xCat/xCat2 Family

U-Boot User Manual

Doc. No. MV-S400237-00, Rev. D

December 23, 2012, Preliminary

CONFIDENTIAL

Document Classification: Proprietary Information

Marvell. Moving Forward Faster

Document Conventions

	Note: Provides related information or information of special importance.
!	**Caution:** Indicates potential damage to hardware or software, or loss of data.
ϟ	**Warning:** Indicates a risk of personal injury.

Document Status

Doc Status: Preliminary	Technical Publication: 0.x

For more information, visit our website at: http://www.marvell.com

Revision History

Table 1: Revision History

Document Revision	Software Version	Date	Comments
D	5.4.0.x	December 23, 2012	• Updated Section 6.3.1, SDK Installation and Section 6.3.2, Image Build Settings. • Updated Appendix A.2.2.1, Booting from UART, on page 34.
C	5.4.0.x	July 20, 2011	Content updates
B	5.3.4.x	December 26, 2010	Release
A	3.3x	September 30, 2009	Initial release

Table of Contents

1 Introduction 7
1.1 Related Documentation 7
1.2 Glossary 8

2 Marvell U-Boot Components and Features 10
2.1 I/O Interfaces 10
2.2 Memory Interface Support 11
2.3 Additional Features 11

3 Marvell® Software Architecture 12
3.1 Development Environment File Structure 13

4 OS-Independent Software Components 15
4.1 Device HAL Components 15
4.2 Drivers HAL Components 16
4.3 Basic Board Support Package Features 24
4.4 Marvell Hardware OS-Independent Libraries 24
4.5 Marvell OS Service Libraries 25

5 U-Boot OS Layer Components 26
5.1 Serial Interfaces 26
5.2 Network Interfaces 26
5.3 PCIe Components 26
5.4 Prestera Interface 27

6 System Software Configuration 28
6.1 Development Environment 28
6.2 System Modification 28
6.3 System Software Configuration 29

A System Setup 33
A.1 System Hardware Preparation 33
A.2 System Boot Preparation 33

B Memory Map 36

List of Tables

Table 1: Revision History 3
Table 2: Supported Hardware Components 7
Table 3: Supported Network Devices 10
Table 4: Marvell Hardware OS-Independent Libraries 24
Table 5: Marvell OS Services Libraries 25
Table 6: Supported Network Devices 30
Table 7: Script Files Loaded Per Board 35
Table 8: xCat/xCat2 Family Default Memory Map 36

List of Figures

Figure 1: Marvell Software Architecture 12
Figure 2: PCI HAL Software Architecture 19
Figure 3: MII HAL Software Architecture 23

1 Introduction

This manual describes the U-Boot deliverables and components, the API package, as well as additional drivers that utilize the xCat/xCat2 family CPU, LAN, and Serial interface capabilities.

In addition, this manual provides the information necessary to compile, run, and debug the U-Boot on a system based on the xCat/xCat2 family of devices, and consisting of the hardware components listed in Table 2.

In this manual, the xCat/xCat2 family of devices is referred to as “the device” or “the devices”.

Table 2: Supported Hardware Components

Hardware Component	Description
Marvell Board	DB-xCat-24GE-4GP DB-xCat-24FE-4GP RD-xCat-24GE-2SFP DB-xCat2-BGA-24GE DB-xCat2-BGA-48FE DB-xCat2-QFP-24GE DB-xCat2-QFP-24GE-2SFP

Before using a board with the U-Boot, verify that the Marvell® configuration is set on this board (refer to the relevant board documentation mentioned in Section 1.1).

1.1 Related Documentation

The following documents provide additional information about these development systems:

- DB-xCat-24GE-4GP – *Development Board User Guide* (Document Number MV-L100588-10)
- DB-xCat-24GE-4GP – *Board Configuration* (Document Number MV-L100588-11)
- DB-xCat-24FE-4GP – *Development Board User Guide* (Document Number MV-L100590-10)
- DB-xCat-24FE-4GP – *Board Configuration* (Document Number MV-L100590-11)
- RD-DX3121-24G-2SFP – *Reference Design User Guide* (Document Number MV-L100586-10)
- RD-DX3121-24G-2SFP – *Board Configuration* (Document Number MV-L100586-11)
- DB-xCat2-24GE-4GP – *Development Board User Guide* (Document Number MV-L100754-10)
- DB-xCat2-48FE-4GP – *Development Board User Guide* (Document Number MV-L100755-10)

1.2 Glossary

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Product Interface
BAR	Base Address Register
BAT	Block Address Translation
BSP	Board Support Package
CAS	Column Access Strobe
CBC	Cipher Block Chaining
CESA	Cryptographic Engine Security Acceleration
DAS	Direct Attached Storage
DDR SDRAM	Double-Data-Rate Synchronous Dynamic Random Access Memory
DES	Data Encryption Standard
DIMM	Dual In-line Memory Module
DIP	Destination IP Address
DMA	Direct Memory Access
ECC	Error Correcting Code
END	Enhanced Network Driver
EPIC	External Programmable Interrupt Controller
EPS	External Product Specification
FEI	Fast Ethernet Interface
FFS	Flash File System
GEI	Gigabit Ethernet Interface
GND	Generic Network Driver
GPP	General Purpose Pins
HAL	Hardware Abstract Layer
HID	Human Interface Device
ISR	Interrupt Service Routine
JFFS2	Journalling Flash File System, Version 2
LSP	Linux Support Package
MGI	Marvell Gigabit Ethernet Interface
MPP	Multi-Purpose Pin register
MMU	Memory Management Unit
MTD	Memory Technology Device
MTU	Maximum Transmission Unit
NAS	Network Attached Storage
NAPT	Network Address and Port Translation
NAT	Network Address Translation
NFP	Network Fast Processing
NIC	Network Interface Card
NOE	NIC Over Ethernet Port
OOB	Out of Band
PCI	Peripheral Component Interconnect
PCIe	PCI Express

Acronym	Definition
PID	Platform Industrial Device
PNP	Plug and Play
PP	Packet Processor (switch)
RTC	Real Time Clock
S@R	Sampled at Reset
SDMA	Serial Direct Memory Access
SIP	Source IP Address
SoC	System On Chip
SPD	Serial Presence Detect
SPI	Serial Peripheral Interface
TFTP	Trivial File Transfer Protocol
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VoIP	Voice Over IP
WB	Write Back
VTS	Validation Test Suite
WT	Write Through

2 Marvell U-Boot Components and Features

This section describes the components of the U-Boot software package as well as the supported and unsupported features.

2.1 I/O Interfaces

2.1.1 Serial UART

The software package supports the Marvell integrated NS16550 Universal Asynchronous Receiver/Transmitter (UART) interface. This interface supports a standard PC baud rate—up to 115200 on both channels.

