFF00000 +20000
erase FFF00000 +${filesize}
cp.b 0x1100000 0xFFF00000 ${filesize}
echo 'Upgrading Linux Kernel'
tftp 0x1100000 uImage
protect off FE400000 +200000
erase FE400000 +${filesize}
cp.b 0x1100000 0xFE400000 ${filesize
```

Convert the script file to a U-Boot image file

```
$ mkimage -A ppc -O linux -T script -C none -a 0 -e 0 -n "Upgrade script"
-d upgrade-all.txt upgrade-all.img
```

The file `upgrade-all.img` is created. Copy this file to the `C:\tftpd` directory.

Copy the script image into memory

```
=> tftp 0x1000000 upgrade-all.img
```

Execute the script with the autoscr command

```
=> autoscr 0x1000000
```

The Linux kernel, device tree, and root file system are fetched from the TFTP server and programmed into flash.

Running U-Boot commands in environment variables

The proceeding commands to upgrade the flash

```
=> tftp 0x1000000 upgrade-all.img
=> autoscr 0x1000000
```

can be simplified by placing them in an environment variable

```
=> setenv update-all 'tftp 0x1000000 upgrade-all.img; autoscr 0x1000000'
=> saveenv
```

From now on, the flash images can be updated using the latest update script by running this environment command

```
=> run update-all
```

Program the FPGA bitstream with U-Boot

The Virtex 5 FPGA can be configured with a parallel flash. The same flash which holds U-Boot, Linux, 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`.

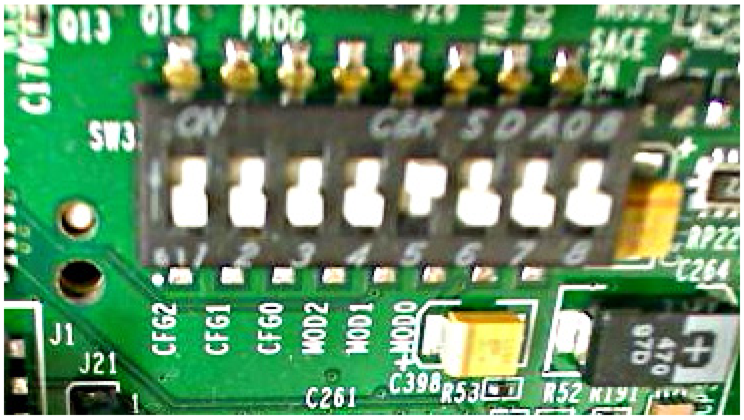


Figure 9: **ML507 SW3 Settings for BPI UP configuration 0**

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.bit` and `download.bin` are available in the `ready_for_download` area.

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
```

3. Copy the generated `download.bin` to the TFTP directory `C:\tftpd`
4. Program the image with U-Boot

```
=> tftp 0x1000000 download.bin
=> protect off FE000000 +${filesize}
=> erase FE000000 +${filesize}
=> cp.b 1000000 FE000000 ${filesize}
```

5. Press the ‘PROG’ button. The FPGA is configured from flash, and U-Boot is executed.

References

1. UG347 ML505/506/507 Evaluation Platform
2. XAPP1107 Getting Started Using Git
3. http://www.denx.de/wiki/DULG/ELDK DENX Embedded Linux Development Kit
4. http://git.xilinx.com Xilinx GIT server and access portal
5. http://xilinx.wikidot.com Xilinx Open Source documentation
6. http://tftpd32.jounin.net TFTPD32 Open Source TFTP server for Windows

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/01/09	1.0	Initial Xilinx release.

Notice of Disclaimer