**Freescale Semiconductor**
Application Note

AN2924
Rev. 1.0, 09/27/2005

# U-Boot for HPC II

*by* *Srikanth Srinivasan,*
*Maurie Ommerman*
*CPD Applications, Freescale Semiconductor, Inc.*

*risc10@email.sps.mot.com*

This application note describes the process of obtaining U-Boot source code and building a binary that runs on Freescale’s High Performance Platform II (HPC II, also referred to as Taiga).

**Contents**

## 1 Introduction

HPC II is a high-performance PowerPC-based platform featuring Freescale’s MPC7448 microprocessor with Tundra Semiconductor’s Tsi108 bridge chip. The board bring-up tasks have been successfully completed with U-Boot, a common open source firmware and boot-loader (sourceforge.net/projects/u-boot), and Linux kernel (2.6.11), with both RAM disk and network-mounted root file systems, running on the target platform. A version of U-Boot has been ported for the HPC II and is available as part of the Linux BSP from the HPC II product summary page found in the Freescale web site. It is also possible to obtain code from the top of the CVS tree in SourceForge and enhance it to support the board-specific features. Steps to implement the latter have been listed and explained in Section 4, “U-Boot”, of this paper. The paper has been organized as follows—terminology used in the rest of the application note, a brief description of the hardware, obtaining and building the source, downloading the firmware to the platform, using U-Boot on HPC II, and steps for porting U-Boot to the HPC II target platform.

## 2 Terminology

The following terms are used in this document:

| | |
|---|---|
| U-Boot | Universal Boot Loader—an open-source firmware project providing code under GPL |
| HPC II | Freescale's High Performance Platform II featuring MPC7448 and Tsi108 |
| Linux OS | Linux Operating System |
| Yellow Dog | A distribution of Linux |
| Debian | A distribution of Linux |
| Host | Computer used for compiling and linking target programs |
| TFTP | Trivial file transfer protocol—a method of sending files across the ethernet interface |
| PVR | Processor version release—a read only register which identifies the processor |
| OCN | OnChipNetwork fabric switch—Pronounced OCeaN |
| PCI | Peripheral Component Interconnect—a common bus architecture for add-on chips and cards |
| MB | Mega bytes—1 million bytes |
| GB | Giga bytes—1 billion bytes |
| RTC | Real-Time Clock |
| NVRAM | Non-Volatile RAM, battery-backed, usually used for storage of environment variables |

## 3 Hardware and Software

The target hardware includes a HPC II board. For details on the platform itself, refer to the MCEVALHPC2-7448 documentation on the Freescale web page.

The host equipment includes software and hardware for code development, download, and execution, as well as tools for debugging. For code development, it is advisable to use PowerPC platforms such as a Genesi Pegasos machine with Debian or Yellow Dog distribution of Linux. However, an x86 Linux machine with a cross-compiler for PowerPC will work just as well.

PROMjet is a flash emulator used for quick download and test of software (these can be done away with once the ability to program the on-board flash in U-Boot exists). HPC II provides PROMjet ports (two 16-bit ports) for use when the firmware is not fully functional with on-board flash operations. The emulator is controlled using software on x86 Windows and, therefore, needs a Windows-based PC. The firmware displays messages through serial ports on the platform. A serial communication program, such as HyperTerminal, mincom, or Smartcom, is employed to facilitate software debugging.

Usually, until flash erase and program functions are supported in U-Boot, the PROMjets and a serial port monitor are used on a Windows machine. After that, development can be continued on a Linux machine.

An integral part of U-Boot development is the use of COP-based tools. Powerscan, an internal COP debugger (not available to customers), along with the supporting hardware, was the debugging tool of choice for the authors, mostly out of familiarity. Customers can use similar tools, such as PowerTap and BDI2000, to serve the same purpose.

# 4 U-Boot

## 4.1 What is U-Boot?

U-Boot is an open source firmware for hardware platforms. It supports many processor architectures, boards, and network drivers. It is a popular firmware for bring-up of PowerPC processor based platforms.

U-Boot is widely used in embedded space platforms. It is similar to DINK32, Open Firmware, and x86 bios, in that it is the first code run at startup.

## 4.2 What Does It Do?

U-Boot configures different hardware blocks on the board and brings them out of reset into a sane state. It can load and start an OS automatically (auto-boot) or, alternatively, it allows the user to run commands to start the OS. The subset of default commands that are part of U-Boot also provide capability to the user to perform memory, network, flash operations, and more prior to OS boot-up.

## 4.3 What Does It Not Do?

It is not designed for debugging (setting breakpoints, stepping through the user code, etc.).

## 4.4 Origins

U-Boot has its origins in PPCBoot. It was started by Wolfgang Denk of DENX Software Engineering. More information and free distributions are available at http://www.denx.de/twiki/bin/view/DULG/WebHome.

## 4.5 This U-Boot Version's Origins

Freescale supports an internal SourceForge repository, which is populated from the world wide web repository for relevant projects. U-Boot is obtained from the Master U-Boot repository on the public web, http://sourceforge.net/projects/u-boot.

The Freescale internal SourceForge repository maintains current versions in sync with external SourceForge. It is only available for Freescale employees. Contact risc10@freescale.com for current U-Boot source code for Freescale platforms that do not have board support packages (BSP) and/or have not been sent out to open source.

Customers can get U-Boot source from the Metrowerks web site when the BSP for the HPC II board is available (3Q2005). U-Boot will be available as part of the BSP software.

This version of U-Boot was derived from a version based on external SourceForge and modified for an evaluation platform built by Tundra Semiconductor, Inc.

## 4.6 U-Boot Control Sequence

When power is applied to the platform, the U-Boot code begins executing and performing the following operations:

1. Power-on/reset
2. Reset vector points to FLASH (0xFFF00100)
3. Begins execution in FLASH

4. Relocates itself to RAM
5. Continues execution in RAM
6. Initializes higher level devices
7. Begins command interpreter

## 5 Source Code

The source tree for U-Boot is similar to the source tree for Linux. The structure and some of the files are organized in a way similar to those in Linux. Like the Linux kernel, U-Boot is subject to the GPL license.

Figure 1 shows the U-Boot directory structure.

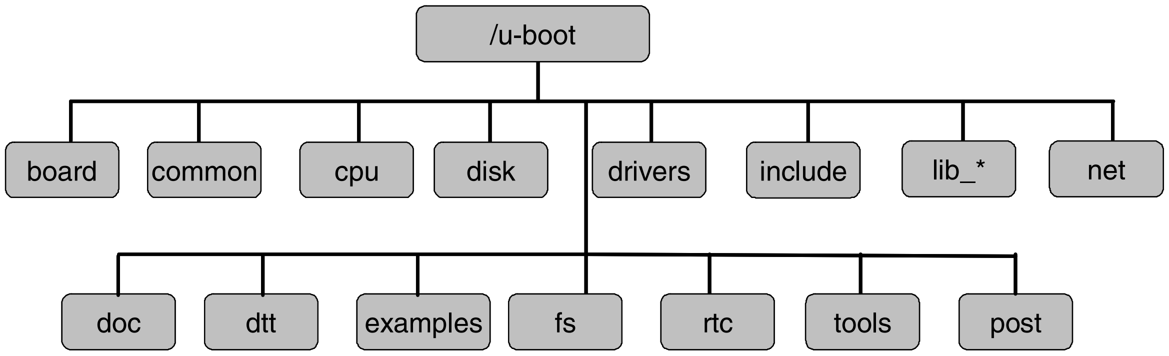


**Figure 1. U-Boot Directories**

Following is a short description of each directory:

| | |
|---|---|
| board | contains platform, board-level files; i.e. Sandpoint, cds, Marvell. The board directory contains all the specific board initialization functions, which are called from lib_ppc/board.c. |
| board/Freescale/freeserve2 | contains specific board info for HPC II; i.e. serial, asm_init.S config.mk, flash.c, pci.c, tsi108.[hc], ns1650.[hc], etc. |
| common | contains all the commands; i.e. cmd_boot.c, cmd_date.c,environment, env.c, main.c, etc. |
| cpu | contains all the cpu specific information; i.e. 74xx_7xx, mpc824xcpu/74xx_7xx contains cpu.c, cpu_init.c, cache.S, interrupts.c, start.S, traps.S, etc. |
| disk | contains partition and device information for disks |
| drivers | contains various device drivers; i.e. ns9750_eth, rt18019, serial, usbtty, etc. |
| include | contains various header files; i.e. console.h, version.h, usb.h, pci.h. version.h has the version display header #define u-boot_VERSION “xxx” include/configs contains the configurations for all the various boards |
| lib_generic | contains generic libraries; i.e. bzlib.c vsprintf, string.c, etc. |
| lib_ppc | contains ppc specific libraries; i.e. bat_rw.c, board.c (memory sizes, baud rates, and calls board specific routines in /u-boot/board), interrupts.c, ppcstring.S, time.c, etc. |
| net | contains ethernet files; i.e. eth.c, net.[ch], nfs.[ch], bootp.c [ch], etc. |
| doc | contains various readme |
| dtt | contains temperature sensor code |

| | |
|---|---|
| examples | contains some stand-alone example code; i.e. hello_world.c, etc. |
| fs | contains file system directories and code; i.e. fat, fdos, jffs2 |
| rtc | contains real time clock code; i.e. date.c, mpc8xx.c, etc. |
| tools | contain various tool directories and files; i.e. env, gdb, logos, scripts, mkimage.c, etc. |
| post | contains directories and files; i.e. post.c codec.c, cache.c, memory.c, uart.c, etc. |

Relevant directories for possible modifications are:

| | |
|---|---|
| u-boot/. | u-boot/include |
| u-boot/board | u-boot/cpu |
| u-boot/lib_ppc | u-boot/common |

The major changes required for porting to any board and, specifically to this board, are the changes related to a new processor and a new bridge chip, as well as any other peripheral devices. For the HPC II board, the processor, MPC7447A and MPC7448, and bridge chip, Tsi108 are new. The flash, although new, complies with the Common Flash Interface (CFI) standards and flash operations can, therefore, use the existing driver by adding flash specific definitions.