2.1.2 Network

This software package supports the following network devices:

Table 3: Supported Network Devices

Network Device	Interface Name	Module File Name
xCat Family GbE0/1 xCat2 Family GbE1	egiga	mvEgiga
xCat/xCat2 Packet Processor – through CPU Port/SDMA	ppsdma/ppmii	mii/mvPrestera

For more information about these network interfaces, refer to Section 5.2, Network Interfaces, on page 26.

2.1.3 Ethernet PHY Support

The U-Boot software package supports the 88E1111 and 88E1116 Marvell Ethernet PHY for the MGI interface.
RD-xCat-24G-2SFP and DB-xCat-24GE-4GP Rev 2 use the 88E1116 Marvell Ethernet PHY.
DB-xCat-24FE-4GP and DB-xCat-24GE-4GP Rev 1 use the 88E1111 Marvell Ethernet PHY.

2.1.4 Packet Processor PHY Support

The packet processor uses the Marvell Ethernet PHY, which is configured via the EEPROM and can also be configured by the software.

2.1.5 PCIe Interface

The software utilizes the xCat Family PCIe stack located under the U-Boot source tree to implement the PCIe scans and configuration.

2.1.6 USB

The xCat/xCat2 USB controller can be used as a USB device/host controller. Although these modes are not enabled in the U-Boot, it configures the PHY with the working configuration.

2.2 Memory Interface Support

DDR

The xCat/xCat2 Family BootROM initializes the DRAM according to a header located on its boot device.

This header contains the data that allows the BootROM to configure the DRAM windows, timing, parameters, and other settings.

2.3 Additional Features

2.3.1 Clock Rate

The xCat/xCat2 device has:

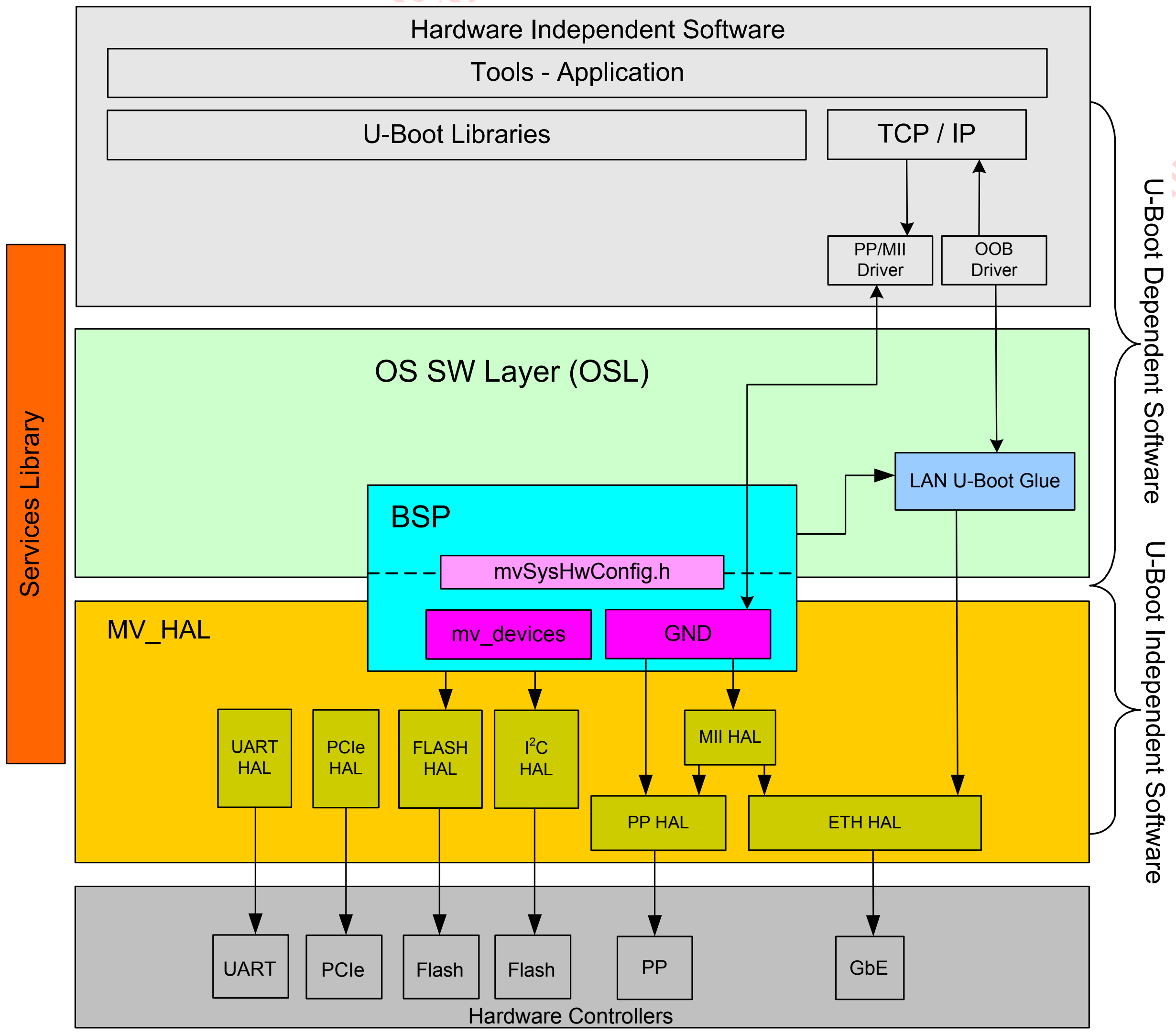
- CPU clock – Used for the CPU core
- System clock – Used for the DDR interface
- TClk – Used for the system controller clock

The rates of all of these clocks are calculated by using the Sampled at Reset register.

3 Marvell® Software Architecture

The Marvell® software architecture supports all hardware functionalities, provides a user-friendly interface, and supports multiple operating system environments. This software architecture supports the dynamic hardware and software environment by providing a modular software structure and APIs.

Figure 1: Marvell Software Architecture



This architecture divides the software package into the following layers and libraries:

Hardware Abstraction Layer (HAL)	Implementation of OS-independent and hardware-dependent APIs defined by Marvell. HAL implementation code per hardware unit is called **HW Lib**.
OS Software Layer (OSL)	Each operating system defines hardware-dependent APIs (Linux Support Package (LSP) APIs and Driver APIs). The Marvell OSL hardware-independent layer implements each operating system API using the Marvell HAL APIs. OS Glue is the OSL implementation per interface (for example, network interface). The traditional device driver consists of the OS Glue and HAL Libraries code (see below).
OS Services Library	The OS service library provides the HAL with a common interface to OS services. Per OS, this library enables the HAL to utilize OS services, such as: • I/O (for example, printf) • The OS-architecture-dependent support (for example, interrupt lock, MMU services)
Common Library	This library is accessible to all layers (including OSL), introducing common software APIs.
System Hardware Configuration File	This header file defines the system's global hardware configuration, for example, the system memory map. Whenever a change is made in any layer, this file must be checked for updates.
Generic Network Driver (GND)	This is the software component that implements the network driver that is independent of the OS and Hardware layers. It is initialized with two sets of callbacks—one for OS operations (receive, send, etc.) and the other for hardware operations.

