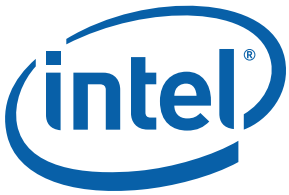


Intel NetStructure® IXB2850 Packet Processing Boards

Technical Product Specification

January 2007

Document Number: 05-2443-006

Contents

Figures

Tables

Revision History

Date	Revision	Description
January 2007	006	Updates to coincide with the introduction of new SKUs and the SRA 2.1 software release.
February 2006	005	In Section 11.3.1, “Local Software Upgrade Procedure”, added a statement to check the Release Notes for a list of the components to be upgraded for a specific release. In Section 11.3.1, “Local Software Upgrade Procedure”, and subsections, substituted specific release version references with “<release_version>” to make the procedure generic. Updated Section 11.3.1.5, “Upgrading the BMC Starter”, step 2 and step 7 to show more current versions of the IPMI Controller Dual Image and CPLD. Updated Section 4.6.2.1, “PICMG Point-to-Point Connectivity Record” to include a note about prioritized point-to-point connectivity records in FRU 0.
January 2006	004	Updated Section 8.4.2.2, “FIC Point-to-Point Connectivity Record”. Updated Section 8.5, “Board UART Configuration” and Section 8.5.1, “UART Hardware Configuration” to describe differences between spin 3A and spin 4 boards. Deleted Section 8.5.2, “UART Software Configuration”, since this functionality is not supported. In Table 35, “IPMI 1.5 Command Support “ updated “Add SDR” entry to indicated that it is supported by the BMC. Added Section 12.8, “Weight”.
December 2005	003	Updates to CPLD, FRU Information and SDR upgrade procedures and instructions for running diagnostics. Updates to Media Access Module functional description. Baseboard SW1 DIP switch description correction. Removal of references to Real Time Clock (RTC).
September 2005	002	Supports production release (SRA) boards.
June 2005	001-01	Draft of doc to support trial release.

1.0 Introduction

1.1 Product Description

Intel NetStructure® IXB2850 boards are AdvancedTCA* form-factor packet-processing blades designed for applications requiring protected packet processing. The boards are designed around the high-performance, multi-threaded Intel® IXP2850 network processor that encompasses 16 RISC-based microengines and an embedded control processor provided by the Intel XScale® core. The IXP2850 contains two integrated cryptography units for accelerating IPsec cryptographic algorithms such as, Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES) and the Secure Hash Algorithm (SHA1).

IXB2850 boards are designed to provide high-performance packet processing for a range of network applications and feature a 4x1 Gbps I/O interface to an external Wide Area Network (WAN) or Local Area Network (LAN).

IXB2850 boards are fully compliant with the AdvancedTCA standard and supports the PICMG* 3.1.3 fabric specification, which defines a dual/dual star topology with 1+1 Gbps links on the base interface and 4+4 Gbps links on the fabric interface; four interfaces on the baseboard and four on a Fabric Interface Card (FIC). This design allows customers to partition control traffic from data or payload traffic for transport over the backplane. More importantly, the transport uses redundant paths to enable High Availability (HA) for the system.

Applications include, but are not limited to:

- IP Multimedia Subsystem (IMS) Call Session Control Function (CSCF) element
- Media gateways
- Session gateway controllers
- Service edge routers

For specification updates and software downloads see the product web page at: http://www.intel.com/design/telecom/products/boards/packet_processing/9412/overview.htm.

1.2 Applicability

This Technical Product Specification (TPS) covers the following boards:

Product Code	Description
IXB28504XGBEFS	Board with fiber external interfaces; may contain restricted substances
IXB28504XGBEFSW	Board with fiber external interfaces; lead-free to second level interconnect
IXB28504XGBEFSR	Board with fiber external interfaces and an IXP2855 network processor that operates like IXP2850 network processor; lead-free to second level interconnect

1.3 Key Features

Key features of IXB2850 boards include:

- Baseboard with the IXP2850 network processor
- Dual Star Gigabit Ethernet base interface (compliant with PICMG 3.0)
- Dual Star 4+4 Gigabit Ethernet fabric interface, compliant with PICMG 3.1, option 3
- Quad Gigabit Ethernet Mezzanine Card that provides four, 1 Gigabit Ethernet I/O ports
- Incorporates an Intelligent Platform Management Controller (IPMC) that supports the Intelligent Platform Management Interface (IPMI)
- Full AdvancedTCA compliance
- Boot code, Linux Support Package (LSP), and diagnostics
- Device drivers for baseboard and mezzanine cards
- Support for Carrier Grade Linux* (CGL)
- Support for the Intel® Internet Exchange Architecture Software Development Kit (Intel® IXA SDK)
- Front panel I/O

1.4 Functional Description

Figure 1 shows a functional block diagram of an IXB2850 board. The main functional blocks include:

- Forwarding Network Processor
- Media Access Module
- Backplane Access Module
- Quad Gigabit Ethernet Mezzanine Card
- PMC Site

The following sections provide high-level descriptions of these functional blocks.

1.4.1 Forwarding Network Processor

The forwarding Network Processor (NP) is located on the NP Module, a separate card plugged into the baseboard. This module contains a single IXP2850 network processor and its environment (such as QDR, RDRAM, and flash). An additional QDR memory or TCAM coprocessor card can be added to the network processor through the mezzanine interface located on the NP Module.

The IXP2850 network processor has programmable microengines to forward data traffic and an XScale core to run all other software required to control forwarding tables and all other parts of the board.

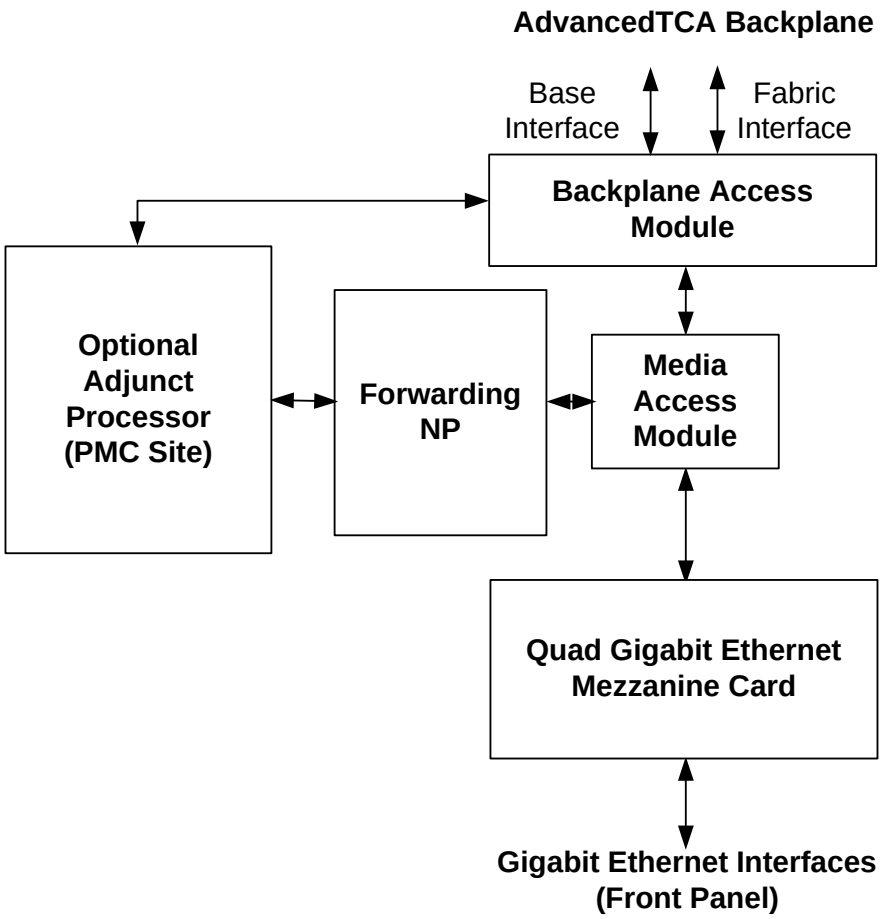
To access high-performance media access devices, the IXP2850 is equipped with a 16-bit SPI-4.2 bus. See Chapter 3.0, “Hardware Overview” for more information.

1.4.2 Media Access Module

The IXP2850 network processor is the heart of an IXB2850 board, and the SPI-4.2 is its media interface bus.

The SPI-4.2 bus is a 16-bit wide bus (in each direction), dedicated to high-speed network forwarding tasks. Signal conversion and distribution occurs inside the Media Access Module. The Media Access Module is comprised of a SPI-3/4 Bridge chip and Fork FPGA set used to convert SPI-4.2 from the network processor into four SPI-3 lines routed to the backplane access module and to personality mezzanine cards. See Section 3.2.1, “Media Access Module” on page 28 for a block diagram of the circuitry and more detail.

Figure 1. IXB2850 board functional block diagram



1.4.3 Backplane Access Module

This module connects to the main backplane communication channels, the base interface and the fabric interface, each of which is described below.

Base Interface

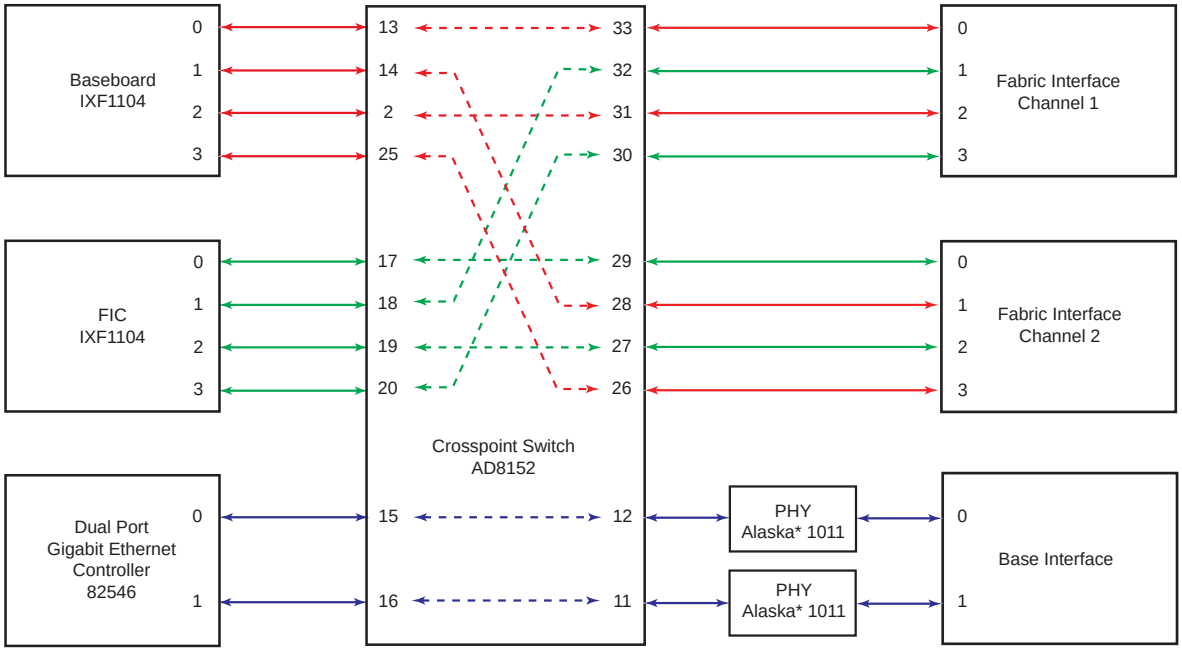
This interface is used to access the AdvancedTCA base interface, which is defined as a 1000BASE-T Ethernet by PICMG 3.0. Ports 0 and 1 of the Intel® 82546 Dual Port Gigabit Ethernet Controller together with two PHY 0 and 1 devices are used to connect to the base interface. This interface, in a PICMG 3.1 scenario supported by IXB2850 boards, is used for control traffic only. Data traffic is routed through the fabric interface.

Fabric Interface

Two Intel® IXF1104 Quad Ethernet Media Access Controller (MAC) devices (located on the baseboard and the Fabric Interface Card) are used to access the AdvancedTCA fabric interface, which is defined as a 1000BASE-BX Ethernet by PICMG 3.1.

Note: All fabric interface channels are support through the IXF1104 Gigabit Ethernet MAC SerDes interfaces. In the default configuration, the ports on the baseboard and FIC IXF1104 devices are connected to fabric interface channels 1 and 2 as indicated in Figure 2.

Figure 2. Crosspoint switch configuration



1.4.4 Quad Gigabit Ethernet Mezzanine Card

The Quad Gigabit Ethernet Mezzanine Card is based on the IXF1104 Gigabit Ethernet device with four Gigabit interfaces to a Fiber Media Interface Card (MIC-F). The Media Interface Card provides four Gigabit Ethernet interfaces that are accessible from the front panel.

1.4.5 PMC Site

The PCI Mezzanine Card (PMC) site on IXB2850 boards can accommodate an optional Adjunct Processor (AP) Mezzanine Card, typically used to host control plane software. It can perform routing, signaling, or service control functions. It is possible that either a single AP Mezzanine Card hosts several processing elements, or that several such cards perform dedicated control functions.

The AP Mezzanine Card conforms to the Processor PCI Mezzanine Card (PrPMC) standard, so that a PrPMC module (such as a Pentium® III processor-based PrPMC module) can be used in the PMC site on an IXB2850 baseboard. A System Clock Generator PMC Mezzanine Card could also be used in this position on the board.

1.5 Data Flow

The AdvancedTCA standard is defined in PICMG 3.x specs, with the base definition described in PICMG 3.0 (specifying the mechanics, airflow, power distribution, etc.). One part of this specification describes the backplane, its connector types, and connection topology.

For inter board communication, the following connection types are defined:

- Base Interface: Defined as 1000BASE-T Ethernet working in a dual star topology; each board is connected to two redundant fabric boards.
- Fabric Interface: Only backplane topologies are defined because this interface type can be changed and its type is specified in subsequent PICMG 3.x specifications. For example, PICMG 3.1, option 3 specifies the fabric interface as 1000BASE-BX (fiber) Gigabit Ethernet. The fabric interface can be used in a dual-star topology.

IXB2850 boards are designed to work with the base and fabric interfaces. The backplane access module can be configured to work in dual-star (single or double port) on the fabric interface. Figure 3 shows the data flow when using the base interface and Figure 4 shows the data flow when using the fabric interface.

Figure 3. Data flow when using the base interface

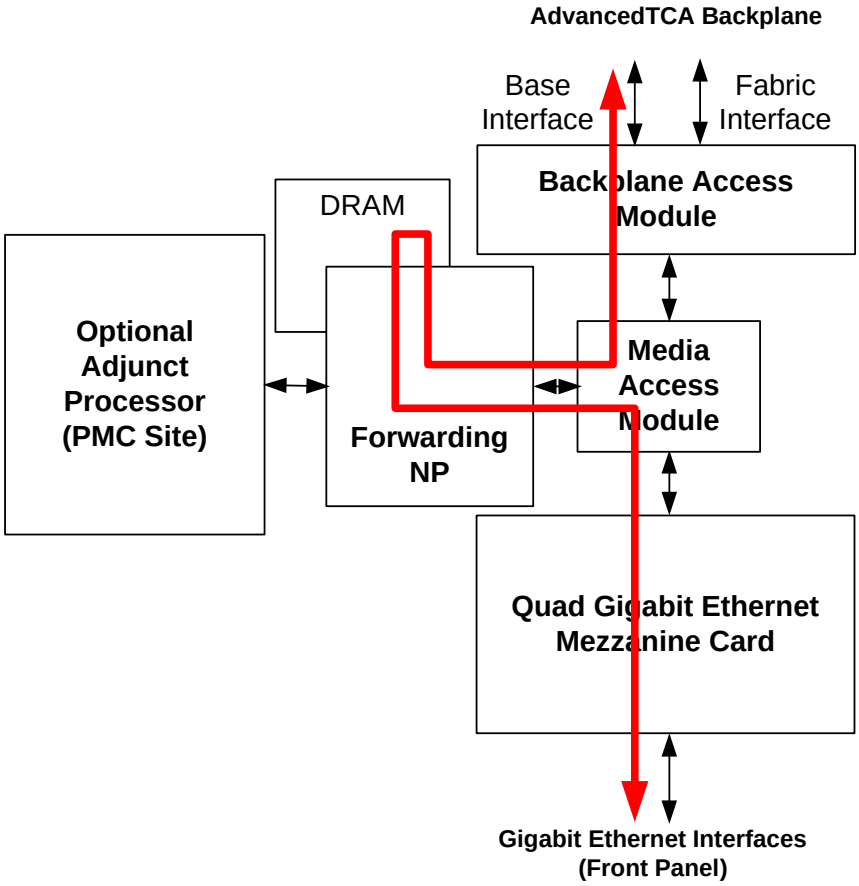
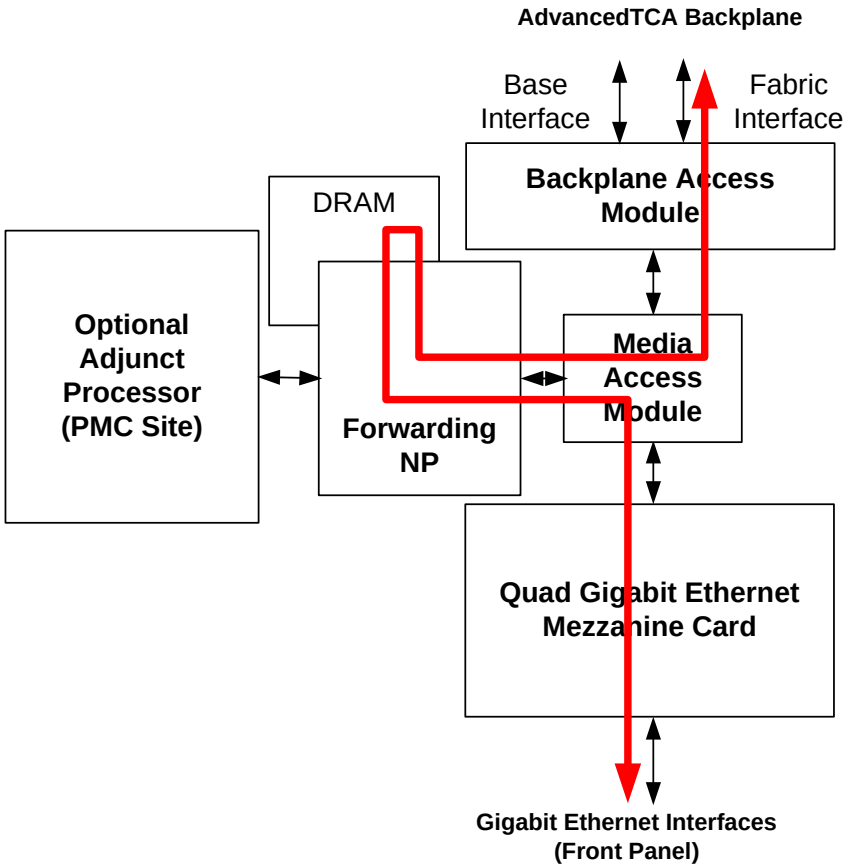


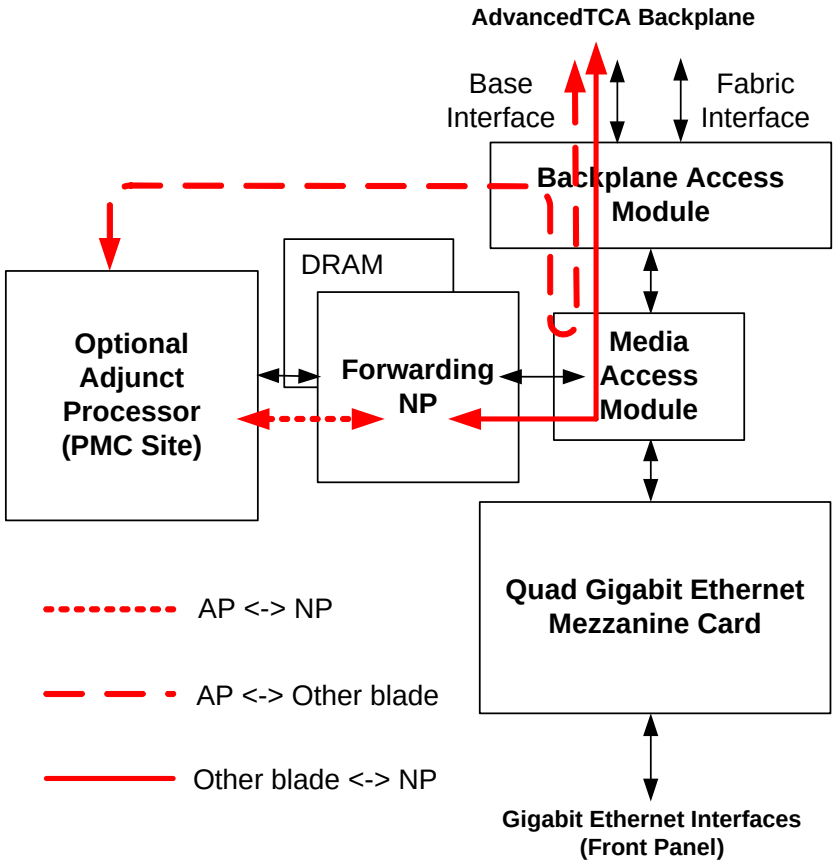
Figure 4. Data flow when using the fabric interface



1.6 Control Flow

Figure 5 shows a PICMG 3.0 scenario where only the base interface is used, and both data and control traffic are forwarded through this interface. In this case, control traffic is generated/terminated by the Adjunct Processor (AP) Mezzanine Card and is mixed with the data traffic of the network processor in the Media Access Module.