A common practice is to use an existing configuration for some similar board and create a new configuration for the new board.

First, create a new set of directories and files to mimic the existing configuration. The directories and files necessary are:

1. u-boot/board/Freescale
2. u-boot/board/Freescale/freeserve2
3. u-boot/board/Freescale/freeserve2/freeserve2.c
4. u-boot/include/configs/freeserve2.h
5. u-boot/Makefile

Then, create new directories, copy files, and make changes to Makefile and platform specific files to retarget the code for HPC II.

## 5.1 u-boot/board

These commands will create a new directory and populate it with an existing set of files, changing the file names appropriately.

```
cp  -r   Tundra/tsi108board    Freescale/freeserve2
cd  Freescale/freeserve2
mv tsi108board.c  freeserve2.c
```

## 5.2 u-boot/makefile

Modify the top level makefile to specify the new configuration. All the configurations are listed under various titles. Use the major title, 74xx_7xx, since the MPC7447A/8 is a 74xx style processor. Under this title, add these lines:

```
FS2: unconfig
  @./mkconfig $(@:_config=) ppc  74xx_7xx  freeserve2  Freescale
```

## 5.3 u-boot/include/configs

Finally, a header file needs to be created with all the definitions (i.e. #define) for the various processor and board parameters.

```
cp  TSI108BOARD.h   fs2.h
```

## 5.4 Steps for Make

At this point, there is a new configuration called FS2 which, when built, will mimic the original Tundra board. The following commands will build the executables for the Tundra board. This is our first step to ensure that the targets will build satisfactorily. Following this, the files will be modified in these directories for our specific HPC II board.

```
make distclean
make FS2_config
make
```

The result of these commands will build the following targets and there should be no compilation errors. (The compile-time warnings are ok, but should be fixed before a final release.)

1. u-boot.bin is the binary image.
2. u-boot.srec is the Freescale S-Record image.
3. u-boot is the ELF file and can be used to generate the disassembly.
   — objdump -D u-boot > u-boot.dis
4. u-boot.map has the global addresses for symbols, etc.

# 6 Modifying the Code

Figure 2 shows the block diagram for the HPC II board.

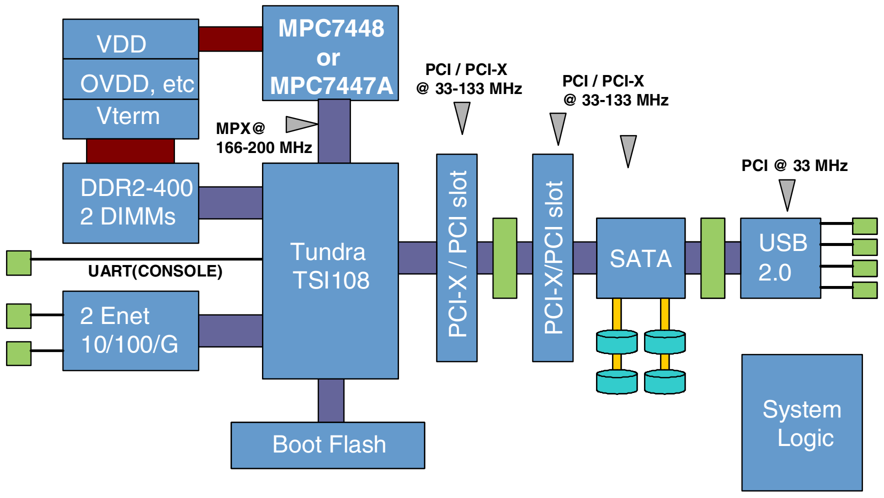


**Figure 2. Block Diagram of HCP II**

The processor can be either an MPC7447A or MPC7448; the code must be able to recognize either processor from it's PVR. Memory, Ethernet, Flash, PCI, USB, and SATA are all controlled via the Tsi108 bridge chip. The system logic block is an FPGA that controls start up and temperature.

A series of files needs to be modified to accommodate the processor and bridge chip. These are listed below:

1. Low level processor and board initialization
   u-boot/include/configs/FS2.h
   u-boot/board/Freescale/freeserve2/tsi108_init.c, board_early_init_r function
2. RS232 serial input and output.
   u-boot/board/Freescale/freeserve2/serial.c
3. DRAM controller initialization
   u-boot/include/configs/FS2.h
   u-boot/board/Freescale/freeserve2/asm_init.S
4. PCI bus controller initialization
   u-boot/include/configs/FS2.h
   u-boot/board/Freescale/freeserve2/tsi108_init.c
   u-boot/board/Freescale/freeserve2/pci.c
5. Flash memory programming from U-Boot
   u-boot/include/configs/FS2.h
   u-boot/include/flash.h
   u-boot/board/Freescale/freeserve2/cfi_flash.c
6. Ethernet drivers for U-Boot; TFTP file download
   u-boot/include/configs/FS2.h
   u-boot/board/Freescale/freeserve2/tsi108_eth.c <----------Gigabit ethernet in Tsi108
   u-boot/drivers/rtl8139.c <-----PCI-based Network Interface Card
7. No hard drive controllers are defined at this time
8. Board configuration

The basic board configuration is done in the file include/configs/FS2.h and includes protective conditional compilation. It is invoked at compile time via the makefile configuration command. Also, the Tsi108 bridge chip memory reference address, i.e. the address of the start of the Tsi108 registers, is set. The DUART serial port registers are also specified here.

These changes are shown below:

```
/* Macro for conditional compilation protects all FS2 specific code */

#define CONFIG_FS2

Source code is protected by the construct:

#ifdef CONFIG_FS2

#else  /*! CONFIG_FS2 */

#endif /* CONFIG_FS2 */

Set base address for the system controller Tsi108

#define CFG_TSI108_CSR_RST_BASE 0xC0000000

#define CFG_OCN_CLK    133000000 /* Internal OnChipNet bridge switch fabric speed in
MHz */

#define CFG_DUART_IO     (CFG_TSI108_CSR_RST_BASE + 0x7808)
```

The DUART clock divisor is set in the file board/Freescale/freeserve2/serial.c, serial_init():

```
        int clock_divisor = (CFG_OCN_CLK/2) / gd->baudrate;
```

## 6.1 Bridge Chip Switch Fabric Specification

The Tsi108 has a switch fabric for communication between the various integrated peripheral devices within it. This diagram shows the relationship of the fabric. The CPU can access the memory via the bridge chip. The other devices access memory through the on-chip memory controller and DMA.

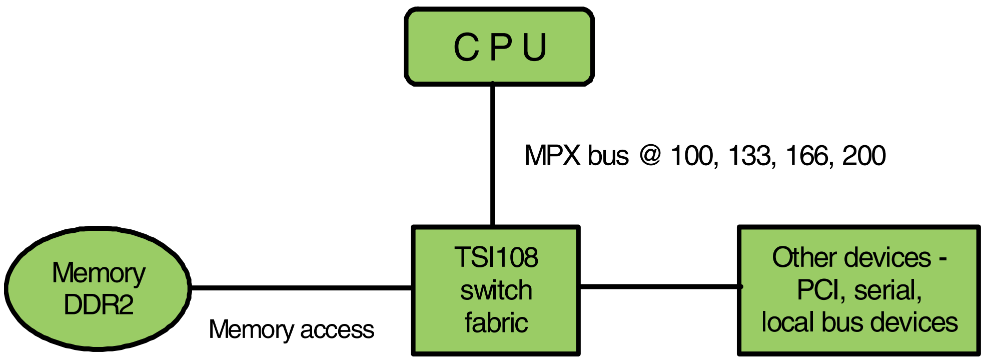
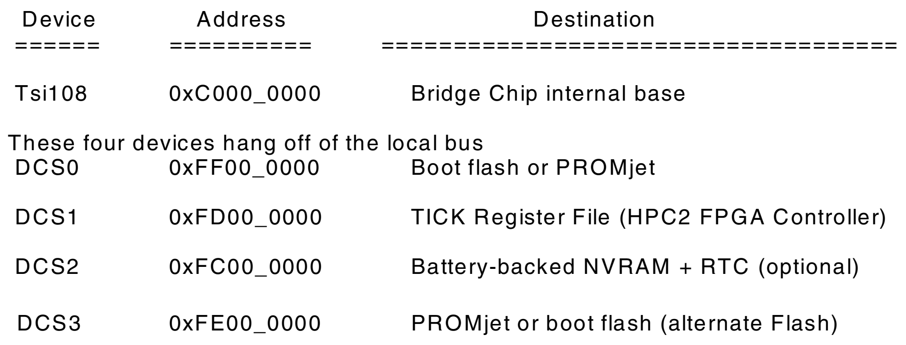

CPU
MPX bus @ 100, 133, 166, 200
Memory DDR2
Memory access
TSI108 switch fabric
Other devices - PCI, serial, local bus devices

**Figure 3. OCN Fabric Switch**

In order to access these devices via the bridge chip, the memory map must be specified.



| Device | Address | Destination |
|---|---|---|
| Tsi108 | 0xC000_0000 | Bridge Chip internal base |
| These four devices hang off of the local bus | | |
| DCS0 | 0xFF00_0000 | Boot flash or PROMjet |
| DCS1 | 0xFD00_0000 | TICK Register File (HPC2 FPGA Controller) |
| DCS2 | 0xFC00_0000 | Battery-backed NVRAM + RTC (optional) |
| DCS3 | 0xFE00_0000 | PROMjet or boot flash (alternate Flash) |

**Figure 4. Base Address Values**

The flash is defined in board/Freescale/freeserve2/cfi_flash.c in flash_init().