The Marvell software architecture can also accelerate the customer software development as follows:

- The board's structural information is located in **\boardEnv\mvBoardEnvSpec.c** and **\boardEnv\mvBoardEnvSpec.h**. These files define the board hardware elements. Customers who design their own boards can easily identify the element they need to modify, and then port the Marvell software to their own board software.
- The HAL layer's OS independence is achieved by using the **OS services** library. Customers who have a proprietary OS can easily port the Marvell software to their operating system by modifying the contents of the **OS Services** directory.

3.1 Development Environment File Structure

3.1.1 mv_hal Directory

This directory includes implementation of the controller and board HAL library defined by the Marvell software architecture.

The HAL is a container for all hardware libraries for the controller interfaces and board devices. These libraries contain OS-independent APIs representing the specific functionality of the device, for example, DDR SDRAM, Flash, and PCIe (in the xCat devices).

The HAL depends on a board implementation. Each modification to the board hardware results in modification in the HAL layer.

3.1.1.1 boardEnv Board Environment Directory

This directory (located under the **<controller>_family** directory) includes a software library that provides an OS-independent software interface that is identical to the hardware functions of all Marvell boards. It also includes board-specific configuration data, such as an I^2C device bus address and PCIe interrupt routing to GPP pins.

The **mvBoardEnvSpec.c** file contains a single `MV_BOARD_INFO` structure for each of the supported Marvell boards. The header file contains the device's board IDs and the MPP configuration values.

3.1.2 mv_kw\kw_family Directory

This directory includes the OS-independent implementation pertaining to xCat/xCat2. The directory includes the board environment HAL and the controller environment HAL.

3.1.2.1 sys Directories

These directories are located under the **ctrlEnv** directory. Each file includes a HAL library that initializes the address decode windows for a specific SoC integrated interface. These files contain OS-independent APIs providing the device-specific functionality. They include the CPU interface, PCIe, Ethernet, CESA, and others.

3.1.3 common Directory

This directory implements the common library defined by the Marvell software architecture. It includes software elements that are common to all Marvell software modules, and contains the following:

- Common software utilities (for example, byte swap)
- Debug utilities
- Marvell typedefs and returned status
- Common macros
- Errata control mechanism

3.1.4 USP Directory

This directory contains the implementation of drivers that rely on the xCat HAL libraries:

- Egiga
- Switch Ethernet ports
- I^2C (TWSI)
- UART, etc.

3.1.5 Tools\doImage Directory

This directory contains a source code for building **doImage** to perform booting using the bootROM. The code is compiled using Marvell's SDK 1.2.4. **doImage** is responsible for creating the xCat/xCat2 bootable image that consists of a boot header (refer to the chip datasheet) and the U-Boot binary image itself.

Note

The DDR configuration register is part of the boot header.

3.1.6 pssServices Directory

This directory contains PSS/CPSS-related services and APIs.

4 OS-Independent Software Components

This section describes the OS-independent Hardware Abstraction Layer (HAL) components provided in the software package.

Note

Each HAL is relevant only when the controller has an integrated unit or the board has the relevant device.

4.1 Device HAL Components

4.1.1 Board Environment HAL

The board environment HAL software is responsible for defining the board properties. This is done by providing a set of functions that return the board properties, such as the board revision and system clock rates. The board environment HAL software also provides a set of macros that describe the board topology, for example, PCIe interrupts and wiring.

Supported Features

The HAL software:

- Initializes the board environment.
- Initializes the chip select parameters of external devices connected to the board, such as RTC and Flash.
- Gets the board ID number, model, revision, and name.
- Gets Ethernet PHY addresses.
- Auto-detects the system clock rate (if supported).
- Gets the board MPP configuration.
- Gets the GPIO interrupt wiring scheme information (PCIe, RTC, and others).

4.1.2 Controller Environment HAL

The controller environment HAL software is responsible for defining the controller properties. It provides a set of functions that return the controller properties, such as model, name, and revision. In addition, the Controller Environment HAL provides a set of macros that access the controller registers, and define a set of common enums that can be used in other HAL units.

Supported Features

The following features are supported:

- Initializing the controller environment by initializing Multi-Purpose Pin (MPP) registers.
- Initializing internal interface address decode windows.
- Control over the CPU address decode windows.
- Getting various controller implementation information, such as:
  - Number of supported PCIe interfaces
  - Number of supported Gigabit Ethernet controllers
  - Controller model

- Controller revision (silicon revision)
- Controller name (string)

4.1.2.1 SoC Types Support (Feature Matrix)

Part of the controller environment HAL is Feature Matrix, which is used to enable/disable features not supported by the current SoC type, or used to set different configurations to supported features. The SoC type is detected dynamically and is used by the board and controller environment HALs. This is also used in OS-dependent code, such as interrupts and network configuration.

4.1.3 Generic Network Driver (GND) HAL

GND is an OS and hardware independent network driver. It has a set of callbacks for the driver and the hardware.

The work performed by GND can be described in terms of processing packets and/or buffers, without specific knowledge of the following:

- What hardware (GbE controller of SoC or PP) a packet is received from/transmitted to. For more information on the network stack (kernel module), refer to *AN-302, "Packet Flow Through the Linux Kernel on Devices Using the Prestera® xCat Family of Packet Processors"*.
- What driver (VxWorks END or Linux net_device driver) a packet is received from/forwarded to.
- What kind of synchronization is used (semaphores, interrupt locks, task locks, etc.).

Supported Features

The following features are supported:

- Dynamic configuration of:
  - Maximum number of buffers in the frame
  - Rx buffer size
  - Rx buffer size offset (to store useful information from a driver)
  - Number of Rx/Tx queues and number of Rx/Tx descriptors per queue
  - Tx mode type—synchronous or asynchronous
- Sending a packet to the appropriate HAL (and waiting until the Tx is performed, if the synchronous Tx is used)
- Extracting the received frames from every Rx queue and forwarding them to the driver
- Extracting the received frame from a specific Rx queue and returning it to the caller

Unsupported Features

The Rx processing mechanism does not support a prioritization scheme. The Rx queues are processed from 0 to 7 in a sequence. If a priority scheme is needed by the driver, it uses the GND API, which allows it to receive an Rx frame from a specific Rx queue. Thus, priorities can be implemented at the driver level.

4.2 Drivers HAL Components

4.2.1 Counters/Timers HAL

The counters/timers HAL software provides a simple software API to the counters/timers unit in the controller.

Supported Features

The following features are supported:

- Timer and counter modes

- Set of functions that implement the following counter/timer unit features:
  - Load an init value to a counter/timer.
  - Read the value of a counter/timer.
  - Get and set the Control Register of a counter/timer.
  - Enable and disable a counter/timer.
  - Start a counter/timer according to the user's configuration.
  - Get a timer frequency.