Figure 5. Control flow scenario when using the base interface for data and control flows



2.0 Getting Started

This chapter describes how to configure a system that includes Intel NetStructure® IXB2850 boards making it possible to run the IXB2850 firmware.

2.1 System Requirements

Verify that your setup matches the requirements given in Table 1. If there are any discrepancies, you will need to address them before proceeding.

Table 1. System requirements

System Component	Requirement
Chassis	• AdvancedTCA* chassis
Shelf Management Controller (ShMC)	• Must be supported by the chassis (see the chassis vendor documentation)
Switch Fabric Board	• AdvancedTCA board • Gigabit Ethernet support for base interface channel 1 and channel 2 • Gigabit Ethernet support for fabric interface channel 1 and channel 2 (required when using boards with a Fabric Interface Card, FIC) • Uplink port for external Ethernet connections
Management PC Hardware	• Pentium® III processor or higher • Fast Ethernet Interface • Gigabit Ethernet Interface • Al least two serial ports (optionally, a multi-UART adapter or a USB/Serial converter †
Management PC Operating System	• Red Hat* Linux 9.0
Ethernet Hub	• 10/100 Mbs Ethernet Hub or Switch (for debug network connection)
Line Cards	• IXB2850 boards
† For each board in the system, two serial ports on the Management PC are used. If a user wants to have a connection to the Shelf Management Controller (ShMC) console, a third serial port is required.	

2.2 IXB2850 Board Hardware Configuration

IXB2850 boards must be configured for the desired mode of IPMI operation. The possible modes of operation and the corresponding settings for jumper J2 on the IXB2850 board are given in Table 2. See Figure 27, "IXB2850 jumper and switch locations" on page 75 for the location of jumper J2 on IXB2850 boards.

Table 2. Setup for IPMI

Mode	Description	Jumper Settings	Notes
PICMG 3.0	IPMI support for PICMG 3.1 ShMC	J2: 1-2 – ON J2: 15-17 – ON	The default (recommended) mode
Legacy	No ShMC installed or no PICMG 3.1 shelf manager used	J2: 1-2 – OFF J2: 15-17 – ON	Should only be used when the PICMG 3.0 shelf manager is not available
BMC power control disabled	The BMC does not control power on the board	J2: 15-16 – ON	Not recommended for normal operation and should only be used for service. Can be used temporarily to boot the board when the onboard BMC is damaged.

2.3 System Components Connection

The connection of components in a basic system that contains an IXB2850 board is shown in Figure 6.

Note: The grayed elements in Figure 6 are optional for some development/usage scenarios.

To perform the IXB2850 system hardware setup, proceed as follows:

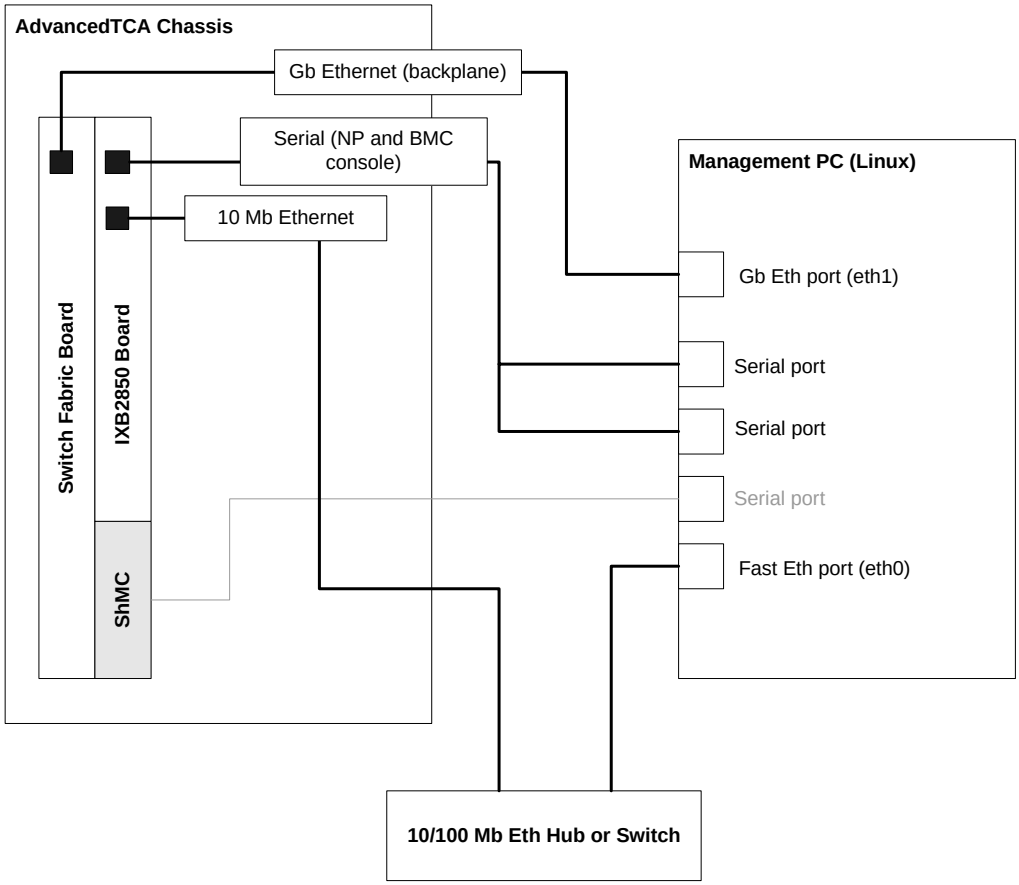
1. Configure the AdvancedTCA chassis and the Switch Fabric boards according to vendors' documentation.
2. Insert the IXB2850 board, Shelf Management Controller (ShMC) and the Switch Fabric boards in the chassis. Take into consideration that some AdvancedTCA chassis allocate specific slots for Switch Fabric boards and an IXB2850 board should not be placed in those locations.
3. Set up the connections between the IXB2850 boards, Ethernet hub, and the Management PC as follows:
   a. Connect the Management PC Fast Ethernet interface to the Ethernet hub.
   b. Connect the Management PC Gigabit Ethernet interface to the Switch Fabric board uplink Ethernet port.
   c. Connect the Management PC serial port to the IXB2850 board NP console port.
   d. Connect the Management PC serial port to the IXB2850 board BMC console port.

Note: The debug console cable, IXB3GDEBUGCABLE, a separately orderable item that connects to the COM port on IXB2850 boards provides serial interface connections (DB-9 connectors) for both the NP console and the BMC console. See Section 12.8.1, "Debug Console Cable Specification" on page 165 for more information.

Note: The NP and BMC console port from the IXB2850 point of view is the same port (named CONSOLE). See Section 12.8.1, "Debug Console Cable Specification" on page 165 for more information.

4. Connect the IXB2850 board debug Ethernet port to the Ethernet hub.

Figure 6. Chassis, IXB2850 board, switch fabric board and management PC configuration



2.4 Management PC Configuration

Configure hardware interfaces and services on the Management PC as described in the following subsections.

2.4.1 Hardware Interfaces Configuration

Hardware interfaces that need to be configured include:

- Network Interfaces
- Serial Ports

2.4.1.1 Network Interfaces

Set up network interfaces on the Management PC as follows:

1. Configure Fast Ethernet interface by editing the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. In the following example, it is assumed that Fast Ethernet interface is named *eth0*. The name of the interface may differ depending on the hardware configuration.

```
# IXB2850-specific
DEVICE=eth0
```

```
BOOTPROTO=static
BROADCAST=10.69.255.255
IPADDR=10.69.22.253
NETMASK=255.255.0.0
NETWORK=10.69.0.0
ONBOOT=yes
```

Note: The configuration of the Fast Ethernet interface is given as an example and can be adjusted to the local network configuration.

2. Configure the Gigabit Ethernet interface by editing the `/etc/sysconfig/network-scripts/ifcfg-eth1` file. In the following example, it is assumed that the Gigabit Ethernet interface is named *eth1*. The name of the interface may differ depending the hardware configuration.

```
# IXB2850-specific
DEVICE=eth1
BOOTPROTO=static
BROADCAST=172.255.255.255
IPADDR=172.0.0.10
NETMASK=255.0.0.0
NETWORK=172.0.0.0
ONBOOT=yes
```

Note: The BROADCAST, IPADDR and NETWORK IP addresses of the Gigabit Ethernet port are given as an example and can be adjusted to the user configuration.

3. Cycle the e*th0* and *eth1* interface:

```
#/sbin/ifdown eth0
#/sbin/ifup eth0
#/sbin/ifdown eth1
#/sbin/ifup eth1
```

2.4.1.2 Serial Ports

The Management PC serial ports connected to the console ports of the IXB2850 board (NP and/or BMC) should be set to operate in the following configuration:

- 115,200 baud
- 8 bits, no parity, 1 stop bit
- Hardware and software flow control disabled

The `minicom` application, running on the Management PC, is recommended as the serial terminal application. The configuration procedure for the `minicom` (v. 2.00.0) is given below:

1. Run the Linux minicom.
2. Configure the serial port parameters in `minicom` as follows:
   a. Press **Ctrl-A O** and select the **Serial port setup** option.
   b. Specify the serial port device, for example, `/dev/ttyS1`.
   c. Set **Bps/Par/Bits** to: 115200 8N1
   d. Set **Software and Hardware Flow Control** to **No**.
3. Configure the serial port parameters in minicom as follows:
   a. Press **Ctrl-A O**, select **Filenames and paths** and specify the script directory (option **C**):
   `/tftpboot`

4. Press **Enter**, then select **Exit** to return to the main `minicom` window.
5. Press **Crtl-A O** and select **Save setup as...** from the Configuration dialog box. Specify the name for current configuration (for example, `ttyS1`). Once saved, a `minicom` configuration can be reloaded automatically by specifying the configuration file name as an argument when invoking minicom (for example: `minicom ttyS1`).

2.4.2 Services

The following services must be configured on the Management PC:

- NFS Server
- TFTP Server
- DHCP Server

2.4.2.1 NFS Server

Configure the Management PC as an NFS server as follows:

1. Create the `/tftpboot/` directory on the Management PC.
2. Add an entry in the `/etc/exports` file:

```
# IXB2850-specific
/tftpboot/* (rw,sync)
```

3. Check the NFS service status:

```
# /etc/rc.d/init.d/nfs status
```

Note: If the NFS service is not running, use the following command at the Linux prompt to run the service:

```
# /etc/rc.d/init.d/nfs start
```

4. Restart NFS service:

```
# /etc/rc.d/init.d/nfs restart
```

5. You can add NFS service startup to init scripts by running the following command at the Linux prompt:

```
# /sbin/chkconfig nfs on
```

2.4.2.2 TFTP Server

Configure the Management PC as an TFTP server as follows:

1. Create the `/tftpboot` directory on the Management PC.
2. Edit the `/etc/xinetd.d/tftp` file on the Management PC as follows:

```
service tftp
{
    socket_type = dgram
    protocol    = udp
    wait        = yes
    user        = root
    server      = /usr/sbin/in.tftpd
```

```
    server_args = -c -s /tftpboot
    disable     = no
    per_source  = 11
    cps         = 100 2
}
```

3. Restart the TFTP service:

```
# /etc/rc.d/init.d/xinetd restart
```

4. You can add the startup of the xinetd service to init scripts by running the following command at the Linux prompt:

```
# /sbin/chkconfig xinetd on
```

2.4.2.3 DHCP Server

Configure the Management PC as an DHCP server as follows:

1. Edit the `/etc/dhcpd.conf` file, adding the following entries:

```
# IXB2850-specific
ddns-update-style interim;
ignore client-updates;

# NP booting
subnet 10.0.0.0 netmask 255.0.0.0 {
    default-lease-time   600;
    max-lease-time      7200;
    option subnet-mask 255.0.0.0;
    option broadcast-address 10.255.255.255;
    host IXB2850_board {
       hardware ethernet GLC_BOARD_ETH_ADDR;
       fixed-address GLC_BOARD_IP_ADDR;
    }
}
```

   where:

   — `GLC_BOARD_ETH_ADDR` is the IXB2850 board's debug Ethernet address that appears when the board is started. Look for the line:
   `Ethernet slowport IXMB28x1: MAC address XX:YY:ZZ:VV:WW:PP`

   — `GLC_BOARD_IP_ADDR` is the address that the user allocates for the IXB2850 board.

2. Check whether the `/var/lib/dhcp/dhcpd.leases` file (the lease database) exists. If the file does not exist, create a zero length file:

```
# touch /var/lib/dhcp/dhcpd.leases
```

3. Check the DHCP service status:

```
# /etc/rc.d/init.d/dhcpd status
```

4. If the DHCP service is not running, activate the service:

```
# /etc/rc.d/init.d/dhcpd restart
```

5. You can add the dhcpd service startup to init scripts by issuing the following command at the Linux prompt:

```
# /sbin/chkconfig dhcpd on
```

2.4.3 Running Linux and Drivers

Create a Linux file system and run the test applications to exercise the IXB2850 drivers as described in the following subsections:

- File System
- Starting Linux
- Starting the Gigabit Ethernet Test Application
- Starting the Example IXA SDK Pipeline Application

2.4.3.1 File System

Note: Throughout this document, kernel modules are shown with a .ko extension. This is valid for Linux kernels from the 2.6 line. For kernels from the 2.4 line, kernel modules have a .o extension.

Create a suitable Linux file system as follows:

1. Prepare a Linux file system compatible with the kernel version you are using. Place this file system in the /tftpboot/GLC_BOARD_IP_ADDR directory.

Note: The kernel (SSU image) may also be placed on flash in a kernel partition. To prepare an SSU image from a raw kernel image, the ssu_image tool can be used.

2. Create the directory /tftpboot/GLC_BOARD_IP_ADDR/glc. All drivers for the IXB2850 board will be placed in this directory.
3. Copy the following files from the IXB2850 Firmware distribution to the glc directory:
   a. bb_linux_res.ko (from Firmware/IXB28x0/Linux/drivers/ixd4bb/bin)
   b. bb_linux_drv.ko (from Firmware/IXB28x0/Linux/drivers/ixd4bb/bin)
   c. bfm.ko (from Firmware/IXB28x0/Linux/drivers/bfm/bin)
   d. ixf1104drv_kernel.ko (from Firmware/IXB28x0/Linux/drivers/ixd4gea1f/bin)
   e. soft_reset.ko (from Firmware/IXB28x0/Linux/drivers/soft_reset/bin)
   f. bmcAccess (from Firmware/IXB28x0/Linux/lib/bmcAccess/bin)
   g. bmcAgent (from Firmware/IXB28x0/Linux/applications/bmcAgent/bin)
4. Create the following glc_start script in glc directory:

```
insmod /glc/bb_linux_res.ko
insmod /glc/bb_linux_drv.ko
rmmod bb_linux_res.ko
insmod /glc/bfm.ko
insmod /glc/soft_reset.ko
insmod /glc/ixf1104drv_kernel.ko
while [ ! -c /dev/bb ]
do
    echo "waiting for /dev/bb file"
done
while [ ! -c /dev/gbe ]
do
    echo "waiting for /dev/gbe file"
done
/glc/bmcAccess &
```

```
sleep 1
/glc/bmcAgent &
/sbin/wtd
```

Note: The first line in the script above loads the baseboard driver. The baseboard driver operates in one of two modes: with or without the 64-bit prefix used to distinguish packets coming from or going to the Intel® IXF1104 on the baseboard or FIC as explained in Section 3.3.1, “Ethernet Packet Formats Over the Backplane” on page 31. By default, the driver works with the prefix enabled. In firmware version 1B2 and later, when loading the driver, the **noFabricPrepend** parameter can be used to disable the use of the prefix. To disable the prefix, change the first line of the script to: `insmod /glc/bb_linux_drv.ko noFabricPrepend=1`.

Note: The script above shows the `wtd` application in the `/sbin` directory. The actual location of the `wtd` application is user-selectable. The user may choose to place the `wtd` application in an alternate location, for example, in the `/glc` directory, which contains other files provided by Intel.

5. Configure your file system to start the `glc_start` script when Linux completes boot up.

   For example, in the `/etc/rc.d/rc3.d` directory on the root file system, create an `S70glc` file that contains:

   ```
   modprobe softdog
   /glc/glc_start > /dev/console 2> /dev/console
   ```

6. Copy the `zImage` file with the Linux kernel prepared for the IXB2850 board to the `glc` directory.

2.4.3.2 Starting Linux

Configure IXB2850 boards to automatically run the Linux kernel. While in RedBoot*, edit the configuration file by typing `fc` (you can use command `fc -l` to list the current configuration). The configuration should look like this:

```
Run script at boot: true
Boot script:
.. load -r -v -m tftp -b 0xd008000 /tftpboot/GLC_BOARD_IP_ADDR/glc/zImage
.. exec

Boot script timeout (1000ms resolution): 3
Use BOOTP for network configuration: true
BOOTP requests numbers <1..30>: 3
GDB connection port: 9000
Use BOOTP for Base Interface #1 network configuration: false
Use BOOTP for Base Interface #2 network configuration: false
Local Base Interface #1 IP address: 0.0.0.0
Local Base Interface #2 IP address: 0.0.0.0
Default Base Interface #1 server IP address: 0.0.0.0
Default Base Interface #2 server IP address: 0.0.0.0
Network debug at boot time: false
Skip POST execution: false
Soft reset: true
SRAM initialization with content: true
```

1. Save the configuration and restart the board.
2. After restart, the IXB2850 board should load the Linux kernel from the Management PC and use the provided file system. The user should be able to login as root and start using IXB2850 drivers. To do this, change directory to `/glc`.

Note: While editing the configuration stored in flash using the `fc` command, the Boot script section is erased when the user presses enter and saves the configuration as shown in the trace below.

```
RedBoot> fc
Run script at boot: false true
Boot script:
.. watchdog off
.. fis load linux
.. exec
Enter script, terminate with empty line
>>
Boot script timeout (1000ms resolution): 2
```

The Boot script section must be rewritten every time a user changes **anything** in the configuration.

2.4.3.3 Starting the Gigabit Ethernet Test Application

The simplest way to check if the IXB2850 board is functional and is capable of forwarding traffic is to run the Gigabit Test Application as follows:

1. Copy the `gbe_test_app and ixf1104drv_kernel.ko` files
   - From: The `Firmware/IXB28x0/Linux/drivers/ixd4gea1f/bin` directory on the firmware distribution CD image.
   - To: The `/tftpboot/GLC_BOARD_IP_ADDR/glc` directory on the Management PC.
2. Connect a Gigabit Ethernet traffic generator to the first port of the Quad Gigabit Ethernet Mezzanine card.
3. Boot the IXB2850 board as specified in Section 2.4.3.2, “Starting Linux” on page 23.
4. Use the `insmod` command to load the `ixf1104drv_kernel.ko` driver to the kernel.
5. Run the test application by typing `./gbe_test_app`.
6. In the test application menu, choose option **b**, choose port number **0**, mode **0**. Next choose option **x** to load the loopback microcode. Next choose option **i** to setup promiscuous mode on port **0**.
7. Start the traffic generator and check if packets are returned to it.

2.4.3.4 Starting the Example IXA SDK Pipeline Application

The IXB2850 board software includes a patch for the IXA SDK Framework and a patch for one IXA SDK example application, that is, the IPv4 forwarder in the `src/applications/ipv4_forwarder/4oc12_pos_6gb_ethernet_2801` directory (see the *Release Notes* for details).

The documentation for the example application, is included in the `Documentation/Software_Framework/IXA-SDK/BuildingBlocksAppsDesignGuide.pdf` (chapter 18) and `src/applications/ipv4_forwarder/4oc12_pos_6gb_ethernet_2801/2801-POSEthIPv4Fwd-readme.pdf` and is still valid. The application has been modified to support backplane fabric ports and it uses IXB2850 firmware.

Only the Linux version of the core components are supported for IXB2850 boards.