The TICK (FPGA), NVRAM, and PROMjet are set in board/Freescalefreeserve2/tsi108_init.c.

In order to access these addresses, the BAT registers are used to define accessible memory, and these are defined in include/configs/FS2.h.

## 6.2 Setting the BATs

In include/configs/FS2.h

- #define CFG_IBAT0U  0xFE0003FF
- #define CFG_IBAT0L  0xFE000002
- 
- #define CFG_IBAT1U  0x00007FFF
- #define CFG_IBAT1L  0x00000012
- #define CFG_IBAT2U  0x80007FFF
- #define CFG_IBAT2L  0x80000022

==============NOT USED===========

- #define CFG_IBAT3U  0x00000000
- #define CFG_IBAT3L  0x00000000
- #define CFG_IBAT4U  0x00000000
- #define CFG_IBAT4L  0x00000000
- #define CFG_IBAT5U  0x00000000
- #define CFG_IBAT5L  0x00000000
- #define CFG_IBAT6U  0x00000000
- #define CFG_IBAT6L  0x00000000
- #define CFG_IBAT7U  0x00000000
- #define CFG_IBAT7L  0x00000000

- #define CFG_DBAT0U  0xE0003FFF
- #define CFG_DBAT0L  0xE000002A
- #define CFG_DBAT1U  0x00007FFF
- #define CFG_DBAT1L  0x00000012
- #define CFG_DBAT2U  0x80007FFF
- #define CFG_DBAT2L  0x8000002A
- #define CFG_DBAT3U  0xC0000003
- #define CFG_DBAT3L  0xC0000022

============NOT USED===========

- #define CFG_DBAT4U  0x00000000
- #define CFG_DBAT4L  0x00000000
- #define CFG_DBAT5U  0x00000000
- #define CFG_DBAT5L  0x00000000
- #define CFG_DBAT6U  0x00000000
- #define CFG_DBAT6L  0x00000000
- #define CFG_DBAT7U  0x00000000
- #define CFG_DBAT7L  0x00000000

**Figure 5. BAT Assignments**

The first three instruction BAT registers are used and the first four data BAT registers are used, while the rest are set to zero (are not used).

IBAT1 and DBAT1 define all the instruction and data memory space from 0 to 7FFF_FFFF, which is shown as SDRAM space, designated as PB_SDRAM_BAR_1 and PB_SDRAM_BAR_2 and shown in green in Figure 6 below.

In the memory diagrams below, the BAR stands for the Base Address Register. The memory addressed by each BAR is separated into 32 regions of equal size and same, or different, in attributes.

The memory map is shown below.

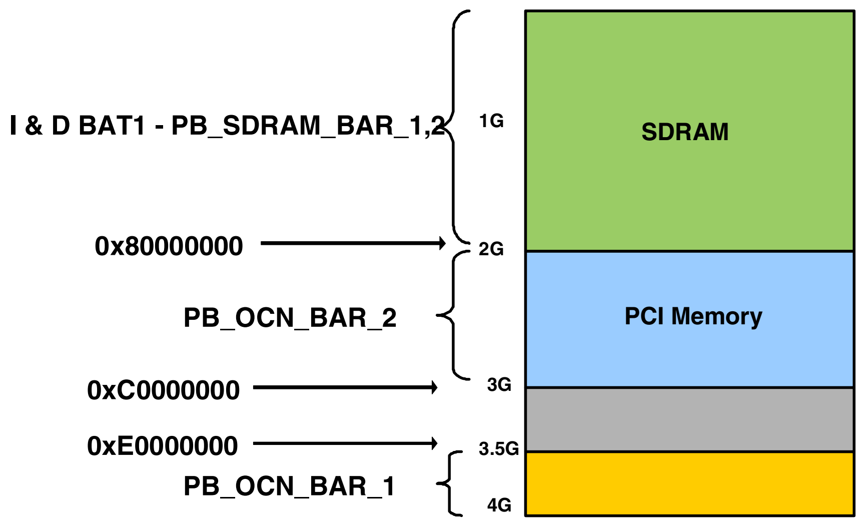


**Figure 6. SDRAM Memory Assignment**

IBAT2 and DBAT2 define all the instruction and data memory space from 8000_0000 to 8FFF_FFF. The PCI memory space is shown below, which is addressed by PB_OCN_BAR_2 shown as blue.

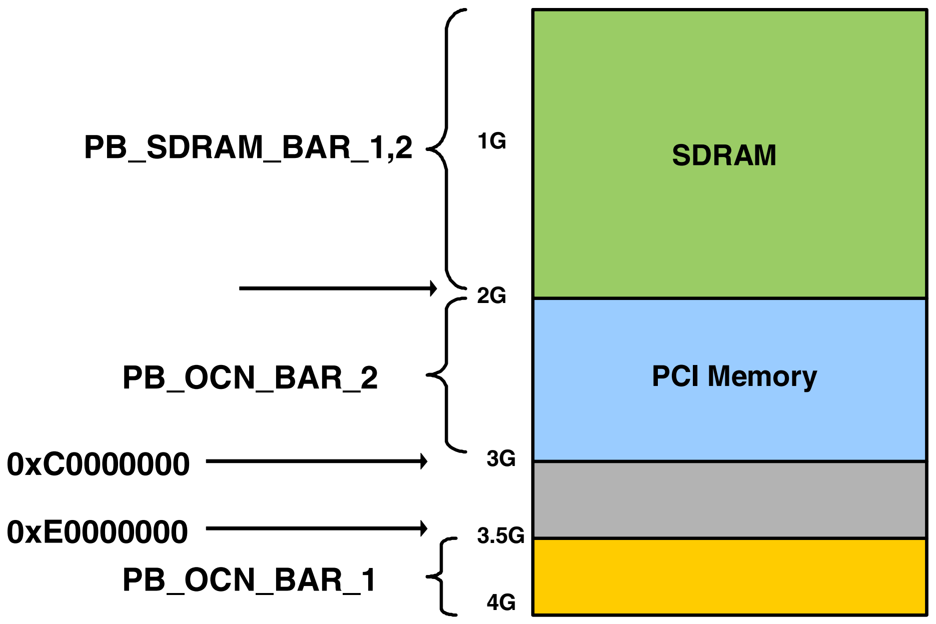


**Figure 7. PCI Memory Assignment**

DBAT3 defines data space from C000_0000 to C001_FFFF (part of the region shown in grey above).

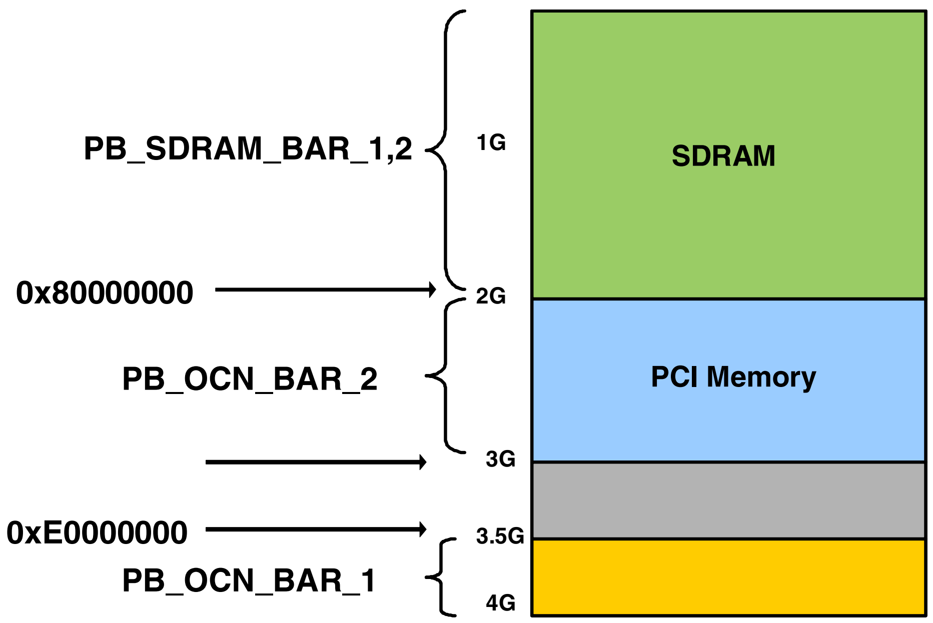


**Figure 8. PB_OCN_BAR_1 Space**

DBAT0 defines the data space from E000_0000 to EFFF_FFFF, which is covered by the PB_OCN_BAR_1 mapping register (designated in yellow). This size of this space is 0.5 GB.