Unsupported Features

This driver does not include counter/timer management—the user is responsible for the state of the counter. For example, the user must verify that the counter/timer is disabled before enabling it. This is done to increase the API performance by reducing a timer state query in each API.

4.2.2 DDR SDRAM Interface HAL

The DDR SDRAM HAL software provides an API to retrieve information regarding the DDR SDRAM that is installed on the board.

Supported Features

- Getting a DDR SDRAM bank size and base address

Unsupported Features

- DDR 2 interface automatic configuration according to system parameters
- Finding the best DDR SDRAM configuration that fits the system timing

4.2.3 General Purpose I/O Port (GPP) HAL

The General Purpose I/O Port HAL software provides a simple API to GPP unit.

Supported Features

The following features are supported:

- Control GPP pin mode (input or output)
- Get/set GPP pin value
- GPP data in polarity

Unsupported Features

General-purpose I/O Port interrupts are not supported in the HAL.

4.2.4 Gigabit Ethernet Controller HAL

The Gigabit Ethernet Controller HAL software provides an API for the xCat/xCat2 family integrated Gigabit Ethernet Controller capabilities. Together with the OS-dependent software layer, the network driver is implemented for Gigabit Ethernet.

Supported Features

The following features are supported:

- Operating in zero copy mode, in which no data copying is performed during data flow operations
- Cached descriptors
- Scatter-Gather on transmit operations
- Rx and Tx interrupts coalescing

- Full control over Unicast and Multicast MAC configurations
- Auto-Negotiation for Speed, Duplex, and Flow Control
- RMON and Interface MIB counters
- Jumbo packets
- Provides OS-independent APIs for upper layers:
  - Gigabit Ethernet port initialization (for example, port init, up/down, setDefaults)
  - Gigabit Ethernet port data flow (for example, port send, receive)
  - Gigabit Ethernet port Rx MAC filtering control
  - Gigabit Ethernet port interrupt coalescing control
  - Gigabit Ethernet port MIB counters
  - Gigabit Ethernet PHY operations
  - Gigabit Ethernet debug functions

Unsupported Features

The following features are not supported:

- Multiple Rx or Tx queues per port
- IP, TCP checksum offloading
- Scatter-Gather on Rx operations

4.2.5 Ethernet PHY HAL

The Ethernet PHY HAL software provides an API for accessing the board Ethernet PHY and Ethernet switches registers.

Supported Features

The following features are supported:

- Reading from and writing to PHY registers
- Resetting a PHY
- Restarting the LAN on a PHY
- Checking the link
- Checking the status

4.2.6 NAND Flash HAL

The NAND Flash HAL software is responsible for detecting the NAND Flash type, and for performing basic operations on the Flash according to its type. These basic operations are:

- Erasing
- Writing
- Reading from a portion of the Flash
- Reading from all of the Flash

4.2.6.1 Supported NAND Flash Vendors and Type

Currently, the Flash driver supports only Samsung 8-bit and 16-bit devices.

The Flash driver is designed for the following Flash structures:

- 512 bytes page size
- 16 bytes spare area

Supported Features

The following features are supported:

- Detecting the following properties of the Flash according to the Flash ID (auto-detect) and bus width:
  - Size of the Flash
  - Number of Flash blocks
  - Page data size
  - Spare area size
  - Number of page per block
- Page read and sequential read where available
- Page spare read and sequential read where available
- Page program
- Page copyback program
- Block erase
- Block lock, tight lock and unlock
- ECC algorithm for read/write operation
- User can choose to use its own data read and write functions. The default read and write are implemented using the CPU, but the user can implement them using the DMA engines to improve performance.

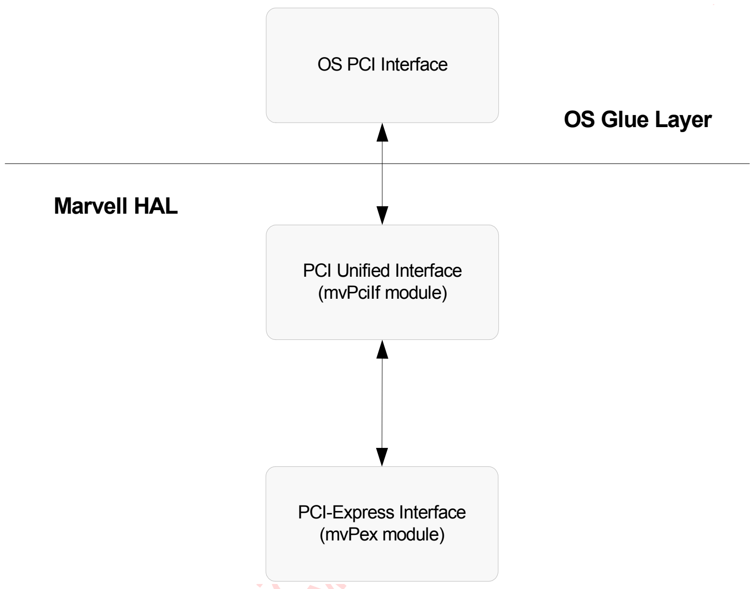
4.2.6.2 Unsupported Features

The Block replacements algorithm is not supported.

4.2.7 PCI Interface (pciIf) HAL

The xCat Family PCI Interface (pciIf) HAL software provides a unified PCI API for all the PCIe interfaces for initializing and implementing PCI unit features (see Figure 2).

Figure 2: PCI HAL Software Architecture


OS PCI Interface
OS Glue Layer
Marvell HAL
PCI Unified Interface (mvPciIf module)
PCI-Express Interface (mvPex module)

The OS-dependent PCI stack does not need to be aware of the underlying PCI type (PCIe).

Supported Features

The following features are supported:

- Initializing the PCIe. According to the PCIe interface number, the correct PCIe init routine is called.
- Configuration read and write (4 bytes only).
- Setting Master and Slave modes to a device.
- Getting PCI interface type (PCIe or conventional/PCI-X).
- Getting/setting a device number and bus number of a local interface.

4.2.8 Serial Flash HAL

This Flash driver provides the user with the capability to detect, configure, read, write, and erase an SPI serial Flash. This device driver provides the Flash command layer for the SPI Flash protocol and uses the SPI controller driver in communicating with the external Flash device.

Supported Features

The following features are supported:

- Detecting and initializing a serial (SPI) Flash
- Performing erase (chip or sector)
- Retrieving Flash and manufacturer IDs
- Reading and writing a block of data
- Selecting the write protect region
- Enabling/disabling the write protect region

Supported Serial Flash Types

- ST M25P32 SPI Flash, 4MB, 64 sectors of 64K each
- ST M25P64 SPI Flash, 8MB, 128 sectors of 64K each
- ST M25P128 SPI Flash, 16MB, 64 sectors of 256K each
- Macronix MXIC MX25L6405 SPI Flash, 8MB, 128 sectors of 64K each
- Macronix MXIC MX25L25635 SPI Flash, 32MB, 512 sectors of 64K each
- SPANSION S25FL128P SPI Flash, 16MB, 64 sectors of 256K each

4.2.9 SPI HAL

The integrated Serial Peripheral Interface (SPI) controller has a programmable baud rate. It is capable of transmitting and receiving 16/8-bit chunks of data simultaneously in both interrupt and polling modes.