To run the example application, proceed as follows:

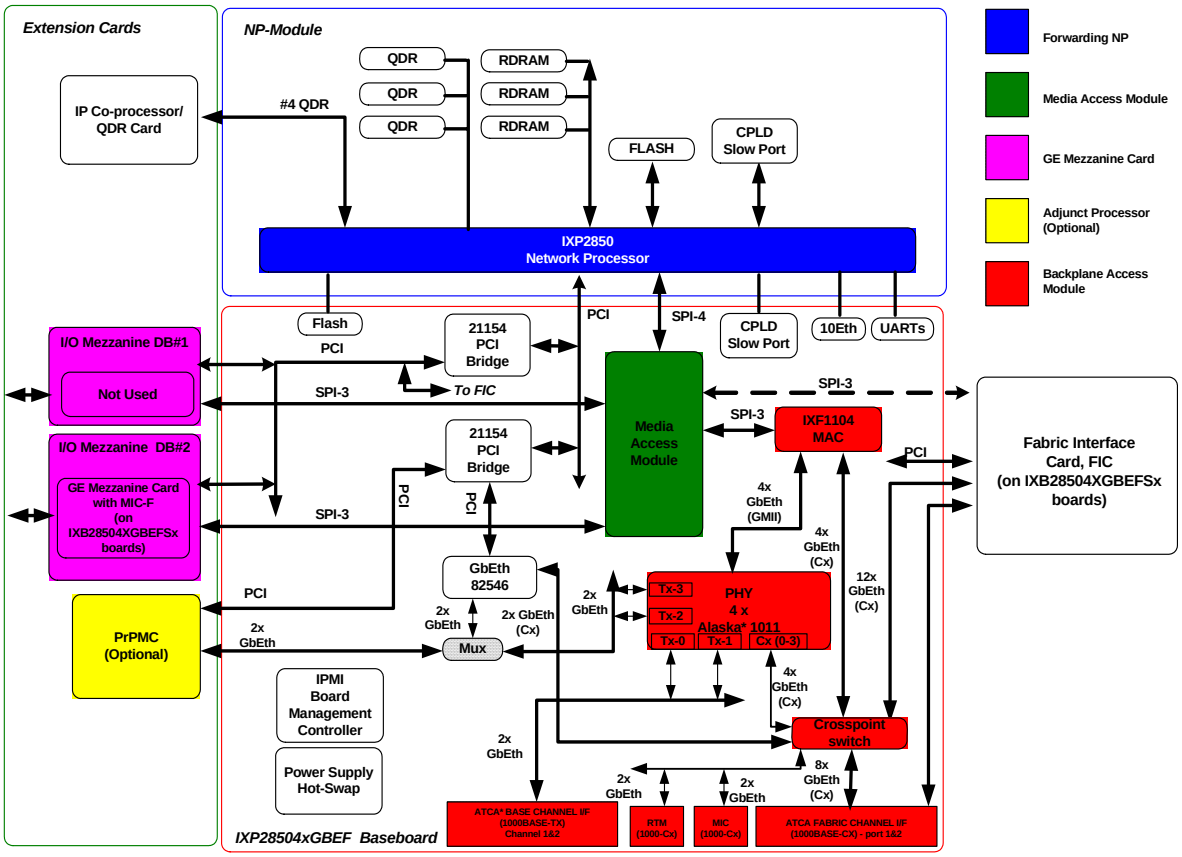
1. Install the IXA SDK Tools CD, the IXB2850 firmware, and the IXA SDK Framework CD.
2. Apply the patch for the IXA SDK Framework and the patch for the example application.
3. Build the application microcode project (see the `2801-POSEthIPv4Fwd-readme.pdf` file for details).
4. Set the new IXP2XXX_TOOLCHAIN_PREFIX environment variable and build the core components (see the `2801-POSEthIPv4Fwd-readme.pdf` file for details).
5. Copy all the output of the core components build and the `microcode pos_ethernet_2801.uof` file to the application directory in the IXB2850 root file system.
6. Run the IXB2850 board, run Linux.
7. In the Linux shell, load and start the application components by running the `run_app` script, then configure it with `rtm_config_linux.sh` (see the `2801-POSEthIPv4Fwd-readme.pdf` file for details).
8. Connect a host station that includes a Gigabit Ethernet card with IP address 16.<port+1>.0.10 or a network traffic generator to the Quad Gigabit Ethernet Mezzanine card on the IXB2850 board.
9. Try to send an ICMP echo request (`ping`) to the IP address of the port where you are connected to 16.<port+1>.0.1 or prepare packets for forwarding.

3.0 Hardware Overview

3.1 Introduction

Figure 7 is a block diagram showing the main hardware elements of an IXB2850 board.

Figure 7. IXB2850 board block diagram



IXB2850 boards consists of the following elements:

- Baseboard
- Intel® IXP2850 Network Processor (NP) module
- Fabric Interface Card (FIC)
- Quad Gigabit Ethernet Media Mezzanine Card, with Fiber Media Interface Card (MIC-F)

IXB2850 boards can also be extended using the following optional cards:

- QDR SRAM card or coprocessor card (for example, a TCAM)
- PrPMC Adjunct Processor (AP)

Note: Since the PrPMC is optional, the data path from the PrPMC to the Base Interface is also optional. See Figure 10, "Example data paths when using FIC" on page 30 for more information.

3.2 Baseboard

The baseboard contains two processors that include firmware:

- **Board Management Controller (BMC)** - This processor is responsible for communication with Shelf Management Controllers (ShMCs) via the Intelligent Platform Management Bus (IPMB) on the backplane. It also manages power for the entire board and provides access to the ID EEPROMs located on the baseboard and all other cards. See Chapter 8.0, "Board Management Controller Firmware" for firmware details.
- **IXP2850 Network Processor (NP)** - This processor is used for packet forwarding (microengines) as well as controlling specific devices (see below) on the baseboard (Intel XScale® core). See Chapter 9.0, "Network Processor Firmware" for firmware details.

The following devices on a baseboard are programmable by the Network Processor (NP):

- **Intel® IXF1104** - This Quad Gigabit Ethernet MAC Controller is connected to the NP through the Media Access Module (for packet transmission/reception) and through the slow port for device configuration and management. The slow port is a microprocessor interface typically used to access devices such as asynchronous flash or ROM, but it is also used to access the 32-bit-wide microprocessor port of the IXF1104. The slow port has an 8-bit multiplexed address and data bus, with control lines to latch data and address. Possible IXF1104 chip configurations include:
  - Base/Fabric mode - Two ports used to support the fabric interface and two other ports used for access to the base interface on the AdvancedTCA* backplane.
  - Fabric mode - All four ports are connected to the fabric interface (two ports to channel 1 and two ports to channel 2, or all 4 ports to channel 1).
- **Marvell* Alaska 1011** - Single Gigabit Ethernet PHY device. Four of these devices work as PHYs for the onboard IXF1104 chip and are connected to the IXF1104 through the GMII interface. The Alaska chip is configurable and can be managed indirectly by the NP through the IXF1104.
- **Intel® 82546** - This Dual Gigabit Ethernet MAC Controller is fully accessible via the PCI bus for packet transmission/reception as well as device configuration and management. This MAC controller is used to support the base interface on the AdvancedTCA backplane in the PICMG 3.1 configuration.
- **Media Access Module** - This module switches packets/cells between the IXP2850 SPI-4 bus and four SPI-3/UTOPIA buses connected to the following devices:
  - On-board IXF1104
  - Fabric Interface Card (FIC)
  - Quad Gigabit Ethernet Mezzanine Card (MMC #2)
  - A second Media Mezzanine Card (MMC #1), which is not used on IXB2850 boards.

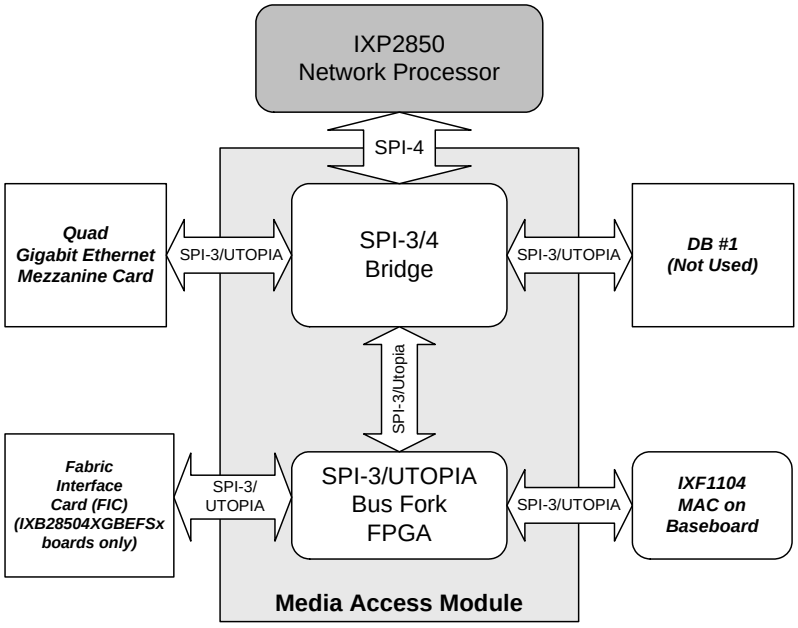
This module consists of an SPI-3/4 Bridge chip and Fork FPGA and is configurable through the slow port. See Section 3.2.1 below.

- **21154 chip** - Two PCI bridges configurable through the PCI bus.
- **10Eth** - A 10 Mbps Ethernet Controller (CS8900A) that is connected to the NP and used for debug purposes (console and debug support). This chip is fully managed (including packet transmission and reception) through the slow port.
- **UARTs** - Includes a dual UART device (16C550) dedicated to the NP, one is used for debug support, and one for communication with the Board Management Controller (BMC). There is also a third UART port embedded in the IXP2850 network processor that is used as an NP console.
- **NP Flash** - 160 MB accessible through the slow port. The flash memory available for the NP is split into five banks of 32 MB each (2 x 16 MB) due to limited NP slow port addressing capabilities. One bank is mounted on the NP module and the remaining four banks are mounted on the baseboard.

3.2.1 Media Access Module

The Media Access Module is used to split the IXP2850 SPI-4 bus into four SPI-3/UTOPIA buses utilized by the Quad Gigabit Ethernet Mezzanine Card, FIC, and on-board IXF1104 chip. The block diagram of the Media Access Module is shown in Figure 8.

Figure 8. Media Access Module block diagram



The main component of the Media Access Module is the SPI-3/4 Bridge chip, which supports three peripheral SPI-3/UTOPIA buses. The SPI-3/4 Bridge chip is a bridge type device. It does not allow switching of traffic directly between peripheral buses; all traffic is bound to or from the network processor, and any switching between peripheral buses, if required, must be implemented in the network processor. The bridge appears as a 12-port PHY device to the network processor:

- Each of these 12 ports is dedicated to one of three peripheral SPI-3 buses (one bus is split in the FPGA)
- The port numbers (0 to 11) determine which peripheral bus the packet came from or goes to

An additional FPGA is used to fork one of the chip's SPI-3 buses. The FPGA can be configured by the NP to direct data flow either through the IXF1104 MAC on baseboard and the crosspoint switch to the base channels and fabric channel 1,2 or through the FIC card to additional fabric channels.

On the baseboard, there are connections between the FPGA and the IXF1104 MAC and between the FPGA and the FIC that make it possible to work with the FIC card and the IXF1104 MAC in 32-bit mode as well as 4x8-bit mode.

3.3 Fabric Interface Card

The Fabric Interface Card (FIC) is used to connect an IXB2850 board to the fabric interface on the AdvancedTCA backplane. IXB2850 boards use a Gigabit Ethernet as the switch fabric transport and conform fully to the PICMG 3.1 standard.

The FIC is based on the IXF1104 chip (Quad Gigabit Ethernet MAC) and for packet processing is connected to the NP through the Media Access Module (see Section 3.2.1). Configuration and management is performed by the NP through the slow port. A block diagram of the FIC is shown in Figure 9.

Figure 9. Fabric Interface Card block diagram

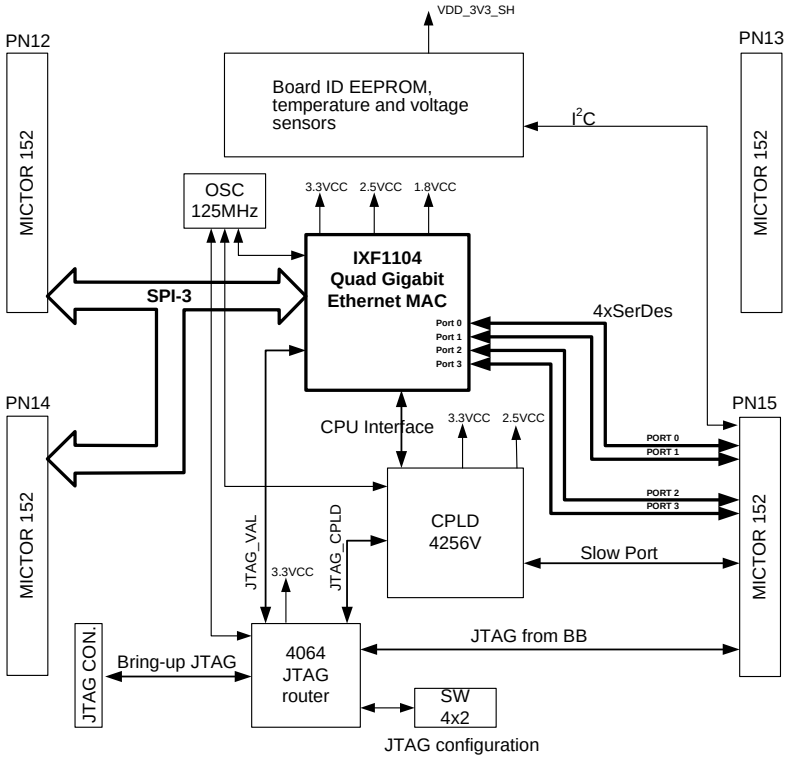
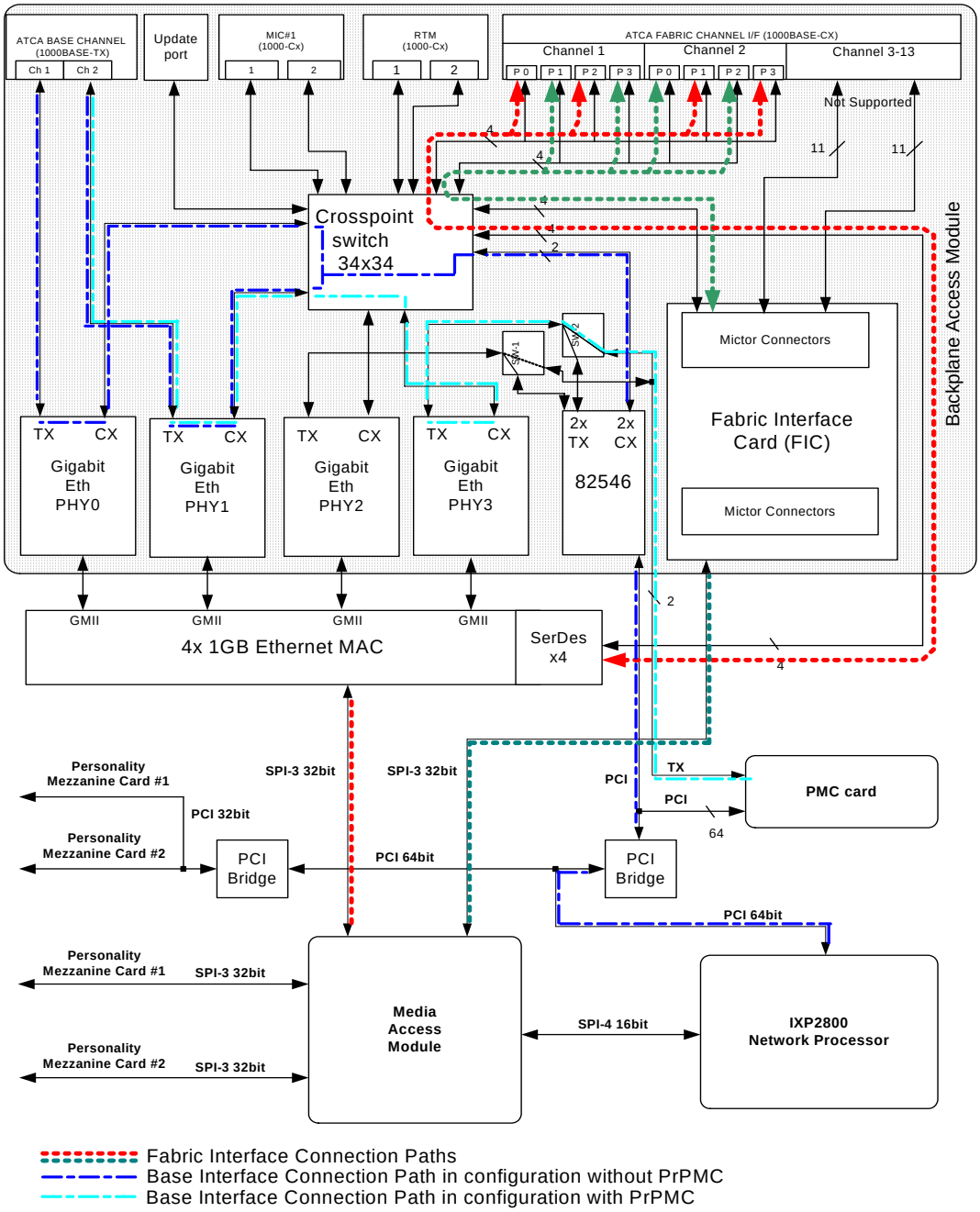


Figure 10 shows example data paths on the IXB2850 board and the role that the FIC plays in providing these paths. It also identifies specifically how the crosspoint switch on the baseboard has been configured for the routing of data to the fabric interface channels. The data paths to the base interface are shown for a configuration that does not use the PrPMC and a configuration that uses the PrPMC.

Note: IXB2850 boards do not support concurrent access to the base interface by the NPU and the PrPMC.

Figure 10. Example data paths when using FIC



3.3.1 Ethernet Packet Formats Over the Backplane

To ensure continuity of data flow to both FIC and baseboard IXF1104 MAC devices, “in-band addressing” is implemented in the Fork FPGA. This addressing scheme enables the transmission of data to, and the reception of data from, the two devices connected to the Fork FPGA. Each data packet transmitted between the Network Processor (NP) and the Fork FPGA includes an additional quad word as a header. In the transmit (TX) direction, the header contains information about the destination device of the packet. In the Receive (RX) direction, the header contains information about the source device.

In the TX direction, the NP creates the header and adds it at the beginning of each data packet. The addition of the quad word header does not modify the checksum that is transmitted in the packet. The Fork FPGA is responsible for removing the addressing header from the data packet before sending the packet to the appropriate device.

The header has a very simple structure since only one bit is needed to specify the destination device. See Figure 11.

Figure 11. In-band addressing header format

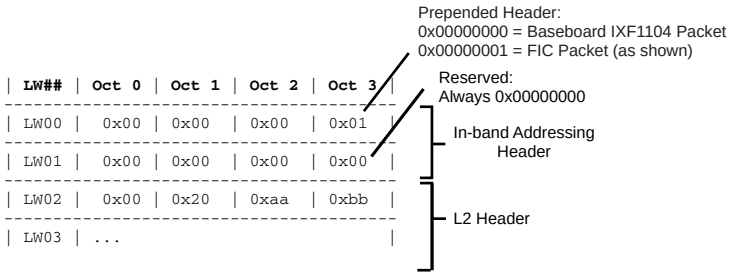


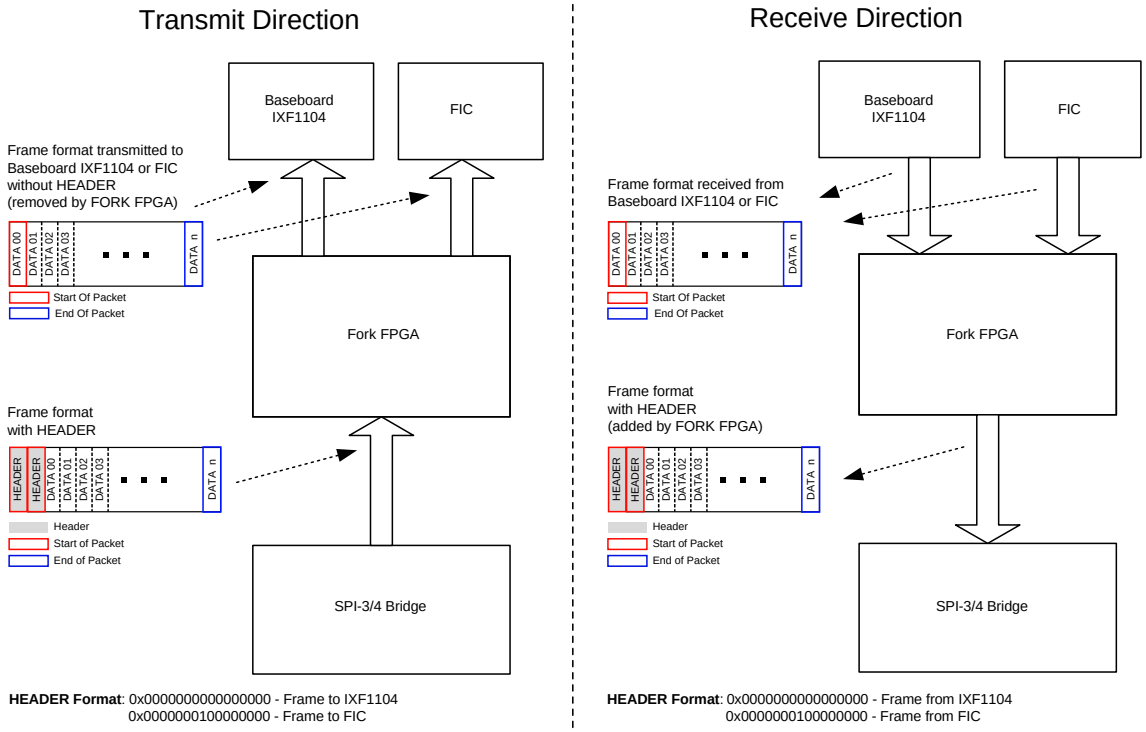
Figure 12 shows operation in the transmit (Tx) and receive (Rx) directions:

- In the transmit direction, the Fork FPGA receives data with the addressing header from the SPI-3/4 Bridge and transmits it to the Baseboard IXF1104 or FIC without the header.
- In the RX direction, the Fork FPGA is responsible for adding the header to each packet received from the Baseboard IXF1104 MAC or the FIC card.