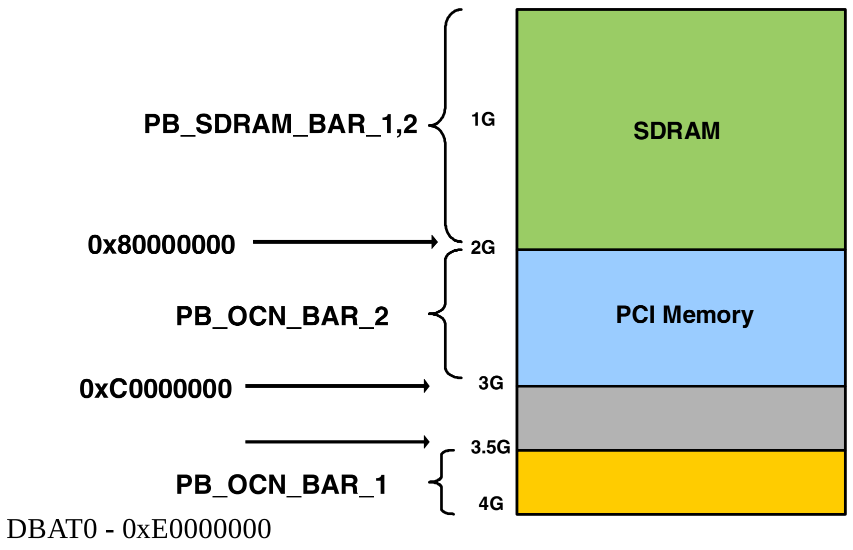


**Figure 9. DBAT0 Space Within All Memory Space**

Finally, IBAT0 defines the instruction space from FE00_0000 to FFFF_FFFF, which is the last section of memory and is used for alternate and boot ROM space, as shown below. This alternate flash space is from FE00_0000 to FEFFFFFF.

In summary, the BATs define these spaces:

DBAT1 & IBAT1 is 0x00000000

DBAT2 & IBAT2 is 0x80000000

DBAT3 is 0xC0000000

DBAT0 is 0xE0000000

The final memory picture shown below contains just the last 1GB of memory, from 3GB to 4GB, and is defined by DBAT0 and IBAT0.

This is the last 1GB of space, starting at 3GB, 0xC0000000.

Tsi108 registers are in the first section of this space.

DBAT3 is 0xC0000000 Tsi108 registers.

IBAT0 is 0xFE000000 (alternate flash) and 0xFF000000 (main flash).

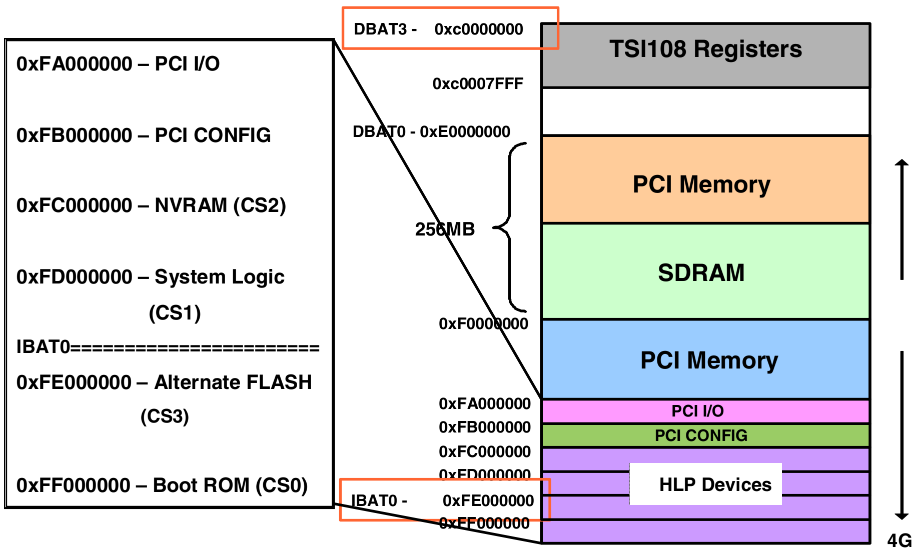


**Figure 10. The Last 1GB (i.e. 3GB–4GB) of Space**

The BAT registers are not completely ready for use at this point. The BAT registers, by default, cannot access greater than 256MB, however, BAT1 and 2 need to define a 1GB space. Hence, turn on the XBSEN bit to allow BATs to define these larger spaces.

Add this code to board/Freescale/freeserve2/asm_init.S:

```
mfspr   r5, HID0
oris    r5, r5, 0x0080  // Set HID0[HIGH_BAT_EN] bit #8
ori     r5, r5, 0x0380    //Set SPD,XBSEN,SGE bit #22,23,24
mtspr   HID0, r5
isync
sync


mfmsr   r3
ori     r3, r3, 0x2000
```

```
  mtmsr    r3

  isync

  sync
```

The above piece of code also enables the bit for floating point unit availability. Extended BAT size has been allowed and the high BATs have been turned on (not used). In effect, the following BAT registers have been specified, corresponding to the pictures of the memory maps above.

```
IBAT   BLOCK ADDRESS      SIZE  REPLACEMENT ADDR    WIMG  PROT  VALID   STATUS
====  ================== ====  ==================  =====  ====  =====  ==========
 0     fe000000..ffffffff   32M    fe000000..ffffffff   0000   R/W   [S,U]   [ENABLED ]
 1     00000000..3fffffff    1G    00000000..3fffffff   0010   R/W   [S,U]   [ENABLED ]
 2     80000000..bfffffff    1G    80000000..bfffffff   0100   R/W   [S,U]   [ENABLED ]
 3     00000000..0001ffff  128K   00000000..0001ffff   0000   NONE  [-,-]   [DISABLED]
 4     00000000..0001ffff  128K   00000000..0001ffff   0000   NONE  [-,-]   [DISABLED]
 5     00000000..0001ffff  128K   00000000..0001ffff   0000   NONE  [-,-]   [DISABLED]
 6     00000000..0001ffff  128K   00000000..0001ffff   0000   NONE  [-,-]   [DISABLED]
 7     00000000..0001ffff  128K   00000000..0001ffff   0000   NONE  [-,-]   [DISABLED]

DBAT  BLOCK ADDRESS       SIZE   REPLACEMENT ADDR  WIMG  PROT  VALID   STATUS
====  ================== ====  ==================  =====  ====  =====  ==========
 0     e0000000..ffffffff   512M   e0000000..ffffffff    0101   R/W   [S,U]   [ENABLED ]
 1     00000000..3fffffff    1G    00000000..3fffffff    0010   R/W   [S,U]   [ENABLED ]
 2     80000000..bfffffff    1G    80000000..bfffffff    0101   R/W   [S,U]   [ENABLED ]
 3     c0000000..c001ffff   128K   c0000000..c001ffff    0100   R/W   [S,U]   [ENABLED ]
 4     00000000..0001ffff  128K   00000000..0001ffff    0000   NONE  [-,-]   [DISABLED]
 5     00000000..0001ffff  128K   00000000..0001ffff    0000   NONE  [-,-]   [DISABLED]
 6     00000000..0001ffff  128K   00000000..0001ffff    0000   NONE  [-,-]   [DISABLED]
 7     00000000..0001ffff  128K   00000000..0001ffff    0000   NONE  [-,-]   [DISABLED]
```

## 6.3 Memory Settings

The following code sets the parameters for the DDR2 memory in the bridge chip.

```
In board/Freescale/freeserve2/asm_init.S,


#ifdef CONFIG_FS2      /* Hardcoded settings for HPC II */

/* Micron MT9HTF6472AY-40EA1 : Unbuffered, 512MB, 400, CL3, Single Rank */

#define VAL_SD_REFRESH   (0x61A)

#define VAL_SD_TIMING    (0x0308336b)

#define VAL_SD_D0_CTRL   (0x07100021) /* auto-precharge disabled */

#define VAL_SD_D0_BAR    (0x0FE00000) /* 512MB @ 0x00000000 */

#define VAL_SD_D1_CTRL   (0x07100020) /* auto-precharge disabled */

#define VAL_SD_D1_BAR    (0x0FE00200) /* 512MB @ 0x20000000 */

#endif /* CONFIG_FS2 */
```