Supported Features

The following features are supported:

- Initializing the SPI controller
- Managing the CS signal
- Performing read, write, and read with write cycles
- Direct and Indirect SPI read/write

4.2.10 PCI Express HAL

The PCI Express HAL software provides an API for the xCat family PCIe interfaces, for initializing and implementing the PCIe interfaces. For configuration cycle operations, use the PCI interface API (refer to Section 4.2.7, PCI Interface (pciIf) HAL) that is a unified configuration API for both PCI conventional/PCI-X and PCI-Express.

Supported Features

The following features are supported:

- PCIe interfaces
- Initialization of the PCIe interface:
  - Initializes the PCIe-to-target address space (BARs and windows). This will allow PCIe access into the xCat peripheral targets (for example, DDR SDRAM)
  - Sets local interface bus and device numbers according to the user configuration
  - Enables local interface master (master enable) and slave (memory and IO access enable)
- Configuration read and write (4 bytes only)
- Access control windows API
- Get/set device number and bus number of the local interface
- Get/set and remap a PCIe address space window (remaps the address going from PCIe into xCat internal targets)
- PCIe PHY interface shutdown
- Supports 2 modes for PCIe interface 1X4 or 4 interfaces X1

Unsupported Features

PCIe interface interrupts (error report) – interrupts are not supported in the HAL.

4.2.11 Prestera HAL

The Prestera HAL software provides an API for access to the xCat/xCat2 device, and provides an interface for network support on the packet processor ports.

Supported Features

The following features are supported:

- Reading from/writing to the xCat/xCat2 memory – the HAL support address completion and base address.
- Getting port MIB counters.
- Receiving and transmitting packets through the packet processor ports, using the switch SDMA mechanism.

4.2.12 I^{2}C HAL

The I^{2}C HAL software provides an API for the I^{2}C (TWSI) interface, so it can act as a master generating read/write requests, and as a slave responding to read/write requests.

Supported Features

The following features are supported:

- Initializing the I^{2}C interface:
  - Setting the serial clock baud rates
  - Slave address 7-bit or 10-bit and the serial clocks
  - Enabling the I^{2}C slave
- Resetting the I^{2}C unit.
- Simple I^{2}C read/write transaction from/to a slave:
  - With 7-bit or 10-bit addresses
  - With more or less than 256 bytes address space
  - From/to a specific offset
  - While hiding the protocol requirement from the user

- Support for multiple I^2C channels by adding channel number to all I^2C HAL driver APIs

Unsupported Features

I^2C interrupts are not supported in the HAL.

4.2.13 XOR HAL

The XOR HAL software provides an API to the XOR unit for initializing and implementing the XOR unit features.

Supported Features

Initializing the XOR engine unit by:

- Setting all XOR engine unit target windows
- Setting the XOR engine channels Command register
- Functions that set, get, and enable an XOR engine channel target window
- Functions that set an XOR engine channel protection window
- Functions that set an XOR engine control register
- Functions that implement XOR engine ECC, MemInit, XOR, CRC and DMA modes
- Functions that set the XOR engine PCI Remap register
- Functions that set an XOR engine channel Control registers
- Functions that get the current state of an XOR engine channel
- Functions that perform a Data transfer using an XOR engine channel

Unsupported Features

- XOR channel interrupts. Interrupts are not supported in the HAL.
- This HAL software API requires source and destination addresses to be in a CPU bus format (physical address). This means that when using the API in chain mode, the user must prepare the descriptor chain so the next descriptor pointer is in a physical format.
- This HAL software assumes XOR descriptors comply with the alignment restrictions.

4.2.14 UART HAL

The UART HAL software provides an API to UART unit for initializing and implementing the UART unit features.

This library introduces the API that controls the xCat/xCat2 family serial channel operations, such as channel init, start, reset, baud-rate setting, and data Rx/Tx.

4.2.15 USB HAL

The xCat device contains a USB 2.0 compliant controller and an embedded USB 2.0 PHY. The USB PHY can be activated either as a USB Device or as a USB Host. It is configured by the boot parameters.

In the USB Host mode, the xCat USB Controller is fully EHCI compliant, and it does not need a special driver. Any operating system that includes a USB Host software stack supporting the EHCI standard can work with the xCat USB controller with minor hardware changes.

Supported Features

The following features are supported:

- As a USB Host, the xCat family USB controller connects to all USB Device types (HS, FS).
- As a USB Device, the xCat family USB controller connects to all USB Host types (HS, FS) and hubs.

- Four independent endpoints support control, interrupt, bulk, and asynchronous data transfers.
- 480 Mbps High Speed (HS /12 Mbps FS, FS only, and LS only) 1.5 Mbps serial data transmission rates.

USB Device Software Deliverables

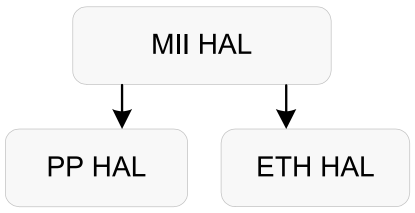
The U-Boot software package supporting the xCat Family development board includes the following USB components:

- **mvUsbRegs.h** - This file contains USB PHY, Bridge, and Core registers.
- **mvUsb.h** - This file contains general definitions and function prototypes to configure the USB component.
- **mvUsb.c** - This file contains implementation of the function needed to configure the USB component.

4.2.16 MII HAL

The MII HAL software provides an interface for network support through the CPU port of the packet processor.

Figure 3: MII HAL Software Architecture



Supported Features

The following features are supported:

- Getting the port's MIB counters.
- Receiving and transmitting packets through the packet processor ports, using the packet processor CPU port mechanism.

Note

Reading from and writing to the packet processor memory range is performed through the Prestera HAL.

4.3 Basic Board Support Package Features

4.3.1 CPU Types

The supported CPU is ARM926EJ-S CPU.

4.4 Marvell Hardware OS-Independent Libraries

Table 4 lists the Hardware OS-independent libraries.