The Fork FPGA operates with the Baseboard IXF1104 and FIC card in SPI-3, 4 x 8 bit SPHY mode. The 4 x 8 bit mode has been implemented to avoid the blocking of data flow from one device by the blockage of data on the other device. The Fork FPGA has 2048 bytes buffers (in the Fork FPGA) for each of the 8 lanes used by the Baseboard IXF1104 and FIC. Data received from the MAC on the FIC card is converted to SPI-3 MPHY 32-bit mode for the SPI-3/4 Bridge.

Note: In firmware version 1B2 and later, the addition of the 64-bit prefix, enabled by default, can be disabled when loading the baseboard driver by including the **noFabricPrepend**=1 parameter on the command line as follows: `insmod bb_linux_drv.ko noFabricPrepend=1`.

Figure 12. Data packets transmitted between NP and baseboard IXF1104 or FIC



3.4 Quad Gigabit Ethernet Mezzanine Card

IXB2850 boards operate with the Quad Gigabit Ethernet Mezzanine Card. The form factor of this board is equivalent to the PMC mezzanine card standard IEEE P1386.1 and the card supports Ethernet fiber as the physical media.

The Quad Gigabit Etherenet Mezzanine Card consists of two parts: the Media Mezzanine Card (MMC), which is the main assembly and a Fiber Media Interface Card (MIC-F), which provides the interfaces to the front panel of the board. See Section 12.1.1, “Quad Gigabit Ethernet Mezzanine Card Mechanical Specification” on page 162 for more information.

Figure 13 is a block diagram of the Quad Gigabit Ethernet Mezzanine Card. The figure shows that the access module consists of an IXF1104 Quad Gigabit Ethernet MAC and a Marvell* Alaska Quad Gigabit Ethernet PHY.

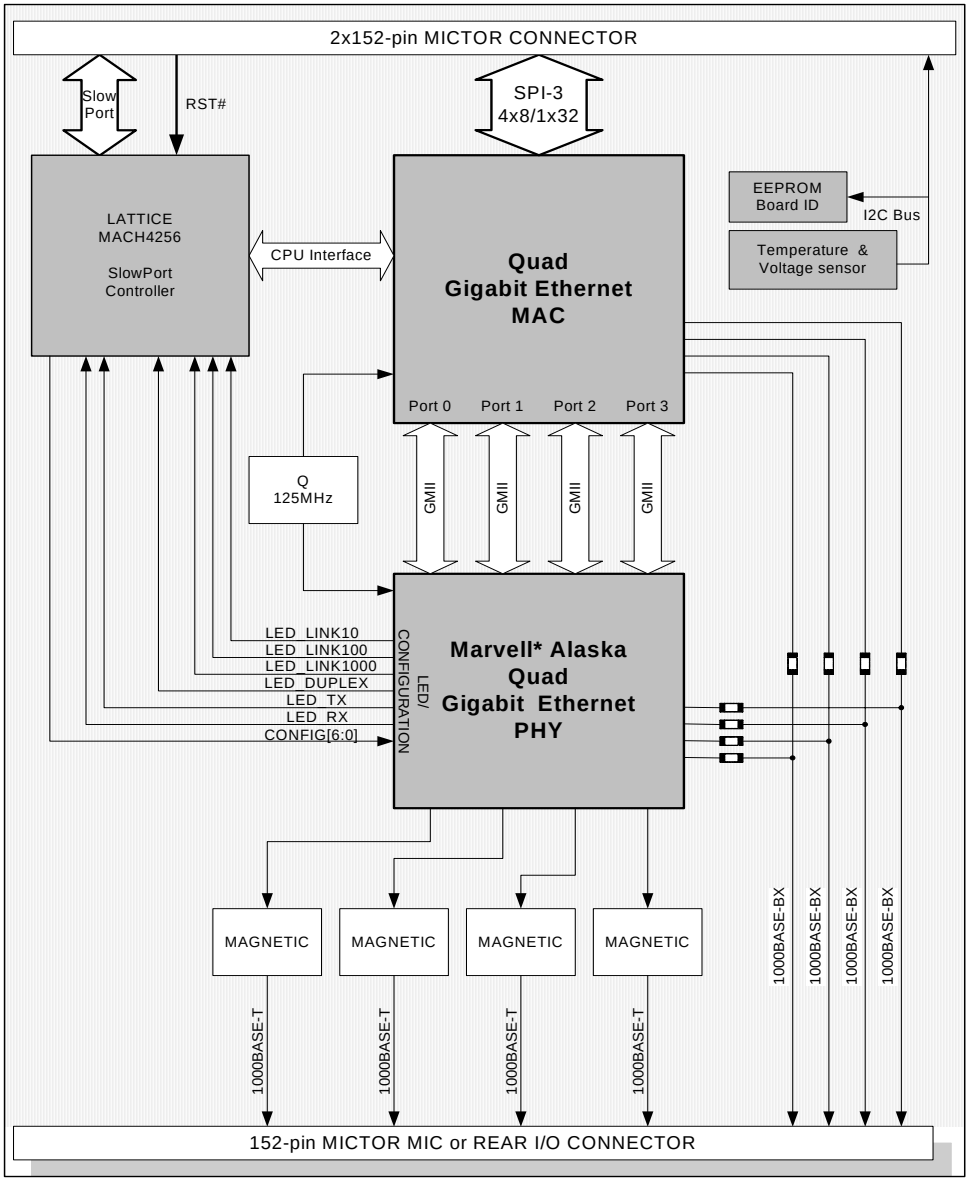
The IXF1104 is a quad port 10/100/1000 Ethernet Media Access Controllers (MAC) that provides for data transfer up to 4 Gbps. It also supports four independent 10/100/1000 Mbps full duplex PHYs which have three different interfaces: SerDes with GBIC support, GMII and RGMII.

The 88E1041 Alaska Fiber Quad contains four independent Gigabit Ethernet transceivers. Support of 1000BASE-BX for fiber Ethernet is provided through an integrated 1.25 GHz Serializer/Deserializer (SERDES). The 88E1041 supports several types of MAC interfaces: GMII, RGMII, TBI and RTBI.

The interface to the network processor is supported through a SPI3 media interface, and the PHY interfaces are Serialized/Deserialized (SerDes) with GBIC support, Gigabit Media Independent Interface (GMII) or Reduced GMII (RGMII) selected on a per-port basis.

Fiber interfaces are made available on the front panel of the IXB2850 board through a Fiber Media Interface Card (MIC-F) attached to the Quad Gigabit Ethernet Mezzanine Card. The fiber ports support Multimode fiber (MMF).

Figure 13. Quad Gigabit Ethernet media mezzanine card block diagram



The following devices are programmable by the NP through the slow port:

- IXF1104 - Quad Gigabit Ethernet MAC
- Marvell* Alaska 88E1011 - Quad Gigabit Ethernet PHY chip connected to the IXF1104 chip via the GMII interfaces

3.5 PCI Mezzanine Card Site

The PCI Mezzanine Card (PMC) site on IXB2850 boards conforms to the IEEE P1386 (PMC/CMC) standard and the VITA 32 (PrPMC) standard. The PrPMC card can be an Adjunct Processor (AP)card. The devices mounted on the PrPMC card can communicate with the NP and other baseboard devices via the PCI bus (data traffic as well as configuration and management information).

Note: IXB2850 boards can support only one PMC card at the time.

4.0 Hardware Management

4.1 Overview

On IXB2850 boards, the Board Management Controller (BMC) manages and monitors the baseboard and extension card hardware components. Management tasks include:

- Power Management
- Board Management Controller Interfaces
- Hot Swap
- Board Reset
- FRU Inventory
- Sensors and SDR
- General Status LEDs

4.2 Power Management

The BMC is the first component of an IXB2850 board to start operating after power is applied to the board (that is, after the board is inserted into a working chassis or after chassis power is switched on). The BMC controls the application of power to all other components on the IXB2850 board, that is, the components are reset initially until the BMC switches them on. However, the decision to switch power on is made by the chassis Shelf Management Controller (ShMC).

Just after board power-on, the BMC establishes a connection with the ShMC over the Intelligent Platform Management Bus (IPMB) on the backplane. After providing the ShMC with information about the board hardware, the BMC enables power for the entire board on request from the ShMC. Next, the BMC releases the reset signal for the Network Processor (NP). A detailed description of the board boot sequence can be found in Section 7.2, “Boot Sequence” on page 82.

Note: The IXB2850 board can also operate in a chassis that is not AdvancedTCA* compliant, where the ShMC is not present. To make this possible, there is a dedicated hardware jumper on the baseboard that determines whether the BMC should communicate with the ShMC. See Table 2, “Setup for IPMI” on page 17 for more information.

The BMC is also responsible for monitoring output voltage and current from the power supplies and can provide this information to the ShMC and NP.

4.3 Board Management Controller Interfaces

Figure 14 shows the hardware interfaces of the Board Management Controller (BMC).

Figure 14. Board Management Controller hardware interfaces

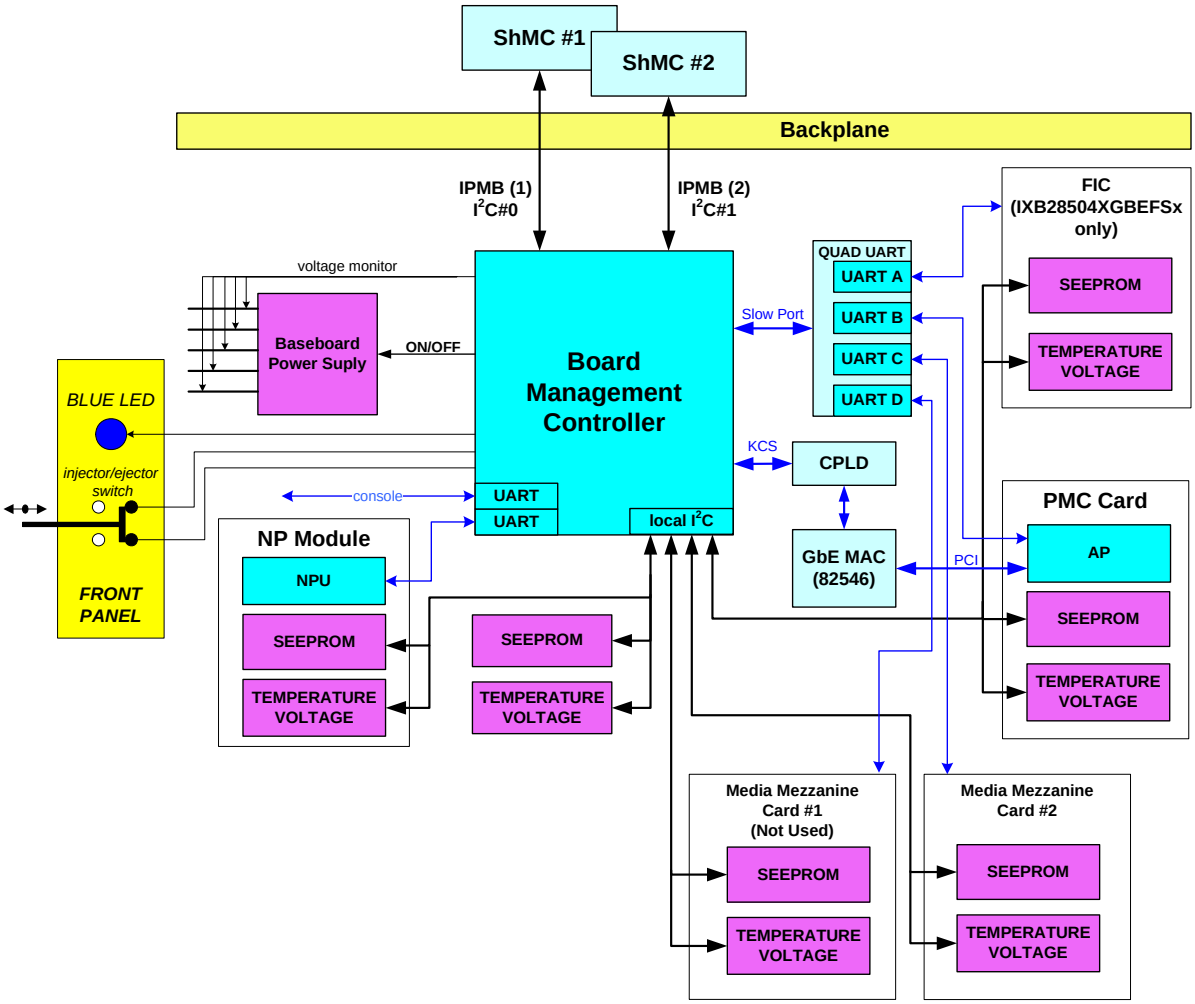


Table 3 describes the groups of interfaces supported by the BMC.

Table 3. Board Management Controller hardware interfaces

Interface	Description
Intelligent Platform Management Buses	Redundant connection to Shelf Management Controllers (two I^2C buses over the backplane)
Local I^2C buses	Connections to ID EEPROMs and sensors installed on the baseboard and extension cards
UART	Serial connection to the NP and CPU on extension cards and debug console

Table 3. Board Management Controller hardware interfaces (Continued)

Interface	Description
KCS	Connection to the Adjunct Processor on the PMC card over PCI
Power control	The BMC is responsible for power management of the whole board (switching on/off and power supply output voltage monitoring)
Front Panel	LEDs and injector/ejector switch control

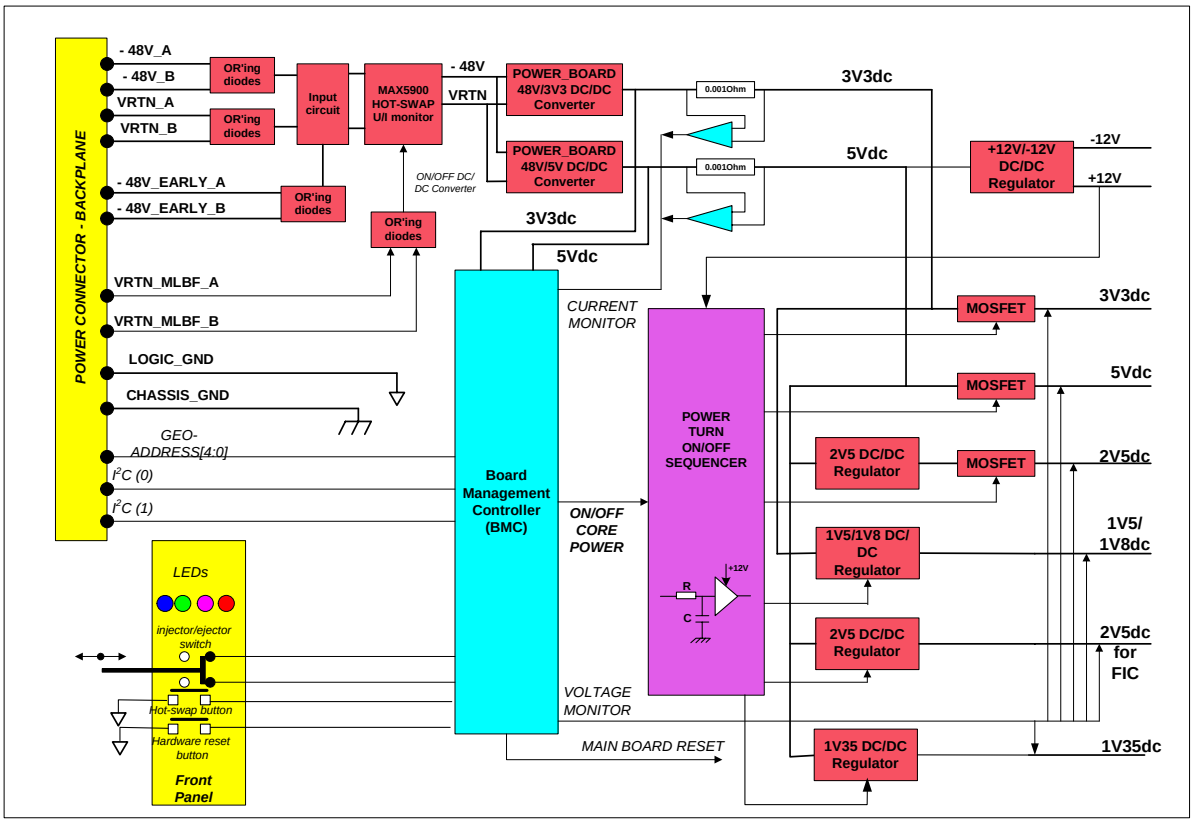
4.4 Hot Swap

The IXB2850 board design provides a mechanism for implementing High Availability (HA) systems, supported by the Hot Swap and Intelligent Platform Management Interface (IPMI) control subsystem that integrates the following elements:

- Hot Swap controller, Maxim* MAX5900
- Board Management Controller (BMC)
- Board injector/ejector switch with clamp
- LEDs circuit
- MOSFET transistors
- Power connector

Figure 15 shows the interaction between the baseboard, power supply, hot swap and BMC subsystems.

Figure 15. Baseboard power supply, hot swap and BMC subsystems



4.4.1 Staged Power Connection to the Chassis

IXB2850 baseboard connection to the chassis consists of pins with four different lengths. With this design, board connection and disconnection are carried out in three stages as described in Section 4.4.2, “Board Insertion” on page 38 and Section 4.4.3, “Board Removal” on page 39.

Table 4 lists the power circuit pins of the board power connector (Zone 1) and their connection sequence.

Table 4. Power connector pins and mating sequence

Designation	Relative Contact Length	Mating Sequence	Description
-48V_A	Medium Long	Second	-48 volt input, feed A
-48V_B	Medium Short	Third	-48 volt input, feed B
VRTN_A	Long	First	-48 volt return, feed A
VRTN_B	Long	First	-48 volt return, feed B
-48V_EARLY_A	Long	First	-48 volt input, feed A pre-charge
-48V_EARLY_B	Long	First	-48 volt input, feed B pre-charge
SHORT_PIN_A	Short	Last	Enables DC to DC converter, feed A
SHORT_PIN_B	Short	Last	Enables DC to DC converter, feed B
LOGIC_GND	Long	First	Ground reference for card logic
CHASSIS_GND	Long	First	Ground to shielding mechanics and filters

4.4.2 Board Insertion

Board insertion consists of four stages (the sequential contact of three groups of pins and the closing of the board clamp) as outlined in the subsections below.

4.4.2.1 Long Pins Contact

- ESD and static electricity are discharged to the chassis ground.
- The local board ground is connected to the main logic ground.
- The -48 volt early power supply and -48 volt ground are connected to the board. This connection makes it possible to pre-charge input capacitance.

4.4.2.2 Medium Pins Contact

- The main power is fed to the board.
- The -48V_A pin and the -48V_B pin are mated at different times.

4.4.2.3 Short Pins Contact

- The local power supply is enabled. The VRTN_MLBF_A pin and the VRTN_MLBF_B pin are mated at the same time.
- If the -48 volt supply is operational, the MAX5900 hot-swap controller is started by the BMC power supply.
- The BMC local system is started.
- The blue LED is on.

4.4.2.4 Injector/Ejector Switch Closed

- The BMC sends a notification to the ShMC that the board has just been inserted.
- The BMC enables the board power supply.
- The blue LED is off.
- Board insertion is complete.

4.4.3 Board Removal

Board removal consists of four stages (the opening the board clamp and the sequential disconnection of three groups of pins) as outlined in the following subsections.