Note that the code is protected by the #ifdef CONFIG_FS2

The above code snippet employs hard-coded values to initialize the memory controller. For the memory controller to perform efficiently, the frequency of the bus is detected and the timing and control registers are initialized accordingly. A flag called SDC_HARDCODED_INIT is used to choose between hard-coded values and spd-based initialization values.

```
sdc_init:
    ori r4,r29,TSI108_SD_REG_OFFSET // r4 - ptr to SDRAM registers

     LOAD_U32(r5,0x00)
    stw r5,SD_INT_ENABLE(r4)    // Ensure that interrupts are disabled
#ifdef ENABLE_SDRAM_ECC
    li r5, 0x01
#endif // ENABLE_SDRAM_ECC
    stw r5,SD_ECC_CTRL(r4)      // Enable/Disable ECC
    sync

#ifdef SDC_HARDCODED_INIT /* config sdram controller with hardcoded values */

    //First read the CG_PWRUP_STATUS register to get the
    //memory speed from bits 22,21,20

    LOAD_U32(r3,0xC0002234)
    lwz  r3,0(r3)
    rlwinm  r3,r3,12,29,31

    //Now first check for 166, then 200, or default

    cmpi 0,0,r3,0x0005
    bne check_for_200mhz

    //set values for 166 Mhz memory speed

      /* Set refresh rate and timing parameters */
    LOAD_U32(r5,0x00000515)
    stw r5,SD_REFRESH(r4)
    LOAD_U32(r5,0x03073368)
    stw r5,SD_TIMING(r4)
    sync

    /* Initialize DIMM0 control and BAR registers */
      LOAD_U32(r5,VAL_SD_D0_CTRL)/* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
```

```
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
    stw r5,SD_D0_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D0_BAR)
    stw r5,SD_D0_BAR(r4)
    sync

    /* Initialize DIMM1 control and BAR registers
     *  (same as dimm 0, next 512MB, disabled)
       */
    LOAD_U32(r5,VAL_SD_D1_CTRL)  /* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
    stw r5,SD_D1_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D1_BAR)
    stw r5,SD_D1_BAR(r4)
    sync

    b sdc_init_done

check_for_200mhz:

    cmpi 0,0,r3,0x0006
    bne set_default_values

    //set values for 200Mhz memory speed

      /* Set refresh rate and timing parameters */
    LOAD_U32(r5,0x0000061a)
    stw r5,SD_REFRESH(r4)
    LOAD_U32(r5,0x03083348)
    stw r5,SD_TIMING(r4)
    sync

    /* Initialize DIMM0 control and BAR registers */
      LOAD_U32(r5,VAL_SD_D0_CTRL)/* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
```

```
    stw r5,SD_D0_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D0_BAR)
    stw r5,SD_D0_BAR(r4)
    sync

    /* Initialize DIMM1 control and BAR registers
     *  (same as dimm 0, next 512MB, disabled)
       */
    LOAD_U32(r5,VAL_SD_D1_CTRL)  /* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
    stw r5,SD_D1_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D1_BAR)
    stw r5,SD_D1_BAR(r4)
    sync

    b sdc_init_done

set_default_values:

      /* Set refresh rate and timing parameters */
    LOAD_U32(r5,VAL_SD_REFRESH)
    stw r5,SD_REFRESH(r4)
    LOAD_U32(r5,VAL_SD_TIMING)
    stw r5,SD_TIMING(r4)
    sync

    /* Initialize DIMM0 control and BAR registers */
      LOAD_U32(r5,VAL_SD_D0_CTRL)/* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
    stw r5,SD_D0_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D0_BAR)
    stw r5,SD_D0_BAR(r4)
    sync

    /* Initialize DIMM1 control and BAR registers
     *  (same as dimm 0, next 512MB, disabled)
```

```
        */
    LOAD_U32(r5,VAL_SD_D1_CTRL)  /* auto-precharge disabled */
#ifdef SDC_AUTOPRECH_EN
    oris r5,r5,0x0001   /* set auto precharge EN bit */
#endif
    stw r5,SD_D1_CTRL(r4)
    LOAD_U32(r5,VAL_SD_D1_BAR)
    stw r5,SD_D1_BAR(r4)
    sync

#endif /* SDC_HARDCODED_INIT */
```

The current version of u-boot supports SPD (Serial Presence Detect) initialization of the memory controller. Rather than using hard-coded values to set-up DDR2, the I2C interface is used to read the memory settings and set-up the controller for optimal performance. In this case, the following line in asm_init.S is commented:

```
//#define SDC_HARDCODED_INIT
```

Also the SPD routine is called during the initialization process.

```
#else //!SDC_HARDCODED_INIT
   b tsi108_sdram_spd
#endif //SDC_HARDCODED_INIT
```

The reader is referred to the actual code, in board/Freescale/freeserve2/asm_init.S, for the spd-based init routines to populate the memory controller registers.

## 6.4 HLP Space

The HLP (Host Local Port, commonly known as the local bus) space shown in Figure 10 refers to the Host Local Ports 0–3.

It has four banks (i.e. four chip selects):

| | |
|---|---|
| cs0 0xFF000000 | flash |
| cs1 0xFC000000 | nvram + rtc |
| cs2 0xFD000000 | TICK system logic |
| cs3 0xFE000000 | PROMjet or alternate flash |

Appropriate values are set to access this HLP space.

## 6.5 Base Address Registers (BAR)

Each of the two Base Address Registers maps memory spaces separated into 32 equal regions with independent attributes defined in a table of 32 entries.

PB_OCN_BAR_1: 0xE000_0000 - 0xFFFF_FFFF  512MB

PB_OCN_BAR_2: 0x8000_0000 – 0xBFFF_FFFF  1GB

The Look Up Table (LUT) entries are used to define this space.

Figure 11 shows the BAR1 space allocation as defined in a LUT.

**In board/Freescale/freeserve2/tsi108_init.c**

```
PB2OCN_LUT_ENTRY pb2ocn_lut1[32] =
{
   // 0 - 7
   {0x00000000, 0x00000201},  // PBA=0xE000_0000 -> PCI/X (Byte-Swap)
   {0x00000000, 0x00000201},  // PBA=0xE100_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE200_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE300_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE400_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE500_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE600_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE700_0000 -> PCI/X
   // 8 - 15
   {0x00000000, 0x00000201},  // PBA=0xE800_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xE900_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xEA00_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xEB00_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xEC00_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xED00_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xEE00_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xEF00_0000 -> PCI/X
```

**continued.**

**Figure 11. BAR1 LUT (Look Up Table) Part 1**

The first set of values defines the PCI/X space, {0x00000000, 0x00000201},  // PBA=0xE000_0000 -> PCI/X.

Each set of numbers has a specific meaning. BAR2 starts at 0xE000_0000 and covers 1/32 of the space. Each entry is 0100_0000 bytes long. (An entry covers 1/32 of the total region addressed by PB_OCN_BARn; the size of each region is decided by value in PB_OCN_BARn). Below is a more detailed look at the LUT entries:

{0x00000000, 0x00000201},  // PBA=0xE100_0000 -> PCI/X

(upper, lower), // comments to describe the entry

upper is the translated address, and 0 implies no translation

lower is the attribute list as defined in the Tsi108 reference manual section 17.2.22

0x0000_0y0z  where the y =>endian mode, z => port number

The port numbers are shown below:

0 = HLP

1 = PCI-X

2 = processor interface master

3 = processor interface slave

4 = memory controller

5 = dma controller

6 = ethernet controller

7 = reserved

So, for the first entry, the lower value in the LUT (i.e. 0x0000_0201) means byte swap and port 1 = PCI

Similarly, with the following entry:

{0x00000000, 0x18000004},  // PBA=0xE800_0000 -> SDRAM_OCN, the 0004 means no byte swap and use memory controller to address a region in the memory.

```
In board/Freescale/freeserve2/tsi108_init.c

 // 16 - 23
   {0x00000000, 0x00000201},  // PBA=0xF000_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF100_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF200_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF300_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF400_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF500_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF600_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF700_0000 -> PCI/X

// 24 - 31
   {0x00000000, 0x00000201},  // PBA=0xF800_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xF900_0000 -> PCI/X
   {0x00000000, 0x00000201},  // PBA=0xFA00_0000 -> PCI/X /* PCI   I/O space */
   {0x00000000, 0x00000201},  // PBA=0xFB00_0000 -> PCI/X  /* PCI config space */
   {0x00000000, 0x02000240},  // PBA=0xFC00_0000 -> HLP  /* cs1 nvram */
   {0x00000000, 0x01000240},  // PBA=0xFD00_0000 -> HLP  /* cs2 TICK */
   {0x00000000, 0x03000240},  // PBA=0xFE00_0000 -> HLP  /* cs3  alternate flash */
   {0x00000000, 0x00000240}   // PBA=0xFF00_0000 -> HLP   /* cs0 boot flash */
```

**Figure 12. HLP Space**

0x00000000, 0x00000240}   // PBA=0xFF00_0000 -> HLP   /* cs0 boot flash */

0240 => 2 = byte swap, 4 = enable translation, 0 = port # = HLP

```
In board/Freescale/hpc2/tsi108_init.c, board_early_init_r()

   //Base addresses for CS0, CS1, CS2, CS3

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B0_ADDR, 0x00000000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B1_ADDR, 0x00100000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B2_ADDR, 0x00200000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B3_ADDR, 0x00300000);
   __asm__ __volatile__ ("sync");

   //Masks for HLP banks

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B0_MASK, 0xFFF00000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B1_MASK, 0xFFF00000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B2_MASK, 0xFFF00000);
   __asm__ __volatile__ ("sync");

   out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B3_MASK, 0xFFF00000);
   __asm__ __volatile__ ("sync");

continued.

NOTE: The 16-bit upper bits in the HLP_Bn_ADDR register indicate which HLP bank will be
      used by the HLP port for a given address range.
```

**Figure 13. Setting the Base Address Registers**

CFG_TSI108_CSR_BASE =0xc000_0000 is set in u-boot/include/configs/FS2.h.

The U-Boot macro, out32, is used for writing these values to the Tsi108 registers starting at 0xc000_0000.

The first set writes the CSx addresses and the next set writes the masks.

**In board/Freescale/freeserve2/tsi108_init.c, board_early_init_r()**

```
//Set CTRL0 values for banks

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B0_CTRL0, 0x7FFC44C2);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B1_CTRL0, 0x7FFC44C0);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B2_CTRL0, 0x7FFC44C0);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B3_CTRL0, 0x7FFC44C2);
__asm__ __volatile__ ("sync");


//Set banks to latched mode, enabled, and other default settings

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B0_CTRL1, 0x7C0F2000);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B1_CTRL1, 0x7C0F2000);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B2_CTRL1, 0x7C0F2000);
__asm__ __volatile__ ("sync");

out32(CFG_TSI108_CSR_BASE + TSI108_HLP_REG_OFFSET + HLP_B3_CTRL1, 0x7C0F2000);
__asm__ __volatile__ ("sync");
```

**Figure 14. Setting the Base Address Registers (continued)**

The macro, out32, writes control values.

The values shown in the figure above were obtained from Tundra.

Description of the bank values are: last digit indicate 0 => 8 bit for nvram and TICK, 2 => 32 bit for the flashes.

Finally, in board/Freescale/freeserve2/tsi108_init.c, board_early_init_r() write the HLP address value.

```
out32(CFG_TSI108_CSR_BASE + TSI108_PB_REG_OFFSET + PB_OCN_BAR1, 0xE0000011);
```

This code specifies that the PB_OCN_BAR_1 starts at 0xE000_0000 and the last two bits:

11 => boot bit is cleared and PB_OCN_BAR_1 is enabled.