Table 4: Marvell Hardware OS-Independent Libraries

Hardware Library	Description	Section
Board Environment	Provides a software interface to the hardware functions of all Marvell boards.	4.1.1 "Board Environment HAL"
Controller Environment	Provides a software interface to the xCat/xCat2 family controller internal design data (such as the number of unit engines) and includes common APIs used by other controller internal units' hardware libraries.	4.1.2 "Controller Environment HAL"
Generic Network Driver (GND)	GND is the network driver independent of the OS and hardware. It has a set of callbacks for the driver and the hardware.	4.1.3 "Generic Network Driver (GND) HAL"
cntmr	Provides a software interface to the xCat/xCat2 family Counter/Timer unit.	4.2.1 "Counters/Timers HAL"
DDR2	Provides a software interface to the board's DRAM.	4.2.2 "DDR SDRAM Interface HAL"
GPP unit	Provides a software interface to the xCat/xCat2 family GPP unit.	4.2.3 "General Purpose I/O Port (GPP) HAL"
Gigabit Ethernet unit	Provides a software interface to the xCat/xCat2 family Gigabit Ethernet unit. The Gigabit network interface is implemented together with an OS-dependent driver.	4.2.4 "Gigabit Ethernet Controller HAL"
Ethernet PHY	Provides a software interface to the on-board Ethernet PHY.	4.2.5 "Ethernet PHY HAL"
NAND Flash	Provides a software interface to the NAND Flash device.	4.2.6 "NAND Flash HAL"
SFlash	Provides a software interface to serial Flash devices.	4.2.8 "Serial Flash HAL"
SPI	Provides a software interface to the SPI Flash device.	4.2.9 "SPI HAL"
PCIe	Provides a software interface to the xCat family PCIe interface.	4.2.10 "PCI Express HAL"
prestera	Provides a software interface to access the Prestera chip and network interface support on the PP ports and CPU port.	4.2.11 "Prestera HAL"
TWSI	Provides a software interface to the xCat/xCat2 family I^{2}C interface.	4.2.12 "I2C HAL"

Table 4: Marvell Hardware OS-Independent Libraries (Continued)

Hardware Library	Description	Section
XOR unit	Provides a software interface to the xCat/xCat2 family XOR unit.	4.2.13 "XOR HAL"
UART	Provides a software interface to the serial communication unit and introduces the serial channel API.	4.2.14 "UART HAL"
USB	Provides PHY initialization to the xCat family embedded USB controller.	4.2.15 "USB HAL"
mii	Provides a software interface to access the Prestera CPU port.	4.2.16 "MII HAL"

4.5 Marvell OS Service Libraries

This software component enables other OS-independent software components to use basic, generic OS services (such as printf, malloc, free). This module defines generic APIs that must be implemented for each OS. All APIs implemented by the OS Service library can be divided into the groups in Table 5.

Table 5: Marvell OS Services Libraries

Software Features	Description
Memory management	Functions used to allocate and free different memory areas (virtual/physical, cached/uncached)
Standard I/O functions	OS-dependent "printf" implementation
CPU registers access	Access to CPU specific registers
Packet processor register access	Access to packet-processor-specific registers

5 U-Boot OS Layer Components

5.1 Serial Interfaces

5.1.1 Integrated NS16550 UART Interface

U-Boot supports a UART interface. The integrated NS16550 UART interface driver, located under the U-Boot HAL source tree, supports both channels UART 0/1 with any baud-rate of up to 115200 bps.

5.2 Network Interfaces

5.2.1 Marvell Integrated Gigabit Network Interface

This driver implements the U-Boot Network Interface driver over the Marvell® Gigabit Ethernet Controller. The driver utilizes the Marvell integrated Gigabit Ethernet Controller HAL (ETH HAL) library, described in Section 4.2.4, Gigabit Ethernet Controller HAL, on page 17, to implement the network interface.

Supported Features

- Zero Buffer Copy methodology – No data is copied during either Rx or Tx processes
- RX/Tx Scatter-Gather – When the driver receives a chain of buffers to transmit, it can perform Gather-Write. It does not need to copy any data.
- Multicast address filtering
- Promiscuous or non-promiscuous modes
- MIB counters
- Easy to use API to manipulate the MAC address (using the Environment Variable parameter)

Note

The IP/TCP checksum offloading and Interrupt mode (only polling) are not supported.

To configure this interface, refer to Section 6.3.4, Network Interfaces Configuration, on page 30.

5.3 PCIe Components

This driver implements the U-Boot PCIe Interface and utilizes the Marvell integrated PCI and PCIe HAL library (refer to Section 4.2.7, PCI Interface (pciIf) HAL, on page 19 and Section 4.2.10, PCI Express HAL, on page 20).

The main role of the PCI stack is to perform the following:

- Allocate memory and I/O resources for the system's existing PCI devices, according to their needs.
- Configure PCI-Express BARs and windows (Configuration Cycle).
- Provide functions that find a PCI device according to its Vendor and Device ID.

In addition, the PCI stack provides the following functionalities:

- Displays the PCI devices that are present in the system and their details.
- Configures a device as Root Complex/EndPoint.
- Changes the remap of a PCIe window.

5.4 Prestera Interface

5.4.1 Packet Processor Ports Interface

The U-Boot supports an interface used for network support on the packet processor ports. This interface enables reception and transmission of packets through the packet processor ports, using the packet processor SDMA of CPU Port mechanism. This interface is compatible with the U-Boot driver APIs. The driver utilizes the Prestera HAL and MII HAL.

6 System Software Configuration

6.1 Development Environment

6.1.1 U-Boot Base Version

Two base U-Boot versions are supported—1.1.4 and 2009.08. For each of them, the Marvell U-Boot release is a patch over the U-Boot base version source code.

U-Boot 1.1.4 and U-Boot 2009.08 can be downloaded from ftp://ftp.denx.de/pub/u-boot/.

6.1.1.1 SDK

The U-Boot is compiled using the Marvell SDK 1.2.4 over Linux.

The SDK contains development tools that run on the host machine. It supports both Little Endian and Big Endian.

Note

The U-Boot Little Endian supports booting of both Little and Big Endian for the BSP and VxWorks.

The SDK supplies a set of tools that are needed during the U-Boot build process:

gcc	Cross-compiler
objdump	This utility creates the map file (*.map) and disassembly file (*.dis).
objcopy	This utility creates the image file (*.bin) out of the elf file.

For more information, refer to the SDK Readme file.

6.2 System Modification

The system configuration is modified via the environment variables.

Some of the interfaces supported by the U-Boot depend on the environment variables. For example, the MAC address of the Gigabit Ethernet interface is taken for the environment variable *ethaddr*.

- To display values of all the environment variables, run: ***printenv***.
- To display a specific environment variable's value, run: ***printenv [VAR_NAME]***
- To set a variable's value, run: ***setenv <variable> <value>***. For example, to set the system IP address to 10.0.0.5, run: ***setenv ipaddr 10.0.0.5***
  Note: If <value> contains more than one word, use single quotes to surround the value. For example: ***setenv <variable> '<value>'***.
- To delete a variable, run: ***setenv <variable>***. For example, to delete the system's IP address variable, run: ***setenv ipaddr***
- To save the modifications, run: ***saveenv***.
- To restore the default values of all variables, run: ***resetenv***.
  Note: This operation is irreversible. You cannot restore the previous variable values.