4.4.3.1 Injector/Ejector Switch Opened

Note: The Injector/Ejector switch only needs to be opened slightly in this procedure.

- The BMC sends a notification to the ShMC that the board is going to be removed.
- The BMC disables the board power supply.
- The blue LED is lit, indicating that the board can be safely removed.

4.4.3.2 Short Pins Disconnect

Note: The Injector/Ejector switch is fully opened in this procedure.

- The blue LED is turned off.
- The BMC local system is stopped.
- The VRTN_MLBF_A pin and the VRTN_MLBF_B pin are removed at the same time. The local power supply is disabled.

4.4.3.3 Medium Pins Disconnect

- The -48V_A pin and the -48V_B pin are removed at different times.
- The main power is disconnected.

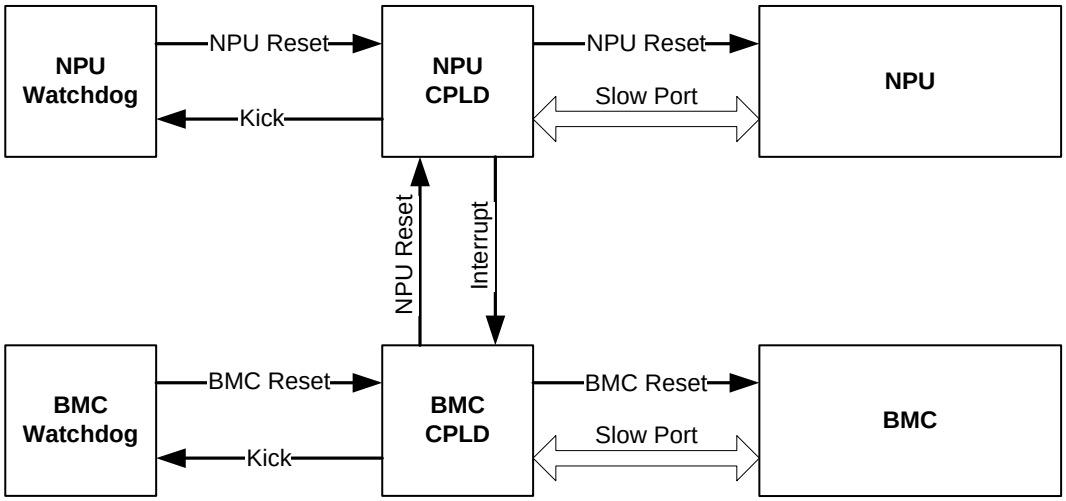
4.4.3.4 Long Pins Disconnect

- The -48 volt early power supply, -48 volt ground, logic ground, and chassis ground are disconnected from the board.
- Board removal is complete.

4.5 Board Reset

The IXB2850 board reset circuit is shown in Figure 16.

Figure 16. Board reset circuit



The circuit in Figure 16 incorporates both the BMC and the NPU. Different types of board reset scenarios are described in the following sections including:

- Entire Board Power-Down
- NPU Reset
- NPU Watchdogs
- BMC Reset
- BMC Watchdog

The reason for each reset is captured by the BMC and an appropriate event is placed in the System Event Log (see Section 8.1.3.2, “Event Logging” on page 88).

4.5.1 Entire Board Power-Down

An entire board can be powered down for any of the following reasons:

- An operator initiated board power-down
  A standard hot swap board power-down operation. This operation is always performed under the control of the ShMC.
- A BMC-initiated board power-down performed under ShMC control
  The board power-down operation is performed under ShMC control because of a hardware problem (for example, the detection of hardware overheating).
- A BMC-controlled board power-down
  This operation is performed by the BMC without ShMC participation. This kind of power-down operation is taken only in a case of a hardware problem that requires a fast response (for example, the detection of excessive hardware overheating).

4.5.2 NPU Reset

The NPU can be reset for the following reasons:

- Operator initiated NPU reset
  A software reset performed at the request of the operator from within the Boot Monitor, or from the operating system (for example, following a diagnostics or firmware upgrade).

- NPU reset by the BMC
  A reset performed by the BMC, for example, a necessary reset during the CPLD upgrade process.
- NPU watchdog reset
  An NPU reset by the watchdog mechanism in the aftermath of a software hang. To capture this kind of NPU reboot, the BMC handles a special interrupt generated by the NPU watchdog timer.

Note: An NPU reset does not affect the BMC and entire board powerup sequence. However, since the NPU is the bus master, the NPU reset does reset the PCI bus and therefore effectively resets the Quad Gigabit Ethernet Mezzanine Card and the Adjunct Processor (AP) card.

Note: The NPU CPLD does not perform a hardware NPU reset, but generates an NPU interrupt. A routine supporting this interrupt (for both Boot Monitor and Linux versions) should first indicate to the NPU CLPD that the NPU is able to reset itself. This should be done before the next watchdog expiration. In other cases, a hardware reset of the NPU is performed by the CPLD.

A warm reset of the NPU using the FRU Control IPMI command is not supported.

4.5.3 NPU Watchdogs

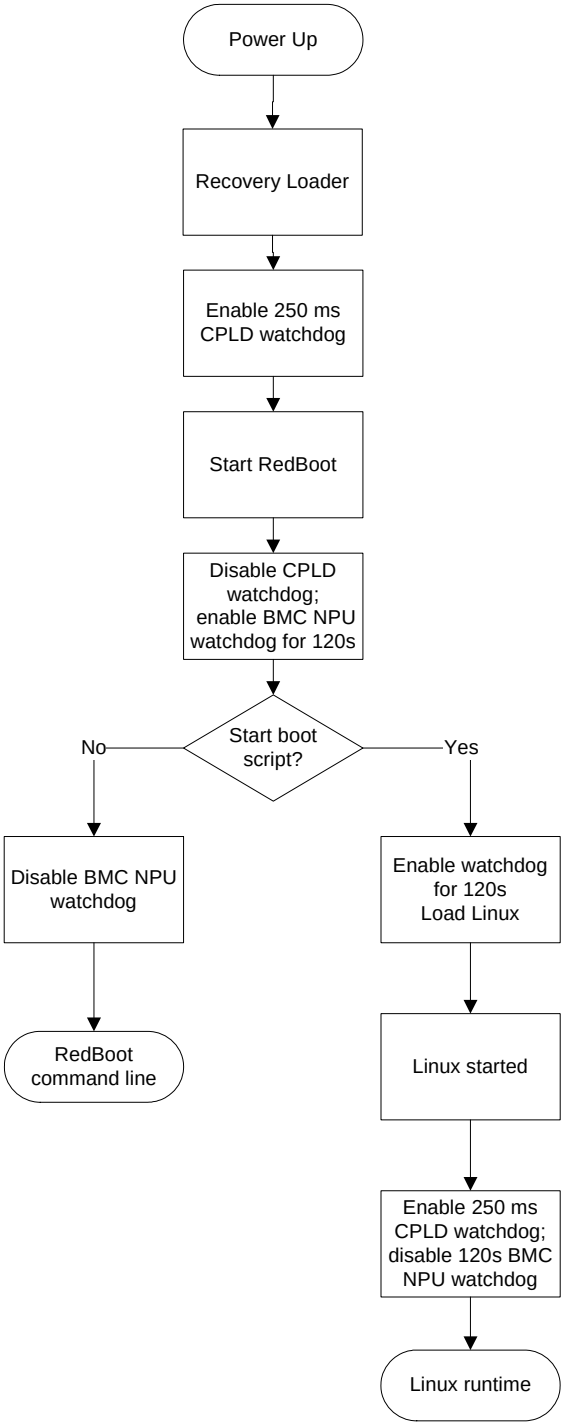
There are two watchdog mechanisms used to supervise the NPU:

- Hardware watchdog in the CPLD
- Software watchdog on the BMC

The CPLD hardware watchdog has a constant interval of 250ms and is used to supervise Boot Monitor startup, as well as the main Linux* watchdog. This watchdog is kicked by writing to a CPLD register.

The software watchdog is used to supervise Boot Monitor operation, as well as loading and startup of the Linux operating system. This watchdog is implemented within the BMC and uses BMC internal times. This watchdog has a configurable interval (up to 120 sec.) and can be enabled, disabled and kicked by the NPU using dedicated IPMI commands. Figure 17 shows the watchdog timers used by the NPU.

Figure 17. Watchdog timers used by the NPU



4.5.4 BMC Reset

The BMC can be reset for the following reasons:

- An operator-initiated BMC reset
  The software reset is performed on operator request (for example, after a firmware upgrade).
- The BMC watchdog resets
  The BMC is reset by the watchdog mechanism in the aftermath of a software hang. To capture this kind of reboot, the BMC handles a special **watchdog reset** flag in the EEPROM. This flag is set after the BMC POST and cleared before every reset performed under BMC control.

Note: These resets do not affect the NPU and overall board power-up.

4.5.5 BMC Watchdog

The BMC is protected from firmware failures by a watchdog timer mechanism. The BMC watchdog has an interval of 200 ms, is implemented in the CPLD, and is controlled by the firmware over the slow port interface.

The BMC watchdog is enabled during BMC startup and is constantly kicked by the BMC firmware; it is never disabled.

The BMC watchdog timer is compliant with the IPMI 1.5 specification (see Chapter 21. BMC Watchdog Timer Commands). IXB2850 boards support these commands as indicated in Section 8.2.1, “IPMI 1.5 Command Support” on page 90.

Steps for using the watchdog timer are:

1. “Set Watchdog Timer” - This command is used for initializing and configuring the watchdog timer.
2. “Reset Watchdog Timer” - This command is used for starting the watchdog timer from the initial countdown value that was specified in the “Set Watchdog Timer” command.
3. Repeat “Reset Watchdog Timer” for restarting the watchdog timer from the initial countdown value that was specified in the “Set Watchdog Timer” command.

The “Set Watchdog Timer” command is also used for stopping the timer and *clearing* “Timer Use Expiration flags”.

The “Get Watchdog Timer” and “Get Sensor Reading” commands retrieve the current setting of the watchdog timer.

The “Re-arm Sensor events” command clears watchdog events status and stops the watchdog timer.

4.6 FRU Inventory

IXB2850 boards consist of a number of mandatory and optional hardware modules. Each module, called a Field Replaceable Unit (FRU), contains ID EEPROM devices that store the FRU information. The BMC has access to these ID EEPROM devices over the I^2C bus. The BMC supports the following hardware modules:

- IXB2850 baseboard
- Intel® IXP2850 network processor module
- Quad Gigabit Ethernet Mezzanine Card

- Fiber Media Interface Card (MIC-F)
- Fabric Interface Card (FIC)
- PMC card

4.6.1 Multiple FRU Support

The IXB2850 board FRUs visible to the ShMC are listed in Table 5.

Table 5. FRUs visible to ShMC

FRU ID	HW module	PICMG Site Type
00h	Baseboard	00h
01h	MMC#1 – Not applicable to IXB2850 boards	07h
02h	MIC#1– Not applicable to IXB2850 boards	07h
03h	MMC#2 – Quad Gigabit Ethernet Mezzanine Card	C7h
04h	MIC#2 – Media Interface Card, MIC-F (fiber)	C7h
05h	FIC – Fabric Interface Card	C2h
06h	NPM – Network Processor Module	C0h
07h	PMC – PCI Mezzanine Card	08h

Multiple FRU support is provided according to the following rules:

Note: Text shown in boldface type indicate IPMI commands.

- The BMC reports FRUs to the ShMC in response to a **Get PICMG Properties** command.
- The **Get Address Info** command is supported for all FRUs listed in Table 5.
- Cold reset, triggered by the **FRU Control** command, is supported only for the baseboard FRU (ID=00h) and causes a hardware reset of the entire board payload.
- The **Get FRU LED Properties**, **Get FRU LED State** and **Set FRU LED State** commands are supported only for the general status LEDs located on the baseboard FRU (ID=00h). One exception is the Lamp Test functionality, where application-specific LEDs are supported also.
- All the FRUs listed in Table 5, with the exception of the baseboard FRU (ID=00h), are not hot-swappable, therefore the BMC returns the “Not supported (C1h)” code for the following commands issued for FRUs other than the baseboard FRU:
  - **Get FRU Activation Policy**
  - **Set FRU Activation Policy**
  - **Set FRU Activation**
  - **Compute Power**
  - **Get Power Level**
  - **Set Power Level**
- All sensors are controlled directly by the BMC so that the **Get Device Locator ID** command is supported only for the baseboard FRU. The “Not supported (C1h)” code is returned for this command issued for other FRUs.

Consult the IPMI v1.5 and the PICMG v3.0 specifications for more information on “mandatory” and “optional” commands.

4.6.2 FRU Information Format

The contents of FRU Information are specified in the *IPMI Platform Management FRU Information Storage Definition*. See also the PICMG 3.0 specification (Section 3.6) for more information. FRU Information can be read using the **FRU read** IPMI command. All of the FRU Information is read/write and is upgradable by the customer.

Table 6 shows the layout of the FRU Information, including the positioning of the records in the MultiRecord area. FRU Information areas always appear in the order shown in Table 6.

Table 6. FRU information layout

Area		Comment
Common Header		Mandatory
Internal Use Area		Optional
Chassis Info Area		Optional
Board Info Area		Optional
Product Info Area		Optional
MultiRecord Area	Record 1	Optional
	Record 2	Optional
	…	…
	Record x	Optional

The MultiRecord Area is used by IXB2850 boards as storage for the following parameters:

- PICMG Point-to-Point Connectivity Record – A description of backplane interfaces supported by a given hardware module
- MAC Addresses Record – Ethernet interface MAC address definition
- Sensor Devices Record – A sensor device description
- Memory Amount Record – A memory amount definition
- Processors Boot Parameters Record - BMP and NPU boot parameters

The following sections contain detailed descriptions for all these records in the MultiRecord Area.

4.6.2.1 PICMG Point-to-Point Connectivity Record

This record contains a description of the AdvancedTCA backplane interfaces supported by a given hardware module. The format of this record is defined by the PICMG 3.0 specification. The contents of the Point-to-Point Connectivity Records for the IXB2850 board are described in de tail in Section 8.4.2, “Point-to-Point Connectivity Records” on page 101.

Note: Prioritized Point-to-Point Connectivity Records are included for FRU 0. This enables support for the Electronic Keying mechanism (PICMG 3.0 options 1, 2 and 3) with Chassis Management Modules (CMMs) that are not fully standard compliant.

4.6.2.2 MAC Addresses Record

This record, proprietary to Intel, contains MAC address definitions for all Ethernet interfaces. The format of the MAC Address Record is presented in Table 7.

Table 7. MAC address record format

Offset (bytes)	Length (bytes)	Definition
0	1	**Record Type ID**. For all records defined in this document, a value of D0h (OEM) should be used.
1	1	**End of List / Version**: • 7:7 – End of List. Set to one for the last record. • 6:4 – Reserved, write as 0h • 3:0 – Record format version (=2h for this definition)
2	1	Record Length (in bytes)
3	1	**Record Checksum**. Holds the zero checksum of the record (calculated according to the FRU specification).
4	1	**Header Checksum**. Holds the zero checksum of the header (calculated according to the FRU specification).
5	3	**Manufacturer ID**. LS byte first. Write as a three byte ID = 000157h.
8	1	**Record Format Version / Intel Record ID**: • 7:5 – MAC Addresses Record format version (=0h for this definition) • 4:0 – Intel Record ID. For the MAC Addresses Record, a value 00h should be used.
9	m	**MAC Address Definition** list. A variable-length list of eight-byte MAC Address Definitions (see Table 8) totaling m bytes in length. Each MAC Address Definition describes one Ethernet interface for the referenced FRU.

Table 8. MAC address definition

Offset (bytes)	Length (bytes)	Definition
0	1	**New definition / Interface type** • 7:7 - New definition. Set to 1 for a new MAC address definition. ***Note:*** This bit is set when a **new** MAC address has been defined and needs to be burned into flash. This is an optimization that allows faster startup, that is, the MAC address burning is only done when needed. • 6:0 - Interface type. The following values should be used: 0h – external debug interface 3h – PCI GE interface 6h – SPI-3 GE interface Bh – NPU internal debug interface #1 Ch – NPU internal debug interface #2 Dh – NPU internal GE interface #1 Eh – NPU internal GE interface #2
1	1	**Interface number**
2	6	**MAC address**

4.6.2.3 Sensor Devices Record

This record, proprietary to Intel, contains a description of sensor devices. The sensor device description is used by the BMC to build the a sensor map (see Section 4.7.1.2, "Sensor Map Building" on page 55). The format of the Sensor Devices Record is given in Table 9.

Table 9. Sensors devices record format

Offset (bytes)	Length (bytes)	Definition
0	1	**Record Type ID**. For all records defined in this document, a value of D0h (OEM) should be used.
1	1	**End of List/Version** • 7:7 – End of List. Set to 1 for the last record. • 6:4 – Reserved, write as 0h • 3:0 – Record format version (=2h for this definition)
2	1	**Record Length**
3	1	**Record Checksum**. Holds the zero checksum of the record (calculated according to the FRU specification).
4	1	**Header Checksum**. Holds the zero checksum of the header (calculated according to the FRU specification).
5	3	**Manufacturer ID**. LS byte first. Write as the three byte ID = 000157h.
8	1	Record Format Version/Intel Record ID: • 7:5 – Sensor Devices Record format version (=0h for this definition) • 4:0 – Intel Record ID. For the Sensor Devices Record, a value 01h should be used.
8	m	**Sensor Descriptor** list. A variable length list of six-byte Sensor Descriptors (see Table 10) totaling m bytes in length. Each Sensor Descriptor describes one sensor device installed on the referenced FRU.

Table 10. Sensor descriptor

Offset (bytes)	Length (bytes)	Definition
0	1	**Device ID**. Sensor device identifier within the confines of being defined FRU (00h reserved for board special sensors, for example, processor or hot swap)
1	1	**Device type**. The following values should be used: • 00h – AD7416 • 01h – AD7417 • 02h – ADM1031
2	4	**Device address**. Sensor device address (for example, I^2C address, slow port address or A/D converter line number)

4.6.2.4 Memory Amount Record

This record, proprietary to Intel, contains the amount definition of memory connected to the NPU memory interfaces. The format of the Memory Amount Record is presented in Table 11.

Table 11. Memory amount record format

Offset (bytes)	Length (bytes)	Definition
0	1	**Record Type ID**. For all records defined in this document, a value of D0h (OEM) should be used.
1	1	**End of List / Version**: • 7:7 – End of List. Set to 1 for the last record. • 6:4 – Reserved, write as 0h • 3:0 – Record format version (=2h for this definition)
2	1	Record Length
3	1	**Record Checksum**. Holds the zero checksum of the record (calculated according to the FRU specification).
4	1	**Header Checksum**. Holds the zero checksum of the header (calculated according to the FRU specification).
5	3	**Manufacturer ID**. LS byte first. Write as the three byte ID = 000157h.
8	1	Record Format Version/Intel Record ID: • 7:5 – Memory Amount Record version (=0h for this definition) • 4:0 – Intel Record ID. For the Memory Amount Record, a value 02h should be used.
8	m	**Memory Definition** list. A variable-length list of five-byte Memory Definitions (see Table 12) totaling m bytes in length. Each Memory Definition describes an amount of memory installed on the referenced FRU and connected to the particular NPU interface.

Table 12. Memory definition

Offset (bytes)	Length (bytes)	Definition
0	1	**Memory type / Interface number**: 7:5 – Memory type identifier. The following values should be used: • 0h – SRAM • 1h – DRAM • 2h – Flash 4:0 – Interface number
1	4	**Memory Amount**. An amount (in kB) of memory connected to the given NPU memory interface (0h is reserved and means a co-processor (for example, TCAM) is connected to the given NPU interface).

4.6.2.5 Processors Boot Parameters Record

This record, proprietary to Intel, contains a definition of the boot parameters for a board processor (BMP or NPU). This record is used by the BMC to properly define board processors' (BMC or NPU) boot parameters. The processor boot parameters upgrade procedure is described in Section 9.4.2, "FRU Information Retrieval" on page 111.

Table 13 shows the format of the Processor Boot Parameters Record.

Table 13. Processor boot parameters record format

Offset (bytes)	Lengt (bytes)	Definition
0	1	**Record Type ID**. For all records defined in this document, a value of D0h (OEM) should be used.
1	1	**End of List/Version**: • 7:7 – End of List. Set to one for the last record. • 6:4 – Reserved, write as 0h • 3:0 – Record format version (=2h for this definition)
2	1	**Record Length**
3	1	**Record Checksum**. Holds the zero checksum of the record (calculated according to the FRU specification).
4	1	**Header Checksum**. Holds the zero checksum of the header (calculated according to the FRU specification).
5	3	**Manufacturer ID**. LS byte first. Write as the three byte ID = 000157h.
8	1	Record Format Version/Intel Record ID: • 7:5 – Processors Boot Parameters Record version (=0h for this definition) • 4:0 – Intel Record ID. For the Processors Boot Parameters Record, a value 03h should be used
9	m	**Processor Descriptor** list. A variable-length list of variable-length Processor Descriptors (see Table 14, Table 15, and Table 16) totaling m bytes in length. Each Processor Descriptor describes boot parameters for one processor installed on the referenced FRU.