## 6.6 Enable Flash Commands

The following code enters the commands into U-Boot when it is compiled for use during execution.

CFG_CMD_FLASH enables flash commands such as flinfo, etc.

CFG_CMD_ENV enables saving the environment variables. It works in conjunction with CFG_ENV_IS_IN_NVRAM; indicating where the environment variables are to be stored.

CFG_CMD_PING enables the ping command to check the status of machines on a designated network.

▪include/configs/FS2.h

```
#define CONFIG_COMMANDS ((CONFIG_CMD_DFL \
                | CFG_CMD_ASKENV \
                | CFG_CMD_CACHE \
                | CFG_CMD_JFFS2 \
        | CFG_CMD_KGDB \
                | CFG_CMD_PCI \
                | CFG_CMD_I2C \
        | CFG_CMD_SDRAM \
                | CFG_CMD_EEPROM \
                | CFG_CMD_NET \
                | CFG_CMD_FLASH) \
                | CFG_CMD_ENV) \
                | CFG_CMD_PING)
```

**Figure 15. Enabling Commands**

**NOTE**

The bold lines indicate the new commands added; the other commands were already present in the code.

In include/configs/FS2.h, the following #defines describe the flash configuration adopted for HPC II board.

```
#define CFG_MAX_FLASH_BANKS   1           /* 1 bank of Flash */
#define FLASH_BANK_SIZE       0x01000000  /* 16 MB Total */

#define CFG_FS2_FLASH_CFI_DRIVER
#define CFG_FLASH_CFI

#define PHYS_FLASH_SIZE          0x01000000

#define CFG_MAX_FLASH_SECT       (128)
```

The flag, CFG_FS2_FLASH_CFI_DRIVER, controls the inclusion of HPC II specific flash code, which is in cfi_flash.c under board/Freescale/freeserve2.

The following code allows for flash to flash programming:

```
        //Enable the code to run from either the promjet or the flash
        //and enable programming the flash from the u-boot flash
        //For Taiga, the on-board flash is 16MB in 2 16-bit chips i.e.
        //32-bit wide flash. With this knowledge, we can determine
        //if we are running from Flash or promjet and accordingly
        //decide the base address and any future interactions with flash.

        info->portwidth = FLASH_CFI_32BIT;
        info->chipwidth = FLASH_CFI_BY16;

        cptr1.cp = flash_make_addr(info,0,FLASH_OFFSET_CFI_RESP);
        cptr2.cp = flash_make_addr(info,0,FLASH_OFFSET_CFI_RESP + 1);
        cptr3.cp = flash_make_addr(info,0,FLASH_OFFSET_CFI_RESP + 2);

        flash_write_cmd (info, 0, 0, FLASH_CMD_RESET);
        flash_write_cmd (info, 0, FLASH_OFFSET_CFI, FLASH_CMD_CFI);

        if ( !(flash_isequal (info, cptr1, 'Q')
            && flash_isequal (info, cptr2, 'R')
            && flash_isequal (info, cptr3, 'Y')))
        {
           printf("Started u-boot from FLASH. Not using Promjet\n");
           base = 0xFF000000;
           info->start[0] = base;
        }
```

The above code is required to enable programming of flash from either start-up locations (the PROMjet or the flash itself). Note that the default base address for the flash is 0xFE000000, as specified in include/configs/FS2.h:

```
#define CFG_FLASH_BASE 0xfe000000  /* Base Address of Flash device */
```

## 6.7 PCI Controller Initialization

Configuration setup is performed in include/configs/FS2.h.

```
/* PCI Memory */

#define CFG_PCI_MEM_SIZE        0x10000000          //256 MB

#define CFG_PCI_MEM32_BASE      0xE0000000


/* PCI view of the system memory */

#define CFG_PCI_MEMORY_SIZE     0x80000000      //2G


/* PCI I/O Space */

#define CFG_PCI_IO_PHYS         0xfa000000

#define CFG_PCI_IO_SIZE         0x01000000             //16MB


/* PCI Config Space mapping */

#define CFG_PCI_CFG_BASE        0xfb000000

#define CFG_PCI_CFG_SIZE        0x01000000         // 16MB
```

Values are written to the Tsi108 registers in board/Freescale/freeserve2/tsi108_init.c, board_early_init_r():

```
//PCI Config space
out32(CFG_TSI108_CSR_BASE + TSI108_PCI_REG_OFFSET + PCI_PFAB_BAR0, 0xFB000001);


//PCI I/O space
out32(CFG_TSI108_CSR_BASE + TSI108_PCI_REG_OFFSET + PCI_PFAB_IO, 0xFA000001);
```

### 6.7.1 Setting Up the NIC

The NIC, Network Interface Card, usually is one of the last steps in coding U-Boot. The NIC allows ethernet facilities for downloading files, specifically, Linux kernels. The configuration code is u-boot/include/configs/FS2.h.

```
#define CONFIG_RTL8139
```

For the Taiga board, an RTL8139 PCI-based NIC was used.

The buffers are in u-boot/drivers/rtl8139.c. Change the relevant code in rtl8139.c source file to change the byte alignment as shown below.

```
/* The RTL8139 can only transmit from a contiguous, aligned memory block.  */
static unsigned char tx_buffer[TX_BUF_SIZE] __attribute__((aligned(32)));
static unsigned char rx_ring[RX_BUF_LEN+16] __attribute__((aligned(32)));
```

## 6.8 Setting Up the Gigabit Ethernet Controller

The NIC initialization mentioned in Section 6.7.1, “Setting Up the NIC”, is required as a back-up if the GigE ports require more time and effort to be functional during the U-Boot development process.

The Gigabit Ethernet driver for the Tsi108 is in board/Freescale/freeserve2/tsi108_eth.c.

To enable the use of this driver, the following flags need to be enabled and the init function needs to be called as part of the Ethernet initialize routine:

In net/eth.c, function eth_initialize()

```
#if defined(CONFIG_TSI108_ETH)
    tsi108_eth_initialize(bis);
#endif
```

In include/configs/FS2.h,

```
#define CONFIG_TSI108_ETH
```

## 6.9 Building and Starting U-Boot

The first command is:

```
make distclean,
```

which cleans up all the directories and removes all intermediate files, including objects and artifacts of a previous build.

This make command will set up all the configuration for the HPC II:

```
make FS2_config
```

Finally, make will compile and link U-Boot:

```
make
```

u-boot is an ELF file of the code.

u-boot.bin is the binary (with ELF headers striped out) and used for download and debugging.

Before Flash programming is available on U-Boot, a PROMjet is used to download the image and start execution.

A COP debug tool was employed for on-chip debugging.

A disassembly of the U-Boot ELF file is required for debugging; the instruction addresses in the disassembly file correspond to the addresses shown during COP debugging. The disassembly is generated using the following command:

```
objdump -D u-boot > u-boot.dis
```

## 7 Running U-Boot

Section 6.9, “Building and Starting U-Boot”, starts U-Boot running in the board. The first thing displayed on the serial console is the flash screen.

```
u-boot 1.1.2(HPC2_V2_r1 20050520) (May 20 2005 - 11:14:29)Freescale Tsi108 HPC2
Reference  Platform(2.0)

CPU:   MPC7447A v1.1 @ 665 MHz
BOARD: HPC2
Top of RAM usable for u-boot at: 02000000
Reserving 208k for u-boot at: 01fcb000
Reserving 257k for malloc() at: 01f8ac00
Reserving 68 Bytes for Board Info at: 01f8abbc
Reserving 48 Bytes for Global Data at: 01f8ab8c
Stack Pointer at: 01f8ab68
New Stack Pointer is: 01f8ab68
Now running in RAM - u-boot at: 01fcb000
In:    serial
Out:   serial
Err:   serial
Initializing MPIC .......done
KGDB:  kgdb ready
ready
u-boot relocated to 01fcb000
Net:   RTL8139#0
=>
```

**Figure 16. U-Boot Splash Screen**

If U-Boot stops anywhere short of the prompt, =>, then it will be necessary to debug the code. Discussion on debugging the code is beyond the scope of this paper. However, it is vital to understand that the first part of U-Boot initialization runs directly from ROM and, therefore, the addresses shown in the disassembly file (obtained from objdump -D command) will correspond directly to the addresses shown from the COP debugging tool. The second part of U-Boot copies itself to RAM and then continues execution from RAM. At start-up, the splash screen displays the relocation value,

```
u-boot relocated to 01FCB000.
```

Once relocated to RAM, addresses displayed by the COP debugging tool must be offset by this relocation address. For example, the address specified by the COP tool, 1fd0200, will correspond to the disassembly address of fff05200. This technique should be kept in mind while reading RAM locations. Prior to relocation, the u-boot code runs from the flash as stated above.

To read memory locations in the flash, it is important to be aware of the addressing behavior of Tsi108 for local bus devices. Due to the way the Tsi108 bridge chip translates addresses on the local bus, the addresses to be read must be calculated as explained below. The BOOT bit in PB_OCN_BAR1 (bit 30) is set to 1 when u-boot starts executing. At this point, the only region of memory visible to Tsi108 is the HLP0 bank to which the Boot ROM (Flash/PROMjet) is connected. During the boot-up process, the LUTs are set-up, memory regions are initialized to be assigned as PCI, SDRAM, etc. and finally the BOOT is cleared. At this point, a larger area of the flash is visible to the CPU. Address masks (as shown in Figure 13) have to be applied to avoid duplication of the image in other regions of the flash. This results in an address translation as follows:

after BOOT bit is cleared

fff00000 + x =================> ff000000 + x

Therefore, to display the first few memory locations in flash, the command in U-Boot is `md ff000000`. In fact, even before arriving at the u-boot prompt, once the BOOT bit is set to 0 in board_early_init_r() function (board/Freescale/freeserve2/tsi108_init.c), the base address of the flash is `ff000000` for any subsequent reads using the COP debuggers.