6.3 System Software Configuration

This section describes the software configuration elements that control various types of system behavior. The configuration is set at compilation time and at runtime.

6.3.1 SDK Installation

The following Marvell SDK packages are used to create a U-Boot image.

- **sdk-armel-1.2.4** and **sdk-armel-3.2** are used for Little Endian.
- **sdk-armeb-1.2.4** and **sdk-armeb-3.2** are used for Big Endian.

These software packages must be installed on a Linux station.

For installation instructions, refer to the SDK ReadMe.html file.

6.3.2 Image Build Settings

This software package release supports building an image using the Marvell SDK development tools running on a Linux station.

To build an image:

1. Open a terminal.
2. Clean the previous compilation by running the ***make mrproper*** command.
3. Set the Makefile configuration. To do so, use one of the following options:
   - When using SDK 1.2.4, run:
     make xCat_total_build_spi
     or
     make xCat_total_build_nand
   - When using SDK 3.2, run:
     make xCat_total_build_spi TOOLCHAIN=3.2
     or
     make xCat_total_build_nand TOOLCHAIN=3.2
4. Build the image by running ***make***. Two **bin** files are created:
   - **<image>_nand.bin** or **<image>_spi.bin** to be burned on the Flash
   - **<image>_nand_uart.bin** or **<image>_spi_uart.bin** to be used when booting from UART

To build the U-Boot with the Big Endian configuration, run the following commands:

make mrproper

make xCat_total_build_nand_endian BE=1

or

make xCat_total_build_spi_endian BE=1

Note

DRAM parameters are located in the BootRom header attached to the image at the end of the build process. These parameters are taken from a text file, which is set in the parameter `MV_DDR_FREQ` in the Makefile.

6.3.3 System Address Space Configurations

The **mvSysHwConfig.h** file contains macros that define address space windows for each of the Marvell peripherals, such as DRAM banks, SPI, NAND, packet processor, and PCIe. The address space window is defined by a base address macro (for example, `SDRAM_CS0_BASE`) and a size macro (for example, `SDRAM_CS0_BASE`).

This software package also includes show routines per each xCat/xCat2 device interface (*xxxAddrDecShow()*) and for the entire device (*mvCtrlAddrDecShow()*)

To present the device mapping, run the ***map*** command in the U-Boot shell.

6.3.4 Network Interfaces Configuration

This section describes the various configurations used to modify the following supported U-Boot network interface behavior.

Table 6: Supported Network Devices

Network Device	Interface Name	Module File Name
xCat/xCat2 Family Gigabit Ethernet controller device (external port, OOB)	egiga	mvEgiga
xCat/xCat2 switch Gigabit Ethernet ports (internal port, inband)	ppsdma/ppmii	mvPrestera/mii

Adding/Removing a Network Interface

Each of the above interfaces can be added to/removed from the U-Boot image support.

To enable the Egiga driver, add *#define MV_INCLUDE_GIG_ETH*. To disable the driver, un-define it.

To enable the Prestera driver, add *#define MV_PRESTERA_SWITCH*. To disable the driver, un-define it.

The following are the interface macros for supported network interfaces:

- `MV_INCLUDE_GIG_ETH` – A macro for the Marvell integrated Gigabit Ethernet controller
- `MV_PRESTERA_SWITCH` – A macro for the xCat switch Gigabit Ethernet ports

6.3.5 Integrated Gigabit Ethernet (GbE) Configuration

Setting GbE MAC

Each xCat/xCat2 family GbE port is assigned a random MAC address by the software.

The port's MAC address can be modified using the environment variables.

1. Start the system. When the countdown starts, press any key to get the U-Boot prompt.
2. Run ***setenv ethaddr <byte0:byte1:byte2:byte3:byte4:byte5>***.
3. To save the parameter to the Flash, run ***saveenv***.
4. Restart the system.

For example, to assign Ethernet MAC 00:11:22:33:44:55, run:

setenv 00:11:22:33:44:55

saveenv

Note

The MAC address change takes effect only after restarting the system.

Assigning an IP Address to a GbE Port

Each xCat family GbE0/1 port and xCat2 family GbE1 port is assigned a default IP address by the software—**10.4.50.165** (defined by `CONFIG_IPADDR` in **mv_kw.h**).

The default IP address can be modified using the environment variables.

1. Start the system. When the countdown starts, press any key to invoke the U-Boot prompt.
2. Run ***setenv ipaddr <byte0.byte1.byte2.byte3.>***.
3. To save the setting, run ***saveenv***.

For example, to assign IP address 10.0.0.5, run:

setenv 10.0.0.5

saveenv

Configuring xCat Switch GbE Ports

The switch Ethernet ports require the same configuration as the GbE, namely—the MAC address and IP address configured on the GbE are also used by the switch Ethernet ports (all ports use the same configuration).

Setting Primary Ethernet Port

By default, 'egiga0' (GbE) is used as the default Ethernet port, and therefore it is the active port.

To change the default port, set the *ethprime* environment variable, save it, and restart the system.

For example, to configure U-Boot to use PP network ports through the PP-SDMA interface, run:

setenv ethprime ppsdma

saveenv

restart

This change only takes affect only after restarting the system.

Note

For the xCat2-QFP chip only: To download and burn the U-Boot image, use the ppsdma interface, because the OOB port is not available on supported boards with this chip.

6.3.6 Packet Processor Port Configuration

The packet processor can send/receive packets to/from the CPU in two ways:

SDMA/Inband	Packets are sent/received through the SDMA.
CPU Port	Packets are sent/received through the RGMII interface.

Setting SDMA Mode

When working in Inband mode (through the packet processor network ports), all ports are configured as members of VLAN 1, thus enabling the traffic to pass through any network port by reconnecting the cable to another network port.

To configure the system to work in SDMA mode:

1. Start the system. When countdown starts, press any key to invoke the U-Boot prompt.
2. Run: ***setenv ethprime ppsdma***.
3. To save the parameter to the Flash, run ***saveenv***.
4. Restart the system.

Setting CPU Port Mode

When working in CPU Port mode, only one port, called **cpu_port**, is created.

To configure the system to work in CPU Port mode:

1. Start the system. When countdown starts, press any key to invoke the U-Boot prompt.
2. Run ***setenv ethprime ppmii***.
3. Run ***setenv eth1addr <byte0:byte1:byte2:byte3:byte4:byte5>***.
4. To save the parameter to the Flash, run ***saveenv***.
5. Restart the system.

6.3.7 Built-in Tests for U-Boot 1.1.4

A set of built-in tests can be used to test the basic functionality of the U-Boot. Each test runs an appropriate U-Boot command and analyzes a result of this command.

To enable built-in tests, add `DIAG_SUPPORT=y` to the Makefile in the relevant board section. To run the tests, execute the ***mv_diag*** command in the U-Boot prompt, and select relevant options in the menu that appears (selecting '6' executes all tests).

Note

Several tests are board-specific, so if tests failed on a customer board, check the expected result and update the built-in tests.