Table 14. Processor descriptor

Offset (bytes)	Length (bytes)	Definition
0	1	**Boot Parameters Definition Length**
1	1	**Version/New parameters/Processor ID**: • 7:5 – Processor descriptor format version (=0h for this definition) • 4:4 – New parameters. Set to one for newly defined parameters. ***Note:*** This bit is set when **new** parameters have been defined and need to be translated into flash configuration. This is an optimization that allows faster startup, that is, the translation is only done when needed. • 3:0 – Processor identifier. The following values should be used: - 0h BMC - 1h NPU#1 - 2h NPU#2 - 3h AP
2	m	**Processor boot parameters**. See Table 15 and Table 16.

Table 15. BMC boot parameters definition

Offset (bytes)	Length (bytes)	Definition
0	1	**Boot flags**: • 7:5 – **BMC Boot Parameters definition format version** (=0h for this definition) • 4:4 – **POST type**. A flag indicating type of Power-On-Self-Test (POST) performed by BMC during board startup. This parameter can have one of two values: - 0b Long POST (all tests will be performed) - 1b Short POST (only selected tests will be performed) • 3:0 – **Reserved**
1	2	**Short POST contents** A bitmask describing which tests have to be performed during POST. If a given bit is set then a corresponding test is performed during BMC POST: • 15:15 – IRQ • 14:14 – GPIO • 13:13 – Flash • 12:12 – Internal RAM • 11:11 – External RAM • 10:10 – IPMB • 9:9 – local I^2C buses • 8:8 – embedded UARTs • 7:7 – external quad UART • 6:6 – Keyboard Controller Style (KCS) • 5:5 – LEDs • 4:4 – Timers • 3:0 – Reserved

Table 16. NPU boot parameters definition

Offset (bytes)	Length (bytes)	Definition
0	1	**Boot Flags**: • 7:5 – **NPU Boot Parameters definition format version** (=1h for this definition) • 4:4 – **POST type**; a flag indicating the type of Power-On-Self-Test performed by the Boot Monitor during board startup. This parameter can have one of two values: - 0b Long POST - 1b Short POST • 3:3 – **Soft reset**; a flag indicating whether the soft reset is being performed. This flag is used by POST to perform the appropriated POST set and can have one of two values: - 0b Normal startup - 1b Soft reset • 2:2 – **Run-time image loading**; a flag indicating whether the Boot Monitor shall load and execute the run-time image for the NPU. This flag can have one of two values: - 0b – Image load; the Boot Manager attempts to load and execute run-time image code - 1b – Command prompt; the Boot Manager does not attempt to load the run-time image code, displays a command prompt and waits for operator action (for example, diagnostics execution) • 1:1 – **SRAM initialization**; a flag determining the SRAM initialization method. This flag is used by RedBoot* to initialize SRAM memory using one of the following methods: - 0b – Initialized with zero; the entire SRAM memory is filled with zeros - 1b – Initialized with current value; every SRAM memory location will be rewritten with the current value of such memory location to correct possible parity errors • 0:0 – **Reserved**
1	1	**Script timeout** – Number of seconds of delay before launching the boot script

Table 16. NPU boot parameters definition

Offset (bytes)	Length (bytes)	Definition
2	1	**Ethernet loading flags**: • 7:7 – **Ethernet tracing**; a flag indicating whether traffic that comes over Ethernet interfaces should be traced on the Boot Monitor console - 0b Traffic is traced - 1b Traffic is not traced • 6:6 – **Debug Ethernet IP configuration type**; a flag indicating type of IP configuration used for debug Ethernet interface during run-time image loading. This flag can have one of two values: - 0b – Automatic; IP configuration for debug Ethernet interface should be obtained via BOOTP protocol - 1b – Static; IP configuration stored as boot configuration parameter are used for the debug Ethernet interface • 5:5 – **Base Interface #1 IP configuration type**; a flag indicating the type of IP configuration used for base interface #1 during run-time image loading. This flag can have one of two values: - 0b – Automatic; the IP configuration for base interface #1 should be obtained via BOOTP protocol - 1b – Static; the IP configuration, stored as boot configuration parameter, is used for base interface #1 • 4:4 – **Base Interface #2 IP configuration type**; a flag indicating the type of IP configuration used for base interface #2 during run-time image loading. This flag can have one of two values: - 0b – Automatic; the IP configuration for base interface #2 should be obtained via BOOTP protocol - 1b – Static; the IP configuration, stored as boot configuration parameter, is used for base interface #2 • 3:0 – **BOOTP request number**; a number of BOOTP requests that are sent by the Boot Manager over Ethernet interfaces in order to obtain IP configuration for these interfaces
3	4	**Debug Ethernet IP address** – A static IP address used for run-time image loading over the debug Ethernet interface (valid only for static IP debug Ethernet configuration)
7	1	**Debug Ethernet subnet mask** – A number of set bits in the subnet mask used for run-time image loading over debug Ethernet interface (valid only for static IP debug Ethernet configuration)
8	4	**Debug Ethernet default gateway** – A default gateway IP address used for run-time image loading over the debug Ethernet interface (valid only for static IP debug Ethernet configuration)
12	4	**Base Interface #1 IP address** – A static IP address used for run-time image loading over base interface #1 (valid only for static IP base interface #1 configuration)
16	1	**Base Interface #1 subnet mask** – A number of set bits in the subnet mask used for run-time image loading over base interface #1 (valid only for static IP base interface #1 configuration)
16	4	**Base Interface #1 default gateway** – The default gateway IP address used for run-time image loading over base interface #1 (valid only for static IP base interface #1 configuration)
21	4	**Base Interface #2 IP address** – A static IP address used for run-time image loading over base interface #2 (valid only for static IP base interface #2 configuration)
25	1	**Base Interface #2 subnet mask** – A number of set bits in the subnet mask used for run-time image loading over base interface #2 (valid only for static IP base interface #2 configuration)
26	4	**Base Interface #2 default gateway** – The default gateway IP address used for run-time image loading over base interface #2 (valid only for static IP base interface #2 configuration)
30	m	**Image Loading Parameters Set list.** A variable-length list of variable-length Image Loading Parameters Sets (see Table 17) totaling m bytes in length. Each Image Loading Parameter Set describes parameters necessary to download and run one run-time image.

Table 17. Flash run-time image loading parameters set format

Offset (bytes)	Length (bytes)	Definition
0	1	Loading Parameters Set flags: • 7:7 – **End of List**. Set to one for the last record. • 6:4 – **Set Type** – Image Loading Parameters Set type. For flash loading parameters a value 0h shall be used. • 3:3 – **Image Run Method**: 0b – using *execute* command 1b – using *go* command ***Note:*** The *execute* command is equivalent to a jump instruction, where there is no return to RedBoot. The *go* command is equivalent to a call instruction, where after code execution, there is a return to RedBoot. For example, *go* is used to run diagnostics, *execute* is used to run the operating system. • 2:0 – **Reserved**
1	1	Loading Parameters Set length
2	m	**FIS partition name**. The name of the FIS partition where a given run-time image should be loaded from. The format is an ASCII string, where 0 is not required at the end of the string.

Table 18. Serial interface run-time image loading parameters set format

Offset (bytes)	Length (bytes)	Definition
0	1	Loading Parameters Set flags • 7:7 - **End of List**. Set to one for the last record. • 6:4 – **Set Type**; Image Loading Parameters Set type. For serial loading parameters, a value of 1h is used. • 3:3 – **Image Run Method**: 0b – Using the *execute* command 1b – Using the *go* command ***Note:*** The *execute* command is equivalent to a jump instruction, where there is no return to RedBoot. The *go* command is equivalent to a call instruction, where after code execution, there is a return to RedBoot. For example, *go* is used to run diagnostics, *execute* is used to run the operating system. • 2:0 – **Reserved**
1	4	**Memory address** – The address of the RAM location where a loaded image should be written to.

Table 19. Ethernet interface run-time image loading parameters set format

Offset (bytes)	Length (bytes)	Definition
0	1	Loading Parameters Set flags: • 7:7 – **End of List**. Set to one for the last record. • 6:4 – **Set Type**; Image Loading Parameters Set type. The following values are used for Ethernet loading parameters: 2h – Debug Ethernet interface 3h – Base Interface #1 4h – Base Interface #2 • 3:3 – **Image Run Method**: 0b – Using the *execute* command 1b – Using the *go* command ***Note:*** The *execute* command is equivalent to a jump instruction, where there is no return to RedBoot. The *go* command is equivalent to a call instruction, where after code execution, there is a return to RedBoot. For example, *go* is used to run diagnostics, *execute* is used to run the operating system. • 2:0 – **Reserved**
1	1	**Loading Parameters Set length**

Table 19. Ethernet interface run-time image loading parameters set format (Continued)

Offset (bytes)	Length (bytes)	Definition
2	4	**Memory address** – The address of the RAM location where a loaded image should be written to.
6	4	**Server IP address** – The IP address of the server hosting run-time image that should be loaded over Ethernet interface.
10	m	**File name** – The full path file name of run-time image that should be downloaded over Ethernet interface and executed. The format is an ASCII string, where 0 is not required at the end of the string.

4.6.3 FRU Information Contents

Table 20 lists the contents of FRU information describing specific IXB2850 board hardware modules.

Table 20. FRU contents

IXB2850 baseboard	Common Header Board Info Area PICMG Point-to-Point Connectivity Record for base interface and fabric interface channel 1 MAC Address Records for the Dual Port Gigabit Ethernet Controller (Intel® 82546), SPI-3 Gigabit Ethernet MAC Controller (Intel® IXF1104) and external debug Ethernet (CS8900) Sensor Device Records for all sensor devices on the baseboard Memory Amount Record describing flash memory installed on the baseboard Processor Boot Parameters Records for the BMC
IXP2850 network processor module	Common Header Board Info Area Sensor Device Records for all sensor devices on the IXP2850 network processor module Memory Amount Record describing DRAM, SRAM and flash memory installed on IXP2850 network processor module Processor Boot Parameters Records for the NPU
Quad Gigabit Ethernet Mezzanine Card	Common Header Board Info Area MAC Address Records for SPI-3 Gigabit Ethernet MAC Controller Sensor Device Records for all sensor devices on the mezzanine card
Fiber MIC (MIC-F)	Common Header Board Info Area Sensor Device Records for all sensor devices on MIC-F
Fabric Interface Card	Common Header Board Info Area PICMG Point-to-Point Connectivity Record for fabric interface channel 2 MAC Address Records for SPI-3 Gigabit Ethernet MAC Controller (IXF1104) Sensor Device Records for all sensor devices on the FIC

4.6.4 FRU Information Upgrade

The FRU Information for each IXB2850 hardware module can be upgraded by the NPU or the ShMC using the IPMI protocol (file transfer, as well as EEPROM programming operation). The standard IPMI commands for reading and writing FRU data can be used for this purpose. These operations can be also be performed using the BMC console (Xmodem for file transfer and console commands for EEPROM programming operations).

There is the possibility of upgrading the entire FRU Information. The standard BMC upgrade procedure (see Section 8.6, “Non-Volatile Storage Programming” on page 104) should be used for this purpose.

4.7 Sensors and SDR

IXB2850 boards have the capability of providing low-level (temperature, voltage, and current) diagnostics for selected devices installed on the board characterized by the highest heat emission or power consumption. These diagnostics capabilities are used by the BMC to control the health of a board and prevent damage. The BMC is responsible for controlling:

- The temperature of selected baseboard and extension card devices
- The baseboard and extension card voltage
- The current drawn from selected power supplies

The BMC uses sensor devices connected to local I^2C bus to control the above-mentioned board parameters. If temperature, voltage or current on a board is higher or lower than the nominal values described in the Sensor Device Records (SDR), the BMC is responsible for sending notification to the ShMC or possibly shutting down entire board.

After booting, the BMC polls only those sensors that do not require initialization. Sensors requiring initialization must be initialized by the ShMC before they are polled by the BMC. Once the BMC detects that a given sensor has been initialized, the sensor is included in the set of monitored sensors.

4.7.1 Sensor Map

The BMC is responsible for providing the ShMC and the NPU with the definition of all board sensors. In response to a request from the ShMC or the NPU, the BMC provides the requested readings taken from a specified sensor. To make this possible, the BMC builds and maintains a sensor map. Each entry in this table describes one sensor. The sensor number is used as the sensor map table index. Note that this table contains physical sensors (for example, temperature or voltage) as well as logical sensors (for example, a processor).

4.7.1.1 Sensor Numbering in the SDR

Each board sensor has its own unique number defined in the Sensor Data Record (SDR). This number is used to address a given sensor during initialization and reading operations. The IXB2850 board uses multi-sensor devices that contain a few physical sensors within the confines of one physical device. For that reason, a raw I^2C bus address is not sufficient information to address a physical sensor. The format of the SDR sensor number is shown in Figure 18.

Figure 18. SDR sensor number format

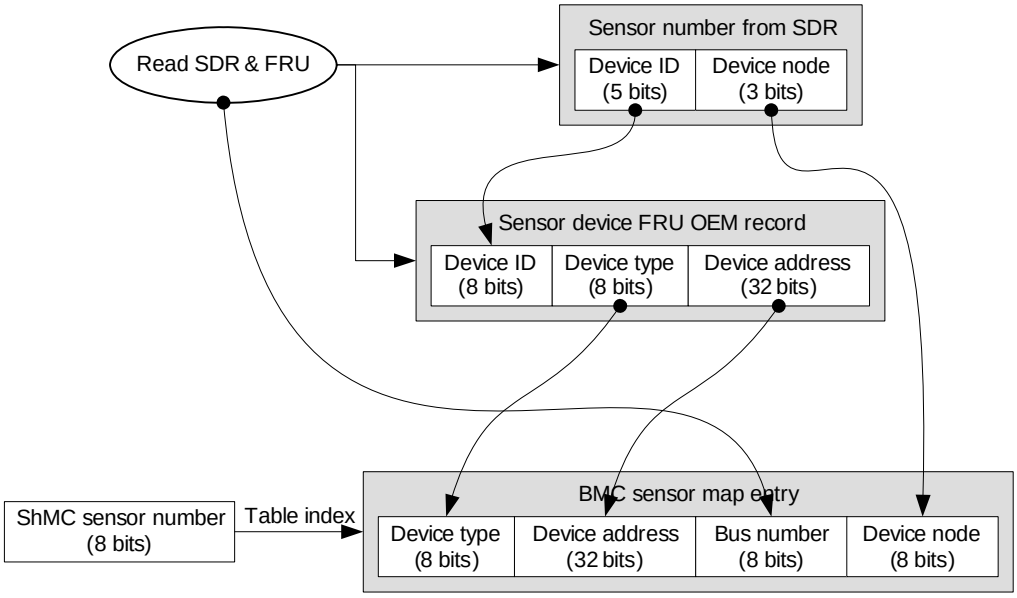
The SDR sensor number consists of two parts:

- **Device ID** – A device identifier pointing to a record in the board FRU Information fully describing the physical sensor device. ID=00h has been reserved for a logical sensor on a board (for example, processor or hot swap sensor).
- **Device node** – The number of a physical sensor within the confines of the referenced physical multi-sensor device.

4.7.1.2 Sensor Map Building

The BMC builds the Sensor Map table based on the Sensor Device Records (SDRs) taken from all IXB2850 board hardware modules. For this purpose, the BMC uses the sensor device definitions taken from the hardware module FRU Information. Figure 19 shows the Sensor Map entry build process.

Figure 19. Sensor map entry building process



Note that a bus number is determined by the BMC based on the SDR and a FRU Information read operation and the number of the bus (for example, local I^2C bus) from which a given SDR has been read.

4.7.2 Sensor Devices

The BMC supports different sensor devices located on an IXB2850 baseboard and its extension cards:

- Temperature monitoring sensors (AD7416)
- Temperature monitoring sensor (ADM1031)
- Temperature and voltage sensor (AD7417)†

Note: † On IXB2850 baseboards, current is monitored using voltage sensors.

All sensor devices placed on a particular hardware module are described by the Sensor Data Record (SDR) included in the FRU Information for the referenced hardware module (see Section 4.7.3 following).

4.7.3 Sensors Data Records

Each IXB2850 board hardware module contains a Sensor Data Record (SDR) describing all sensors placed on a given hardware module. The SDR format is described by the IPMI 1.5 specification. Table 21 lists sensor types supported by IXB2850 boards.

Table 21. IXB2850 board sensor types

Sensor Type	SDR Type	Comments
Temperature sensor	Full Sensor Record (SDR Type 01h)	Sensor Type code = 01h
Voltage sensor	Full Sensor Record (SDR Type 01h)	Sensor Type code = 02h
Current sensor	Full Sensor Record (SDR Type 01h)	Sensor Type code = 03h
FRU Device	FRU Device Locator Record (SDR Type 11h)	Required by PICMG 3.0
Management Controller Device	Management Controller Device Locator (SDR Type 12h)	Required by PICMG 3.0
FRU Hot Swap sensor	Full Sensor Record (SDR Type 01h)	Sensor Type code = F0h (PICMG specific) Required by PICMG 3.0
IPMB Physical Link sensor	Full Sensor Record (SDR Type 01h)	Sensor Type code = F1h (PICMG specific) Required by PICMG 3.0
Processor	Compact Sensor Record (SDR Type 02h)	Sensor Type code = 07h

4.7.4 Processor Sensors

The IXB2850 board processor topology is described in the SDR. Each processor contains its own definition in the form of a Compact Sensor Record (SDR Type 2). Information stored in this record is used by the ShMC to handle event messages generated by that particular processor and to configure it as a sensor. Table 22 describes the format of the Compact Sensor Record processor as a sensor device.

Table 22. Processor SDR format

Byte(s)	Field Name	Value	Comments
1:2	Record ID		The Record ID is used by the Sensor Data Repository device for record organization and access. It is not related to the sensor ID.
3	SDR Version	51h	IPMI 1.5 compatible
4	Record Type	02h	Compact Sensor Record
5	Record Length		
6	Sensor Owner ID		Modified at runtime depending on the slot occupied by the board
7	Sensor Owner LUN	00h	
8	Sensor Number		See Section 4.7.1.1, “Sensor Numbering in the SDR” on page 54.
9	Entity ID	03h	Processor
10	Entity Instance	01h	
11	Sensor Initialization	22h	Init Events, event generation enabled
12	Sensor Capabilities	40h	Sensor Auto Re-arm Support

Table 22. Processor SDR format (Continued)

Byte(s)	Field Name	Value	Comments
13	Sensor Type	07h	Processor
14	Event / Reading Type Code	70h	OEM Discrete with the following defined offsets: • 00h – **Booting** • 01h – **POST** • 02h – **Diagnostics**
15:16	Assertion Event Mask	0007h	Three bit mask
17:18	Deassertion Event Mask	0000h	No deassertion events
19:20	Discrete Reading Mask	0000h	Reading not supported
21	Sensor Units 1	C0h	None
22	Sensor Units 2 - Base Unit	00h	Unspecified
23	Sensor Units 2 - Modifier Unit	00h	Unused
24:25	Sensor Record Sharing	00h	No sharing
26	Positive-going Threshold Hysteresis value	00h	No Threshold Hysteresis
27	Negative-going Threshold Hysteresis value	00h	No Threshold Hysteresis
28	reserved	00h	
29	reserved	00h	
30	reserved	00h	
31	OEM		OEM information • 7:5 – **reserved** • 4:1 – **Processor ID**: - 00h BMC - 01h NPU#1 - 02h NPU#2 - 03h PrPMC (for example, an Adjunct Processor) • 0:0 – **Virtual Console**: - 0b Not supported - 1b Supported
32	ID String Type/Length Code		
33:+N	ID String Bytes		

IXB2850 boards contain two processors acting as sensors, the BMC and the NPU. The board can also support a PrPMC (for example, an Adjunct Processor) acting as a sensor.

4.7.4.1 Processor Events

The event messages generated by the CPU sensors have an IPMB message format as specified in the IPMI 1.5 specification. Table 23 describes the format of the CPU Event Message format.

Table 23. CPU event message format

Data	Byte	Data Field
Request Data	1	**Event Message Revision**. The revision number is **04h** for Event Messages that comply with the IPMI 1.5 specification.
	2	**Sensor Type**. The Sensor Type value from the SDR describing the given CPU sensor is used.
	3	**Sensor Number.** The Processor ID from the SDR describing the given CPU sensor is used.
	4	Event Direction/Event Type: • 7:7 – **Event Direction**; indicates the event transition direction. The value 0b (Assertion Event) is used. • 6:0 – **Event Type**; the Event/Reading Type Code value from the SDR describing the given CPU sensor is used.
	7:5	**Event Data** – The contents and format of this field is specified in: • Table 51, "Event data 1 field format" on page 115 • Table 52, "IPMI event data for boot class" on page 116 • Table 53, "IPMI event data for the POST class" on page 116 • Table 54, "IPMI event data for the diagnostics class" on page 117
Response Data	1	**Completion Code**

4.7.4.2 BMC Support for Processor Sensors

Since the board processor does not have a direct connection to the ShMC, the BMC is responsible for supporting the exchange of data between the ShMC and the board processors.