Figure 15 described turning on the FLASH, ENV, and PING commands. This table of commands determines which commands from the U-Boot superset of commands are available for this instantiation of U-Boot. The ‘help’ or ‘?’ commands will display the subset of commands available with this version of U-Boot.

```
=>help
?       - alias for 'help'
askenv  - get environment variables from stdin
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP
          protocol
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
dcache  - enable or disable data cache
echo    - echo args to console
eeprom  - EEPROM sub-system
erase   - erase FLASH memory
flinfo  - print FLASH memory information
fsinfo  - print information about filesystems
fsload  - load binary file from a filesystem image
go      - start application at address 'addr'
help    - print online help
icache  - enable or disable instruction cache
icrc32  - checksum calculation
iloop   - infinite loop on address range
imd     - i2c memory display
iminfo  - print header information for application image
imls    - list all images found in flash
imm     - i2c memory modify (auto-incrementing)
imw     - memory write (fill)
inm     - memory modify (constant address)
iprobe  - probe to discover valid I2C chip addresses
isdram  - print SDRAM configuration information
itest   - return true/false on integer compare
kgdb    - enter gdb remote debug mode
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
pci     - list and access PCI Configuraton Space
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
tftpboot- boot image via network using TFTP protocol
version - print monitor version

=>
```

**Figure 17. HPC II U-Boot—Available Commands**

The U-Boot command, help <command> (i.e. help printenv), gives more detail on how to use the command.

The next series of figures shows examples of using some of the more popular commands.

## 7.1 Displaying and Setting Environment Variables

The command, printenv, will display the current environment variables.

The commands => setenv loadaddr 200000 and => printenv change the “loadaddr” variable and then reprint all the environment variables. This change is only reflected in the local RAM of U-Boot and is non-persistent (i.e. will revert to it’s previous value upon a reboot).

The command => saveenv will save the environment variables to non-volatile storage, nvram. Upon reboot, Figure 21 shows that the value of loadaddr is changed across a reboot.

```
Example showing printenv & setenv commands

=>printenv
baudrate=115200
loads_echo=0
ipaddr=192.168.1.200
serverip=192.168.1.1
gatewayip=192.168.1.1
netmask=255.255.255.0
bootfile=zImage.initrd.elf
loadaddr=0x400000
stdin=serial
stdout=serial
stderr=serial
ethact=RTL8139#0
                                        continued...
```

Figure 18. Display the Environment Variables

```
Environment size: 218/1020 bytes
=>setenv loadaddr 200000
=>printenv
baudrate=115200
loads_echo=0
ipaddr=192.168.1.200
serverip=192.168.1.1
gatewayip=192.168.1.1
netmask=255.255.255.0
bootfile=zImage.initrd.elf
stdin=serial
stdout=serial
stderr=serial
ethact=RTL8139#0
```

Figure 19. Modify One Variable

```
Environment size: 216/1020 bytes
=>saveenv
Saving Environment to NVRAM...
=>
Now Reboot the system.

u-boot 1.1.2(HPC2_V2_r2) (Jun  7 2005 - 16:37:06)Freescale Tsi108 HPC2
Reference Platform (2.0)

CPU:  MPC7448 v1.0 @ 1200 MHz
BUS SPEED: 200 MHz
MEMORY SPEED:  200 MHz
BOARD: HPC2
DRAM:  512 MB
:
:
:
```

Figure 20. Save the Environment Variables

```
=>printenv
baudrate=115200
loads_echo=0
ipaddr=192.168.1.200
serverip=192.168.1.1
gatewayip=192.168.1.1
netmask=255.255.255.0
bootfile=zImage.initrd.elf
stdin=serial
stdout=serial
stderr=serial
ethact=RTL8139#0
loadaddr=200000
```

Figure 21. Show That loadaddr is Now Persistent

## 7.2 bdinfo—Board Information Used for Booting uImage Linux

bdinfo displays the boot parameters for the Linux kernel when invoked from the U-Boot boot command. In this case, the kernel needs to be compiled to create a uImage binary file.

**bdinfo shows the contents of struct bd_t which is passed to linux built as a uImage**

```
=>bdinfo
bd address  = 0x0FF8DBC8
memstart    = 0x00000000
memsize     = 0x20000000
flashstart  = 0xFFF00000
flashsize   = 0x01000000
flashoffset = 0x0002A800
sramstart   = 0x00000000
sramsize    = 0x00000000
bootflags   = 0x00000001
intfreq     =   1200 MHz
busfreq     =    200 MHz
ethaddr     = 00:00:00:00:00:00
IP addr     = 192.168.1.200
baudrate    = 115200 bps
=>
```

Figure 22. uImage Parameter List

## 7.3 version, ping and pci

The version command displays the U-Boot version, which is the first line of the splash screen. The pci command displays the current cards in any PCI slot and/or on the PCI bus. The ping command, when an Ethernet card is present or the GigE controller is enabled, will ping a specified Ethernet address.

```
=>version

u-boot 1.1.2(HPC2_V2_r2) (Jun  7 2005 - 16:37:06)Freescale Tsi108 HPC2 Referen
ce Platform (2.0)

=>pci
Scanning PCI devices on bus 0
BusDevFun  VendorId   DeviceId   Device Class       Sub-Class
_____________________________________________________________
00.01.00   0x10ec     0x8139     Network controller     0x00

=>ping 192.168.1.1
Trying RTL8139#0
rtl8139_probe() - in
MAC addr = 00:40:f4:7a:eb:a8
rtl8139_probe() - out
Using RTL8139#0 device
host 192.168.1.1 is alive
=>
```

Figure 23. version, pci, and ping Commands

## 7.4 Memory Display

U-Boot does not have the facility to display any processor level registers, however, it can display memory. Since all the Tsi108 registers are memory mapped, the memory display command can show the contents of these registers. The Tsi108 registers start at 0xc000_0000.

```
0xC0004000 - sdram controller registers in Tsi108 (base is 0xc0000000
 and sdram controller is offset from base by 0x4000)

=>md c0004000
c0004000: 00000230 00000001 03083348 0000061a    ...0......3H....
c0004010: 00000000 00000000 00000000 ffffffff    ................
c0004020: 00110021 00110020 0fe00000 0fe00200    ...!... ........
c0004030: ffffffff ffffffff ffffffff ffffffff    ................
c0004040: 00000000 00000000 00000000 00000000    ................
c0004050: 00000000 00000000 00000000 00000000    ................
c0004060: ffffffff ffffffff ffffffff ffffffff    ................
c0004070: ffffffff ffffffff ffffffff ffffffff    ................
c0004080: 00011102 ffffffff ffffffff ffffffff    ................
```

**Figure 24. Bridge Chip Register Values as Memory Addresses**

## 7.5 Memory Modify and Execution of Small Program

U-Boot commands can display and modify memory, and the go command can start execution at any memory location. The next example shows a way to modify memory with a hexadecimal value that is the representation of an instruction. Then, by a “go” to this address, U-Boot can begin execution of this instruction and then a whole program (i.e. a series of instructions).

In order to terminate the program nicely and return to U-Boot, a blr needs to terminate the program. U-Boot will set it’s return address in the link register, lr, and then transfer to the address specified. The program will execute the blr and return to U-Boot.

Figure 25 shows the program and it’s corresponding hex values, which are then entered into memory with the `mm` command at address 0x10_000.

Figure 26 sets address 0x20020 to zero and displays it as zero. Then the command go 100000 starts the program and, following normal termination, returns to U-Boot. Finally, the last command, md 20020, shows that the program executed properly because the address 0x20020 now contains the value 0x23.

**NOTE**

U-Boot commands always assume hex values, thus, the prefix 0x is not used for U-Boot commands.

```
Modify memory, build a small program to set 0x00020020 to 0x23 and
execute it using go command.

The following program executed correctly, and with the d-cache set to
write-through, it wrote to the memory as expected

   lis r5,2            3ca00002
   ori r5,r5,32        60a50020
   li r3,35            38600023
   stw r3,0(r5)        90650000
   blr                 4e800020

This writes 0x23 to memory address 0x20020

=>mm 100000
00100000: 41820034 ? 3ca00002
00100004: 80040024 ? 60a50020
00100008: 39200008 ? 38600023
0010000c: 91240020 ? 90650000
00100010: 39200000 ? 4e800020
```

**Figure 25. Modify Memory with a Program**

```
=>mm 20020
00020020: 00000023 ?                 00000000
00020024: 7c003670 ?

=>md 20020
00020020: 00000000 7c003670 7c004a14 7c005850    ....|.6p|.J.|.XP
00020030: 901f52c0 4bfffe90 801c146c 70000030   ..R.K......lp..0..................
00020110: 3c00c22e 60004507 3d200001 7c0a0096    <...`.E.= ..|...

=>go 100000
## Starting application at 0x00100000 ...
## Application terminated, rc = 0x23

=>md 20020
00020020: 00000023 7c003670 7c004a14 7c005850    ...#|.6p|.J.|.XP
00020030: 901f52c0 4bfffe90 801c146c 70000030   ..R.K......lp..0..................
00020110: 3c00c22e 60004507 3d200001 7c0a0096    <...`.E.= ..|...
=>
```

**Figure 26. Execute the Program**

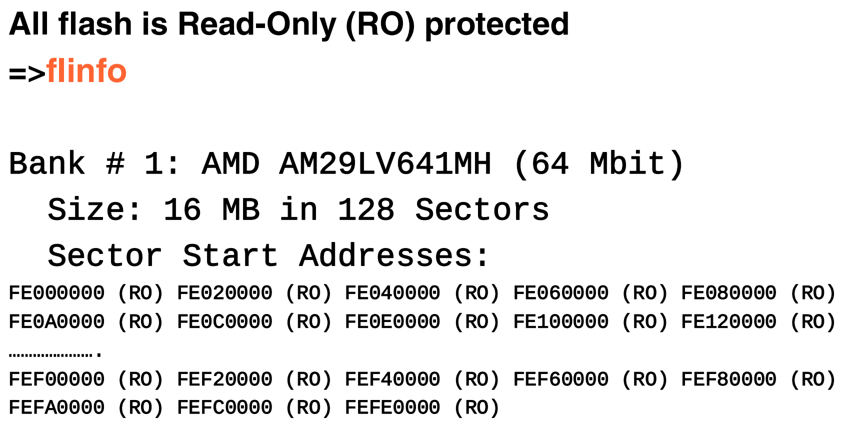
## 7.6 Programming the Flash

There are eight steps to programming the flash. The example below will copy the U-Boot instructions from the PROMjet location, ff00_0000 to the flash location, fe00_0000. Then, by flipping the switch on the board (see Section 8, “References”), the board will boot from flash instead of PROMjet.