A System Setup

A.1 System Hardware Preparation

If you use a development board, proceed as follows:

1. Assemble the board as described in the Development Board User Guide.
2. Verify that the development board jumpers are set according to the jumper settings described in the Development Board Configuration.
3. Connect one end of an Ethernet cable to the Out-of-Band (OOB) port (default is port 0) and the other end to the network outlet. Use either a crossover or regular Ethernet cable. The Marvell PHY auto-detects the cable's polarity.
4. Connect the USB end of the USB-to-Serial cable to the UART1 port (located on the board's AMC extender card) and the other end to your host's COM port.

A.2 System Boot Preparation

A.2.1 Preparing U-Boot for Booting with BootROM Firmware

By default, the U-Boot runs with the BootROM firmware. Upon board activation, the BootROM starts running, initializes the DRAM, copies the U-Boot to the DRAM, and proceeds to executing the U-Boot code. For more information on the BootROM, refer to the "BootROM Firmware" section of the device Functional Specifications.

Burning u-boot.bin to SPI/NAND

Check the exact name of the ***.bin** file stored in the **images** folder. For example, the file named **u-boot-1.1.4-xCat2Qfp98DX-250MHz-spi.bin** is the U-Boot 1.1.4 image for the xCat2-QFP chip, with DDR configured to 250MHz.

The image must be burned to the SPI Flash to boot the system.

1. Set the board jumpers to select the SPI/NAND as the boot device.
2. Create a **u-boot.bin** file for burning to the boot device.
3. Configure the U-Boot network environment parameters—ipaddr, serverip, and netmask.
4. Open a TFTP server and have it point to the U-Boot image.
5. Connect the Ethernet cable to the Out-of-Band port 0 on the board.
6. Boot the system using the ICE tool or from UART, with the designated U-Boot image (refer to Section A.2.2, Boot Options).
7. Stop the U-Boot at countdown.
8. From the U-Boot shell, run ***bubt <Image Name>***.

   For example: ***bubt u-boot.bin***.

   When asked to delete the environment parameters, press **Y** to reset or **N** otherwise.
9. Reset the system.

A.2.2 Boot Options

A.2.2.1 Booting from UART

Note

The steps described below are performed in ***HyperTerminal***. If you use another terminal, verify that it supports the Xmodem protocol and sending characters.

1. On your PC, open the HyperTerminal window and configure the following settings:
   - Connect using COMx (depending on your serial connection to the PC).
   - Bit per second: 115200
   - Data bits: 8
   - Parity: None
   - Stop bit: 1
   - Flow Control: None
2. Switch the board off.
3. In the HyperTerminal, select **transfer** --> **send text file** --> **bb11223344556677_boot.bin**.

Note

bb11223344556677_boot.bin is provided as an attachment to this PDF file.

File attachments are only supported by Adobe Reader 6.0 and above.

To download the latest version of Adobe Reader, go to http://www.adobe.com.

4. Switch the board on.
5. Exit the HyperTerminal and run it again. The "§§§" symbols appear in the HyperTerminal. In case these symbols do not appear, repeat Steps 2–5.
6. In the HyperTerminal, select **transfer** --> **send file** --> **<image>_uart.bin**.
7. In the **Protocol** option, select **Xmodem** and click **send**. The file is loaded and the Marvell prompt appears. In case the prompt does not appear, hit any key or press CTRL+C.
8. Open a TFTP server and point it to the U-Boot image.
9. Connect the Ethernet cable to the Out-of-Band port 0 on the board.
10. From the U-Boot shell, run ***bubt <Image Name>***. For example: ***bubt u-boot.bin***.
11. When asked to delete the environment parameters, press **Y** to reset or **N** otherwise.
12. Reset the system.

Note

Booting from UART loads the image to the DRAM.

Upon reset, the image is deleted from the DRAM, and therefore, the Boot from UART process must be repeated.

A.2.2.2 Booting with Lauterbach ICE

Prior to using this procedure, verify that Trace32 is installed on your PC.

1. Open Trace32 (ICD ARM USB).
2. From the menu, select **File-->Run Batch**.
3. Load the script that is appropriate for the board and Flash type. For example, to boot from the SPI on DB-98DX4122, load the **xCat_dx4122-A1-SPI-320MHz.cmm** script.
4. Choose **File-->Load** and browse to the U-Boot **elf** file to load it to the DRAM.
5. Change the PC counter to 0x1280000.
6. Click **Run.** The U-Boot starts running in your terminal.

The xCat LauterBach configuration script files (**cmm** files) are located in the **Lauterbach_scripts** directory in the U-Boot release. These script files set the system to working mode with the DRAM configuration of 320 MHz. Other DRAM configurations (such as 200 MHz) require different **cmm** files.

Table 7 describes the script files that must be loaded, per board.

Table 7: Script Files Loaded Per Board

Board	SPI	NAND
DB-xCat-24GE-4GP DB-xCat-48GE-4GP	xCat_dx4122-A1-SPI-320MHz.cmm	xCat_dx4122-A1-NAND-320MHz.cmm
DB-xCat-24FE-4GP	xCat_dx2122-A1-SPI-320MHz.cmm	xCat_dx2122-A1-NAND-320MHz.cmm
RD-xCat-24GE-2SFP	xCat_dx3121-A1-200MHz.cmm	N/A
DB-xCat2-24GE-2SFP	xCat2-A1-Qfp-SPI-200MHz.cmm	N/A
DB-xCat2-24GE-4GP	xCat2-A1-Qfp-SPI-250MHz.cmm	N/A

B Memory Map

The following table describes the default memory map for the U-Boot. Values are set according to base and size macros located in the **mvSysHwConfig.h** file.

Table 8: xCat/xCat2 Family Default Memory Map

Bank Name	Physical Start Address	Physical End Address	Size	Note
DDR Bank0	0x0000.0000	0x0FFF.FFFF	GE boards have 128 MB and FE boards have 256 MB per bank	
DDR Bank1	Disable			
DDR Bank2	Disable			
DDR Bank3	Disable			
Internal Registers	0xF100.0000	0xF10F.FFFF	1 MB	
PEX_0 I/O	0xC800.0000	0xC80F.FFFF	1 MB	xCat Family only
PEX_0 Memory 0	0xC000.0000	0xC7FF.FFFF	128 MB	xCat Family only
SPI Flash	0xF800.0000	0xF87F.FFFF	8 MB	
NAND Flash	0x4000.0000	0x47FF.FFFF	128 MB	
CESA Memory	0xE000.0000	0xE01F.FFFF	2 MB	xCat Family only
Packet Processor	0xF400.0000	0xF7FF.FFFF	64 MB	

Marvell Semiconductor, Inc.
5488 Marvell Lane
Santa Clara, CA 95054, USA

Tel: 1.408.222.2500
Fax: 1.408.988.8279

www.marvell.com

Marvell. Moving Forward Faster