Event Storing

A board processor, acting as a sensor, generates events and stores them in the BMC System Events Log (SEL). The IPMI event message sent by the CPU to the BMC contains a Sensor Number set to the Processor ID taken from SDR record describing a given processor. When the BMC stores this event in the SEL, it updates this field with a value assigned to a particular CPU during the building of the BMC Sensor Map.

Event Forwarding

A board processor, acting as a sensor, generates events and sends them to the ShMC. The IPMI **Send Message** command with Channel number = 0h is used for this purpose (see Section 8.2.3, "IPMI Bridging" on page 95). The IPMI event message encapsulated within **Send Message** command contains a Sensor Number set to the Processor ID taken from the SDR record describing a given processor. When the BMC forwards this event message to the ShMC, it updates this field with a value assigned to a particular CPU during the building of the BMC Sensor Map.

4.7.5 Sensors Polling

The BMC continuously polls all initialized physical (temperature, voltage and current) sensors during normal IXB2850 board operation. When the BMC detects that a particular sensor reading exceeds one of the thresholds defined in the SDR record, it sends an appropriate event to ShMC. The ShMC is responsible for responding to these events appropriately (for example, powering a board off).

4.7.5.1 Thermal Protection

The SDR for each temperature sensor defines three thresholds for temperature exceeding typical values:

- Upper non-critical threshold
- Upper critical threshold
- Upper non-recoverable threshold

When the value read from the temperature sensor exceeds the upper non-recoverable threshold, the BMC is responsible for immediately powering a board off without waiting for instructions from the ShMC.

4.8 General Status LEDs

IXB2850 boards provides four general status LEDs on the board front panel. The BMC controls each individual LED on a front panel.

The general status LEDs on the board front panel are connected to BMC via a local CPLD.

The BLUE LED is used for hot swap purposes and its function is described in Section 4.4, “Hot Swap” on page 37.

The other general status LEDs have the following meaning:

- LED 1 – OOS (out of service)
- LED 2 – HEALTH (operational)
- LED 3 – ALS (attention) [not included on some custom boards]

The BMC Agent allows the BMC and ShMC to control application specific LEDs. The BMC Agent receives IPMI messages and uses the baseboard driver API to turn application LEDs on and off. Customer applications can also use the baseboard driver API to control these LEDs.

4.8.1 Multi-Color LEDs

The general status LEDs, with the exception of the blue LED, are multi-color (RGB LEDs). The BMC can switch each LED on/off and set the color based on standard IPMI requests received from the ShMC. All colors defined in the description of the **Get LED Color Capabilities** (NetFn=2Ch/2Dh, CMD=06h) are supported. The general status LEDs factory default colors are as follows:

- BLUE LED – blue
- LED 1 – red
- LED 2 – green
- LED 3 – amber (not included on some custom boards)

4.8.2 Application-Specific LEDs

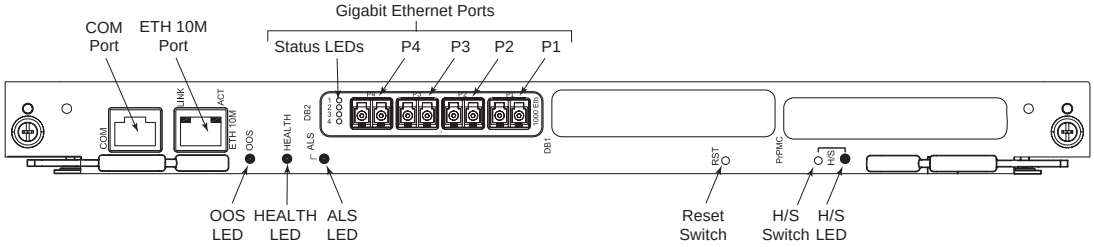
The BMC on IXB2850 boards does not have the ability to directly control the application-specific LEDs on an extension card (for example, the Quad Gigabit Ethernet Mezzanine Card). It can control these LEDs indirectly via the BMC Agent running on the NPU. To force all application-specific LEDs on or off during the lamp test operation, the BMC must send an IPMI command to BMC Agent, which is responsible for LED control.

5.0 Controls, Indicators and Connectors

5.1 Front Panels

Figure 20 shows the controls, indicators and connectors on the front panel of Intel NetStructure® IXB28504XGBEFSx boards.

Figure 20. IXB28504XGBEFSx board front panel



The function of each component is described in the following subsections.

5.1.1 COM Port

The UART-based serial port (J31) that is maintained for connecting to the Board Management Controller (BMC) and the Network Processor (NP). The console is provided by dual UART signals to the slow port of the network processor. See Figure 14, “Board Management Controller hardware interfaces” on page 36 for the inter-connections between these devices.

The UART-based console is not intended for permanent connection when deployed in the field, but is intended for the following debugging/development actions:

- Downloading firmware to the NP flash memory (using the XModem protocol)
- Console access to the BMC and the NP
- Running diagnostic commands
- Running Boot Monitor console commands

The COM port provides connection for two consoles (one for the NPU and one for the BMC) and is not configurable. See Section 12.8.1, “Debug Console Cable Specification” on page 165 for pinout information.

5.1.2 ETH 10M Port

A 10 Mbps Ethernet RJ-45 debug port (J30) that is not intended for permanent connection when deployed in the field, but can be used for the following debugging/ development actions:

- Downloading firmware to the NPU flash memory (using the TFTP protocol)
- FTP server access and mounting file system for NPU over NFS

- Connecting to the Intel® IXA SDK Developer's Workbench via a Workbench backend agent. The Intel® IXA SDK Developer's Workbench is included in the Intel® IXA SDK Tools CD image. A patch is required to use the SDK with IXB2850 boards. See Section 1.1, "Product Description" on page 9 for a pointer to the product web page from which the patch can be downloaded.

5.1.3 Gigabit Ethernet Ports

For IXB28504XGBEFSx boards, the Quad Gigabit Ethernet Mezzanine Card operates in conjunction with a Fiber Media Interface Card (MIC-F) to provide four 1000BASE-SX fiber ports with status LEDs to the front panel.

For each port, the LED status depends on the state of the Transmit Disable and Signal Detect pins of the corresponding optical transceiver on the MIC-F. The default is that the LED is OFF. When the port is enabled for transmission, the LED is ON with an amber color; when port is enabled and an optical signal is detected, the LED is ON with a green color.

5.1.4 Cable Clamp

A clamp that facilitates the tidy routing of the cables connecting to the Gigabit Ethernet ports.

5.1.5 OOS LED

An amber "Out Of Service" LED that is controlled by the Board Management Controller (BMC). When lit, indicates that the board is out of service.

5.1.6 HEALTH LED

A green "Health" LED that is controlled by the BMC. When lit, indicates that the board is active.

5.1.7 ALS LED

A red "Attention" LED. Currently not supported. Not included on some custom boards.

5.1.8 RST Button

Currently not supported.

5.1.9 H/S Button

Currently not supported.

5.1.10 H/S LED

A blue "Hot Swap" LED that is controlled by the IPMC module and mainly used during power-up and hot swap processes. Provides the following indications:

- When lit and not flashing, indicates that the board is ready for Hot Swap removal.
- When flashing with a duty cycle of 100ms on, 900ms off, indicates one of the following:
  - The ShMC is communicating with the IPMC.
  - Request for removal of the board has been initiated.

- When flashing with a duty cycle of 900ms on, 100ms off, indicates one of the following:
  - — Board insertion was denied.
  - — A power-up error was reported to the ShMC; a chassis error occurred.

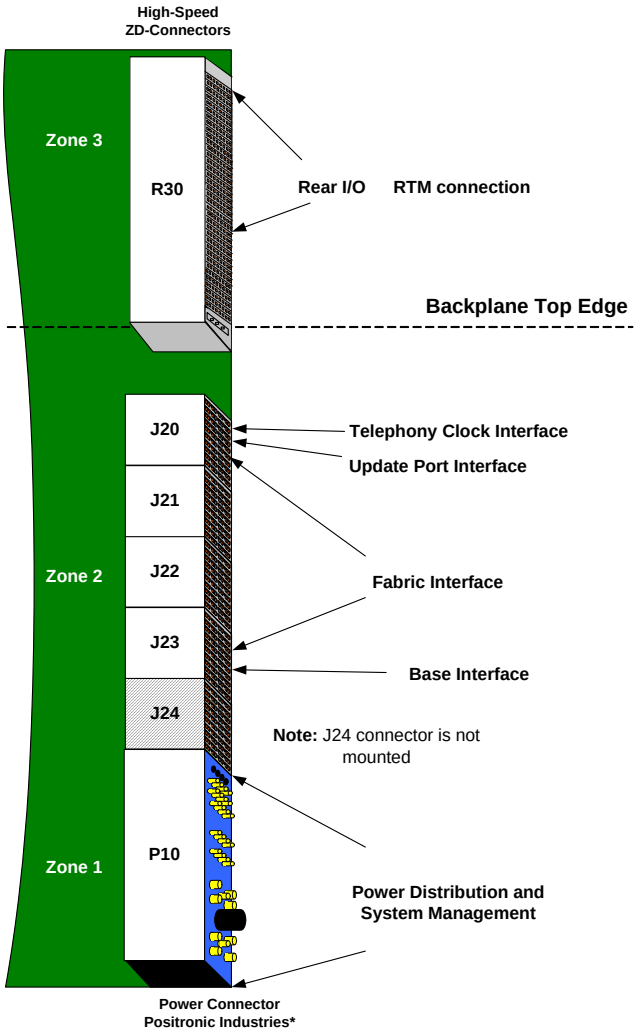
5.2 Backplane Connectors

Backplane connectors on IXB2850 boards are divided into three zones:

- Zone 1: Power Distribution and System Management
  P10 connector; Power Distribution and System Management, which provides a connection to dual-redundant power and system management connections.
- Zone 2: Data Transport Connection
  J20, J21, J22, J23, and J24 connectors; Data Transport, which provide connections to five ZD high-speed connectors.
- Zone 3: RTM (Not Used)
  R30 connector; rear panel I/O; not used on the IXB2850 boards.

Figure 21 shows the locations of the backplane connectors.

Figure 21. Backplane connector locations



5.2.1 Zone 1: Power Distribution and System Management

Connector P10 is a Positronic Industries* VPB30W8 connector for power distribution and system management. The connector provides:

- Power contacts
- Analog test contacts
- Hardware management contacts

Figure 22 shows the layout of the contacts on the connector and Table 24 provides information about the contacts.

Figure 22. Zone 1 power and system management connector layout

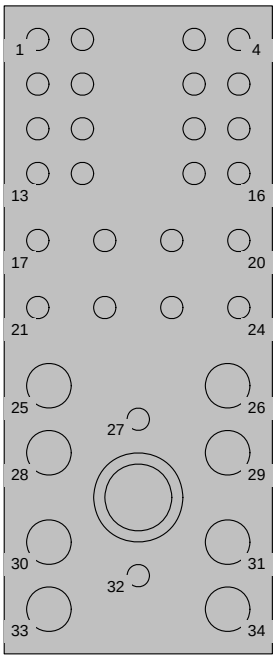


Table 24. Zone 1 power and system management connector contact information

Contact Number	Designation	Mating Sequence On Insertion
1	Spare	N/A
2	Spare	N/A
3	Spare	N/A
4	Spare	N/A
5	HA0 Hardware Address Bit 0	Fourth
6	HA0 Hardware Address Bit 1	Fourth
7	HA0 Hardware Address Bit 2	Fourth
8	HA0 Hardware Address Bit 3	Fourth
9	HA0 Hardware Address Bit 4	Fourth
10	HA0 Hardware Address Bit 5	Fourth
11	HA0 Hardware Address Bit 6	Fourth
12	HA0 Hardware Address Bit 7 (odd parity bit)	Fourth
13	SCL A IPMB Clock, Port A	Fourth
14	SDA A IPMB Data, Port A	Fourth
15	SCL B IPMB Clock, Port B	Fourth
16	SDA B IPMB Data, Port B	Fourth
17	Spare	N/A
18	Spare	N/A
19	Spare	N/A

Table 24. Zone 1 power and system management connector contact information

Contact Number	Designation	Mating Sequence On Insertion
20	Spare	N/A
21	Spare	N/A
22	Spare	N/A
23	Spare	N/A
24	Spare	N/A
25	EMI Chassis Ground	First
26	Logic Ground	First
27	ENABLE_B	Fifth
28	VRTN_A	First
29	VRTN_B	First
30	-48V_EARLY_A	First
31	-48V_EARLY_B	First
32	ENABLE_A	Fifth
33	-48V_A	Second
34	-48V_B	Third

5.2.2 Zone 2: Data Transport Connection

The J20, J21, J22, J23, and J24 connectors are Z-PACK HM-Zd connectors designed specifically for high-speed differential applications. Figure 23 shows a single ZD connector. This is an ERNI/Tyco/AMP connector. Its plug (located on the baseboard) has part number 1469001-1.

Figure 23. Zone 2 data transport connector layout

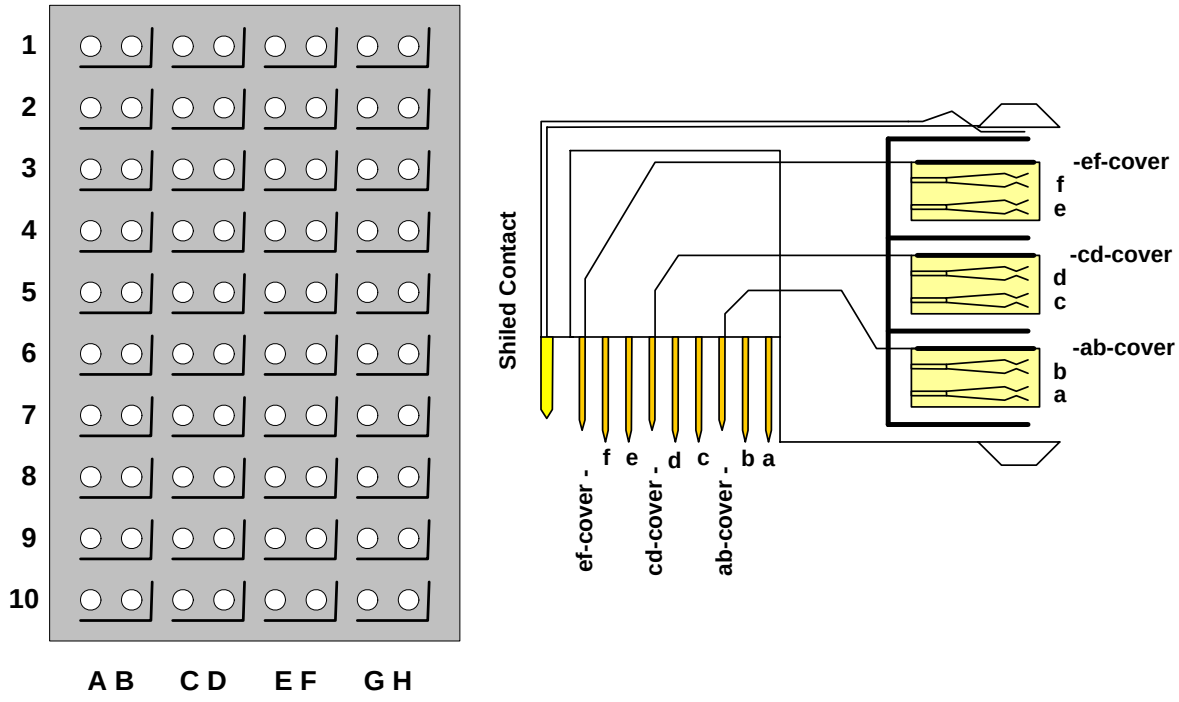


Figure 24 shows how base and fabric channels map to ZD pairs on the ZD connector.

Figure 24. Zone 2 data transport connector base channel and fabric channel port allocation

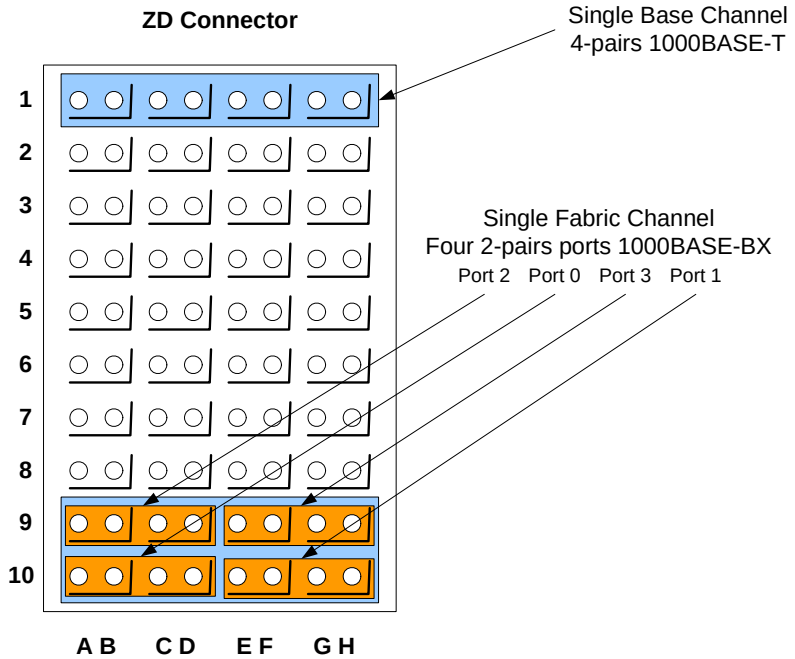


Figure 25 shows active positions on data transport connectors. Data transport connectors J20, J21, J22, J23, and J24 are Differential Signaling (ZD) connectors each with 40 link pairs of ZD (differential) signals. Table 25 and Table 26 give the pin assignments for the J20 and J23 connectors respectively (see Figure 25).

Note: The J21 and J22 connectors that provide the connections for fabric interface channels 3 to 13 in a full-mesh configuration are not used.

Figure 25. Zone 2 data transport connector active positions

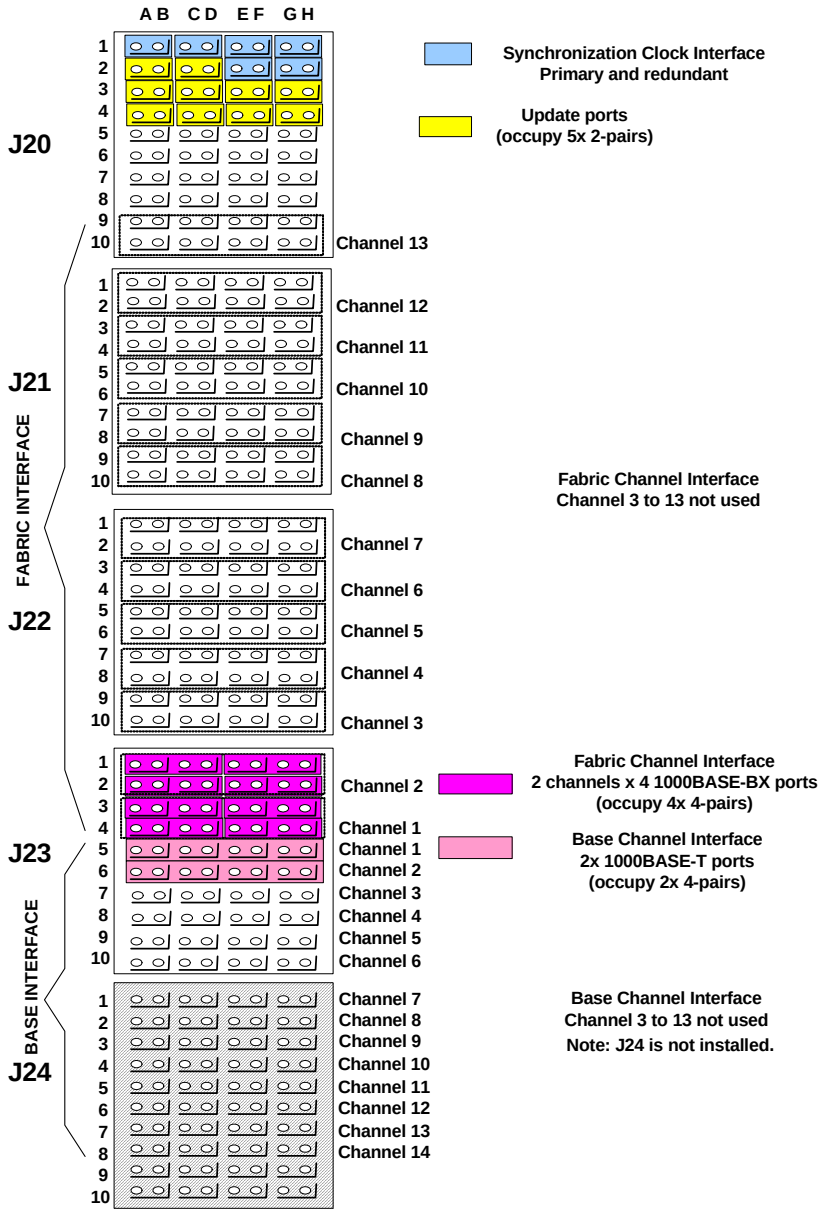


Table 25. Zone 2 data transport connector J20 pin assignments

Row	Name	A	B	C	D	E	F	G	H
1	Clocks	CLK 1A+	CLK 1A-	CLK 1B+	CLK 1B-	CLK 2A+	CLK 2A-	CLK 2B+	CLK 2B-
2	Update Port & Clks	Tx4(UP+)	Tx4(UP-)	Rx4(UP+)	Rx4(UP-)	CLK 3A+	CLK 3A-	CLK 3B+	CLK 3B-
3		Tx2(UP+)	Tx2(UP-)	Rx2(UP+)	Rx2(UP-)	Tx3(UP+)	Tx3(UP-)	Rx3(UP+)	Rx3(UP-)
4		Tx0(UP+)	Tx0(UP-)	Rx0(UP+)	Rx0(UP-)	Tx1(UP+)	Tx1(UP-)	Rx1(UP+)	Rx1(UP-)
5	Fabric Channel 15	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
6		Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
7	Fabric Channel 14	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
8		Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
9	Fabric Channel 13	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
10		Tx0[13]+	Tx0[13]-	Rx0[13]+	Unused	Unused	Unused	Unused	Unused
Pin Assignment Format: Tx/Rx I[J]K where, Tx = transmit, Rx = receive, I = differential pair number (0 to 3), J = channel designator (1 to 15) and K = polarity (+ or -).									