The following sequence of commands shows how flash programming is done using U-Boot on HPC II:

1. Display the sector protection information using flinfo, showing that they are read-only (RO).
   — The command, flinfo, displays the flash protection. Default is on. See Figure 27.
2. Unprotect the sectors
   — The command, protect off all, turns off all protection for the flash. The flinfo command this time shows that there is no protection (i.e. no (RO)). See Figure 28.

3. Display the sector protection information
   - Use flinfo, showing that they are writable. See Figure 28.
   - Display current contents of flash, md fe000000. See Figure 29.
4. Erase all flash
   - Use the command md fe000000 to see if the contents of flash are erased (i.e. all f's). See Figure 30.
5. Copy the contents of the PROMjet into the flash location
   - cp.w <promjet> <flash> <#words>
   - cp.w ff000000 fe000000 b100
     - Copy words from PROMjet ff00_0000 to flash fe00_0000 for 0xb100 words, more than enough to capture all of U-Boot. See Figure 32.
6. Use md fe000000 to see if U-Boot has been copied into the flash. See Figure 32.
7. Change switch settings to boot from flash instead of PROMjet
8. Reset the board
   - Boot U-Boot from flash. Also, the command protect on all resets the protection to read-only.
   - This command should reboot the board from flash and display the splash screen.



```
All flash is Read-Only (RO) protected
=>flinfo

Bank # 1: AMD AM29LV641MH (64 Mbit)
  Size: 16 MB in 128 Sectors
  Sector Start Addresses:
FE000000 (RO) FE020000 (RO) FE040000 (RO) FE060000 (RO) FE080000 (RO)
FE0A0000 (RO) FE0C0000 (RO) FE0E0000 (RO) FE100000 (RO) FE120000 (RO)
…………………….
FEF00000 (RO) FEF20000 (RO) FEF40000 (RO) FEF60000 (RO) FEF80000 (RO)
FEFA0000 (RO) FEFC0000 (RO) FEFE0000 (RO)
```

**Figure 27. Display Flash Protection Info**

```
=>protect off all
Un-Protect Flash Bank # 1
=>flinfo

Bank # 1: AMD AM29LV641MH (64 Mbit)
  Size: 16 MB in 128 Sectors
  Sector Start Addresses:
FE000000      FE020000      FE040000      FE060000      FE080000
FE0A0000      FE0C0000      FE0E0000      FE100000      FE120000
……………….
FEF00000      FEF20000      FEF40000      FEF60000      FEF80000
FEFA0000      FEFC0000      FEFE0000
```

Figure 28. Turn All Protection Off

- Flash is at 0xfe00_0000

```
=>md fe000000
fe000000: 27051956 552d426f 6f742031 2e312e32    '..Vu-boot 1.1.2
fe000010: 28546169 67615f56 325f7232 2920284a    (HPC2_V2_r2) (J
fe000020: 756e2020 31203230 3035202d 2031373a     un  1 2005 - 17:
fe000030: 33373a32 39294672 65657363 616c6520    37:29)Freescale
fe000040: 54736931 30382054 61696761 20526566    Tsi108 HPC2 Ref
fe000050: 6572656e 63652050 6c617466 6f726d20    erence Platform
fe000060: 28322e30 29000000 00000000 00000000    (2.0)..........
```

Figure 29. Display Current Contents of Flash

```
=>erase all
Erase Flash Bank # 1
Erasing in progress..please wait 1-2 mins    20
Erasing Done

=>md fe000000
fe000000: ffffffff ffffffff ffffffff ffffffff    ................
fe000010: ffffffff ffffffff ffffffff ffffffff    ................
fe000020: ffffffff ffffffff ffffffff ffffffff    ................
fe000030: ffffffff ffffffff ffffffff ffffffff    ................
```

Figure 30. Erase and Display Flash

- **Promjet is at 0xff00_0000**

```
=>md ff000000
ff000000: 27051956 552d426f 6f742031 2e312e32    '..Vu-boot 1.1.2
ff000010: 28546169 67615f56 325f7232 2920284a    (HPC2_V2_r2) (J
ff000020: 756e2020 37203230 3035202d 2031363a    un  7 2005 - 16:
ff000030: 33373a30 36294672 65657363 616c6520    37:06)Freescale
ff000040: 54736931 30382054 61696761 20526566    Tsi108 HPC2 Ref
ff000050: 6572656e 63652050 6c617466 6f726d20    erence Platform
ff000060: 28322e30 29000000 00000000 00000000    (2.0)...........
ff000070: 00000000 00000000 00000000 00000000    ................
```

**Figure 31. Display Contents of PROMjet (U-Boot Code)**

```
=>cp.w ff000000 fe000000 b100
Copy to Flash...
Writing in progress....
0x000004
Writing done

=>md fe000000
fe000000: 27051956 552d426f 6f742031 2e312e32    '..Vu-boot 1.1.2
fe000010: 28546169 67615f56 325f7232 2920284a    (HPC2_V2_r2) (J
fe000020: 756e2020 37203230 3035202d 2031363a    un  7 2005 - 16:
fe000030: 33373a30 36294672 65657363 616c6520    37:06)Freescale
fe000040: 54736931 30382054 61696761 20526566    Tsi108 HPC2 Ref
fe000050: 6572656e 63652050 6c617466 6f726d20    erence Platform
fe000060: 28322e30 29000000 00000000 00000000    (2.0)...........
```

**Figure 32. Copy U-Boot from PROMjet to Flash and Display It**

## 7.7 Starting Linux on U-Boot

The following steps will enable Linux to run on U-Boot:

1. Build Linux kernel on the host system
2. Copy kernel to /tftpboot on the host system
3. Download using tftp, the kernel to the HPC II board running U-Boot. This assumes that the network environment variables on U-Boot have been correctly initialized.
4. Start Linux

### 7.7.1 Building Linux on Host System

To obtain the source and build the kernel, in the top level directory,

```
make taiga_defconfig
```

followed by,

`make zImage` (for a hard drive root file system)

or

`make zImage.initrd` (for a RAM disk root file system)

To create a zImage.initrd.elf (kernel + RAM disk file system), it is assumed that the user has the file ramdisk.image.gz in arch/ppc/boot/images directory. The Linux source tarball on the Freescale website includes the ramdisk.image.gz file.

### 7.7.2 Ready the Kernel for Download

Copy the kernel image to the tftpboot directory (/tftpboot).

```
cp arch/ppc/boot/images/zImage.elf /tftpboot
```

or

```
cp arch/ppc/boot/images/zImage.initrd.elf /tftpboot
```

### 7.7.3 Download Kernel From Host System to HPC II

From the HPC II board, make sure all the environment variables are correctly initialized for a network download using the `printenv` command. At the U-Boot prompt:

```
tftp 200000 zImage.elf
```

or

```
tftp 200000 zImage.initrd.elf
```

### 7.7.4 Start Linux

```
go 210000
```

This should launch the kernel and use the specified file system. For the RAM disk file system, during Linux boot-up, change the arguments as follows:

```
root=/dev/sda3
```

to

```
root=/dev/ram
```

By default, the kernel uses the hard-drive file system.

Reference 1 provides the link to the web-page with source code.

## 8 References

The following documents are referenced in this application note:

1. HPC II User Guide Rev 1.1, http://www.freescale.com/powerpc, PowerPC Processors Evaluation Systems and Verification Platforms. The MCEVALHPC2-7448 page provides source code for both U-Boot and Linux, as well as the RAM disk file system.
2. Tsi108 Host bridge for PowerPC, Users Manual, http://www.tundra.com/Products/PowerPC/Tsi108/index.cfm
3. DENX U-Boot and Linux Guide, DENX Software Engineering, http://www.denx.de/twiki/bin/view/DULG/Manual
4. PROMjet Web page, http://www.emutec.com/pjetmain.html

For assistance or answers to any question on the information that is presented in this document, send an e-mail to risc10@freescale.com.

## 9 Document Revision History

Table 1 provides a revision history for this application note.

**Table 1. Document Revision History**

| Rev. No. | Date | Substantive Change(s) |
|---|---|---|
| 1.0 | 9/27/2005 | Initial public release. |

***How to Reach Us:***

**USA/Europe/Locations Not Listed:**
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405,
Denver, Colorado 80217
1-480-768-2130
(800) 521-6274

**Japan:**
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

**Home Page:**
www.freescale.com

AN2924
Rev. 1.0
09/2005