Table 26. Zone 2 data transport connector J23 pin assignments

Row	Name	A	B	C	D	E	F	G	H
1	Fabric Channel 2	Tx2[2]+	Tx2[2]+-	Rx2[2]+	Rx2[2]-	Tx3[2]+	Tx3[2]+-	Rx3[2]+	Rx3[2]-
2		Tx0[2]+	Tx0[2]-	Rx0[2]+	Rx0[2]-	Tx1[2]+	Tx1[2]-	Rx1[2]+	Rx1[2]-
3	Fabric Channel 1	Tx2[1]+	Tx2[1]+-	Rx2[1]+	Rx2[1]-	Tx3[1]+	Tx3[1]+-	Rx3[1]+	Rx3[1]-
4		Tx0[1]+	Tx0[1]-	Rx0[1]+	Rx0[1]-	Tx1[1]+	Tx1[1]-	Rx1[1]+	Rx1[1]-
5	Base Channel 1	BI_DA1+	BI_DA1-	BI_DB1+	BI_DB1-	BI_DC1+	BI_DC1-	BI_DD1+	BI_DD1-
6	Base Channel 2	BI_DA2+	BI_DA2-	BI_DB2+	BI_DB2-	BI_DC2+	BI_DC2-	BI_DD2+	BI_DD2-
7	Base Channel 3	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
8	Base Channel 4	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
9	Base Channel 5	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
10	Base Channel 6	Unused	Unused	Unused	Unused	Unused	Unused	Unused	Unused
Pin Assignment Format: Tx/Rx I[J]K where, Tx = transmit, Rx = receive, I = differential pair number (0 to 3), J = channel designator (1 to 15) and K = polarity (+ or -).									

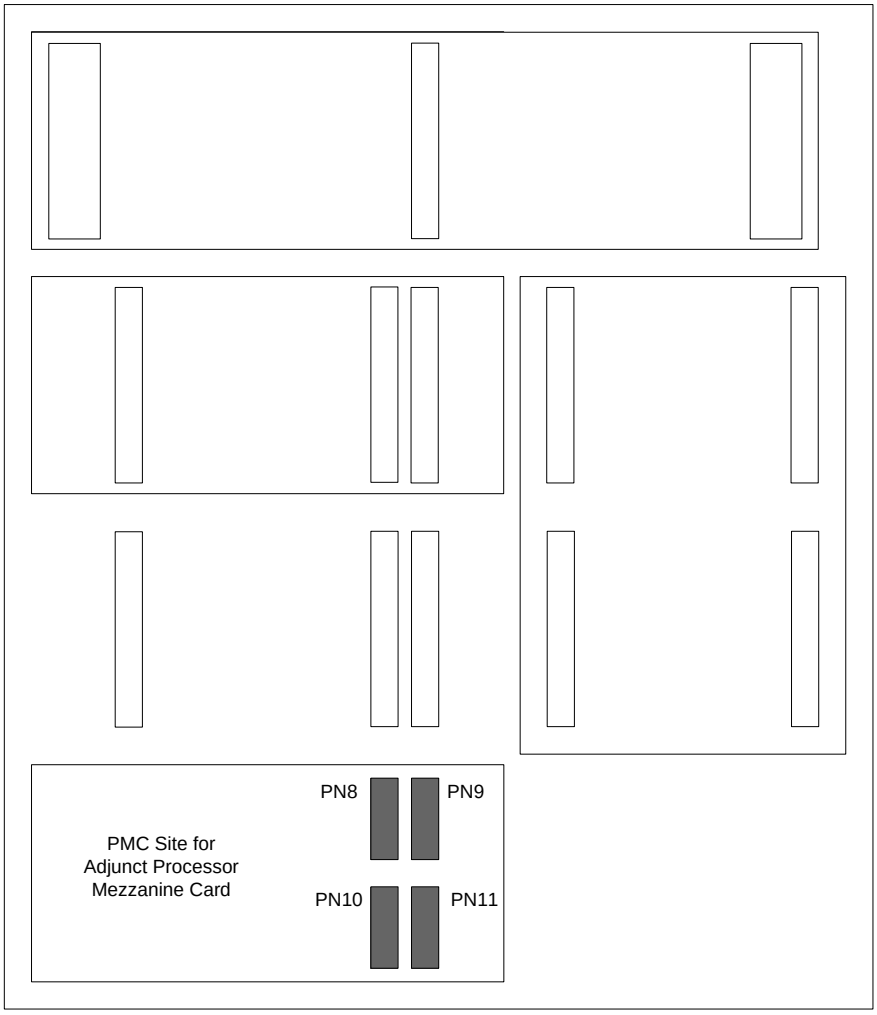
5.2.3 Zone 3: RTM (Not Used)

A Rear Transition Module (RTM) connector. Not used on IXB2850 boards.

5.3 PrPMC Site Interface Connectors

IXB2850 baseboards contains provision for a Processor PCI Mezzanine Card (PMC) that can be added to function for example as an Adjunct Processor (AP).

Figure 26. PrPMC site and interface connector locations on the baseboard



The physical connection between the PMC and the baseboard is made through AMP connectors, compliant with the IEEE P1386.1 PMC standard. Three 64-pin connectors (PN8, PN9, and PN10) are required to provide a 192-pin connection between the two boards (64-bit PCI Bus). The PN11 connector is used for user-defined I/O signals.

Table 27 gives pinout information for the PN8 connector.

Table 27. PMC interface connector PN8 pin assignments

Pin#	Description	Remarks	Pin#	Description	Remarks
1	TCK	JTAG	2	-12V	
3	GND		4	INTA#	
5	INTB#		6	INTC#	
7	PRESENT#		8	+5V	
9	INTD#		10	PCI-RSVD	
11	GND		12	3.3Vaux	
13	PCICLK		14	GND	
15	GND		16	GNT#	
17	REQ#		18	+5V	
19	V(I/O)3.3V		20	AD[31]	
21	AD[28]		22	AD[27]	
23	AD[25]		24	GND	
25	GND		26	PCI_CBE#3	
27	AD[22]		28	AD[21]	
29	AD[19]		30	+5V	
31	V(I/O)3.3V		32	AD[17]	
33	FRAME#		34	GND	
35	GND		36	IRDY#	
37	DEVSEL#		38	+5V	
39	GND		40	LOCK#	
41	PCI-RSVD		42	PCI_RSVD	
43	PAR		44	GND	
45	V(I/O)3.3V		46	AD[15]	
47	AD[12]		48	AD[11]	
49	AD[09]		50	+5V	
51	GND		52	PCI_CBE#0	
53	AD[06]		54	AD[05]	
55	AD[04]		56	GND	
57	V(I/O)		58	AD[03]	
59	AD[02]		60	AD[01]	
61	AD[00]		62	+5V	
62	GND		64	REQ64#	

Table 28 gives pinout information for the PN9 connector.

Table 28. PMC interface connector PN9 pin assignments

Pin#	Description	Remarks	Pin#	Description	Remarks
1	+12V		2	TRST#	JTAG
3	TMS	JTAG	4	TDO	JTAG
5	TDI	JTAG	6	GND	
7	GND		8	PCI-RSVD	
9	PCI_RSVD		10	PCI-RSVD	
11	PUP		12	3.3V	
13	RST#		14	PDN1	
15	3.3V		16	PDN2BUSMODE#4	
17	PME#		18	GND	
19	AD[30]		20	AD[29]	
21	GND		22	AD[26]	
23	AD[24]		24	3.3V	
25	IDSEL		26	AD[23]	
27	3.3V		28	AD[20]	
29	AD[18]		30	GND	
31	AD[16]		32	PCI_CBE#2	
33	GND		34	IDSELB	
35	TRDY#		36	3.3V	
37	GND		38	STOP#	
39	PERR#		40	GND	
41	3.3V		42	SERR#	
43	CBE#1		44	GND	
45	AD[14]		46	AD[13]	
47	M66EN		48	AD[10]	
49	AD[08]		50	3.3V	
51	AD[07]		52	PCI_RSVD	
53	3.3V		54	PCI-RSVD	
55	PCI-RSVD		56	GND	
57	PCI-RSVD		58	EREADY	
59	GND		60	RESETOUT#	
61	ACK64#		62	3.3V	
62	GND		64	MONARCH#	

Table 29 gives pinout information for the PN10 connector.

Table 29. PMC interface connector PN10 pin assignments

Pin#	Description	Remarks	Pin#	Description	Remarks
1	PCI-RSVD		2	GND	
3	GND		4	C/BE[7]	
5	C/BE[6]#		6	C/BE[5]#	
7	C/BE[4]#		8	GND	
9	V (I/O)		10	PAR64	
11	AD[63]		12	AD[62]	
13	AD[61]		14	GND	
15	GND		16	AD[60]	
17	AD[59]		18	AD[58]	
19	AD[57]		20	GND	
21	V (I/O)		22	AD[56]	
23	AD[55]		24	AD[54]	
25	AD[53]		26	GND	
27	GND		28	AD[52]	
29	AD[51]		30	AD[50]	
31	AD[49]		32	GND	
33	GND		34	AD[48]	
35	AD[47]		36	AD[46]	
37	AD[45]		38	GND	
39	V (I/O)		40	AD[44]	
41	AD[43]		42	AD[42]	
43	AD[41]		44	GND	
45	GND		46	AD[40]	
47	AD[39]		48	AD[38]	
49	AD[37]		50	GND	
51	GND		52	AD[36]	
53	AD[35]		54	AD[34]	
55	AD[33]		56	GND	
57	V (I/O)		58	AD[32]	
59	PCI-RSVD		60	PCI-RSVD	
61	PCI-RSVD		62	GND	
62	GND		64	PCI-RSVD	

Table 30 gives pinout information for the PN11 connector.

Table 30. PMC interface connector PN11 pin assignments

Pin#	Description	Remarks	Pin#	Description	Remarks
1	PMC_CHA_MDI0+	1000BASE-T – port 0	2	PMC_CHA_MDI2+	1000BASE-T – port 0
3	PMC_CHA_MDI0-		4	PMC_CHA_MDI2+	
5	GND		6	GND	
7	PMC_CHA_MDI1+	1000BASE-T – port 0	8	PMC_CHA_MDI3+	1000BASE-T – port 0
9	PMC_CHA_MDI1+		10	PMC_CHA_MDI3+	
11	GND		12	GND	
13	PMC_CHB_MDI0+	1000BASE-T – port 1	14	PMC_CHB_MDI2+	1000BASE-T – port 1
15	PMC_CHB_MDI0+		16	PMC_CHB_MDI2+	
17	GND		18	GND	
19	PMC_CHB_MDI1+	1000BASE-T – port 1	20	PMC_CHA_MDI3+	1000BASE-T – port 1
21	PMC_CHB_MDI1+		22	PMC_CHA_MDI3+	
23	GND		24	GND	
25			26		
27			28		
29			30		
31			32	SCL_I2C_FIC_LOCAL	PMC I^2C
33	+3.3V SH	+3.3V (early power) EPC-6321B only	34	SDA_I2C_FIC_LOCAL	PCM I^2C
35			36		
37			38		
39			40		
41			42	TXD	UART
43	RXD	UART	44	RTS	UART
45	CTS	UART	46		
47	+5V	+5V	48		
49			50		
51			52		
53	Test point		54		
55			56		
57			58		
59			60		
61			62	+3.3V (early power)	
63			64		

6.0 Installation and Configuration

6.1 Operating Environment

IXB2850 boards are designed to operate in a chassis that conforms to the AdvancedTCA* (PICMG* 3.x) standards. The chassis consists of the following components:

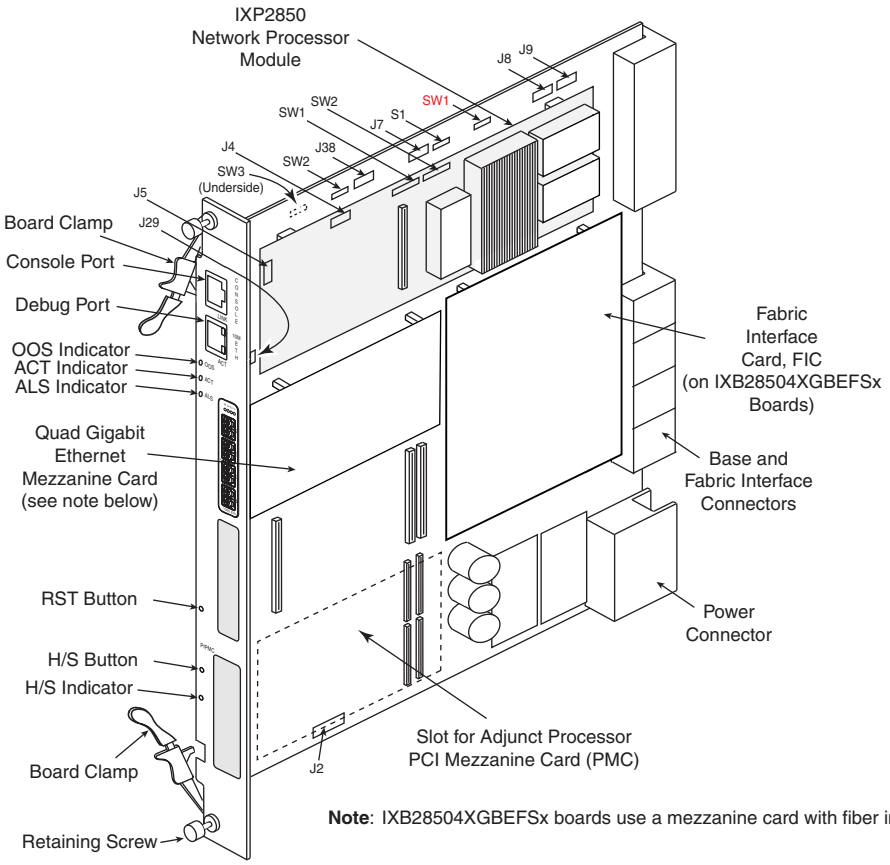
- Two redundant Shelf Management Controllers (ShMCs)
- Two redundant Switch Fabrics boards
- Backplane, which includes the following interfaces within the confines of a slot designed for an IXB2850 board:
  - **Intelligent Platform Management Bus** (IPMB)
    A dual I^2C bus connecting the ShMCs with each board in the chassis.
  - **Base Interface** (connecting each board with both Switch Fabric boards)
    There is a separate channel per connection with a particular Switch Fabric. Each channel constitutes a single Ethernet port (1000BASE-T). This interface is dedicated for control traffic in the PICMG 3.1 configuration and is shared by both control and data traffic in the PICMG 3.0 configuration.
  - **Fabric Interface** (connecting each board with both Switch Fabric boards)
    There is a separate channel per connection with a particular Switch Fabric. Each channel constitutes four Gigabit Ethernet (fiber) ports (1000BASE-CX). This interface is dedicated to data traffic in the PICMG 3.1 configuration.
  - **Clock Interface**
    A source of the reference clock for the IXB2850 board
  - **Update Interface**
    A five channel interface

Note: IXB2850 boards can operate in a chassis that is not compliant with AdvancedTCA, where the ShMC is not present. To make this possible, there is a dedicated hardware jumper on the baseboard that defines whether the BMC should communicate with the ShMC. The chassis does not need to be equipped with Ethernet switches for backplane connections. It is possible to use point-to-point Ethernet connections between two boards. IXB2850 boards do not need to know whether the backplane is switched or not.

6.2 Hardware Configuration

IXB2850 boards include a number of hardware-configurable components such as jumpers and switches. See Figure 27 for locations. These jumpers and switches are preset during manufacturing and generally should **not** need to be changed. When upgrading firmware, specifically when upgrading FRU Information, SDRs and CPLDs, specific jumpers and switches need to be set; see Section 11.3.1, “Local Software Upgrade Procedure” on page 137 and its subsections for more information. The following subsections describe the default settings of the jumpers and switches so that they can be reset to default settings if necessary.

Figure 27. IXB2850 jumper and switch locations



Note: IXB28504XGBEFSx boards use a mezzanine card with fiber interfaces as shown.

NP Module Jumpers and Switches

The following list identifies the function of various connectors and the default settings of jumpers and switches on the Network Processor module:

- **J5**: NetROM connector for NP module
- **J4**: Default settings are:
  - Pins 1,2: ON
  - Pins 3,4: ON
  - Pins 5,6: ON
- **SW1**: Default settings are:
  - 1: OFF
  - 2: OFF
  - 3: OFF
  - 4: ON
  - 5: OFF
  - 6: OFF
  - 7: OFF
  - 8: OFF

  - 9: OFF
  - 0: OFF
- **SW2**: Default settings are:
  - 1: ON
  - 2: OFF
  - 3: ON
  - 4: OFF
  - 5: ON
  - 6: OFF
  - 7: OFF
  - 8: OFF
  - 9: ON
  - 0: ON

Baseboard Jumpers and Switches

The following list identifies the default settings of jumpers and switches on the baseboard:

- **J29**: Default settings are:
  - Pins 1,2: ON
  - Pins 3,4: ON
  - All others: OFF
- **SW3** (underside of board): Default settings are:
  - 1: ON
  - 2: OFF
  - 3: ON
  - 4: OFF
  - 5: ON
  - 6: ON
  - 7: ON
  - 8: ON
- **SW2**: Default settings are:
  - 1: OFF
  - 2: OFF
  - 3: ON
  - 4: OFF
  - 5: OFF
  - 6: OFF
  - 7: ON
  - 8: ON

- **J38**: Default settings are:
  - 1: ON
  - 2: ON
  - 3: ON
  - 4: OFF
  - 5: OFF
  - 6: ON
  - 7: ON
  - 8: OFF
- **J7**: Default settings are:
  - 1: ON
  - 2: ON
  - 3: ON
  - 4: ON
  - 5: ON
  - 6: ON
  - 7: ON
  - 8: OFF
- **S1**: Default settings are:
  - 1 to 8: ON
- **SW1**: Determines which UART is active for the serial console port on the front panel. See Section 5.1.1, “COM Port” on page 60 for details. Default settings are:
  - 1: OFF
  - 2: ON
  - 3: ON
  - 4: ON
- **J8**: General purpose connector for connection to Lattice JTAG programmer, Corelis* or Multi-ICE* for the Board Management Controller (BMC) and Macraigor* for the Network Processor (NP).
- **J9**: Lattice JTAG connector (for programming of JTAG router)
- **J2**: Default settings are:
  - Pins 1,2: ON
  - Pins 7,8: ON
  - Pins 15,17: ON
  - All others: OFF

Note: If you are installing a shelf manager in your AdvancedTCA chassis, J2 pins 1-2 must be set ON. If you are not installing a shelf manager, J2 pins 1-2 must be set OFF.

6.3 Preparing for Installation

It is important to become familiar with the safety aspects and other essential or national requirements. Before installing the board, read the following chapters in this manual:

- Chapter 18.0, “Safety Warnings”
- Chapter 16.0, “Certifications”
- Chapter 17.0, “Agency Information”

6.3.1 Protecting the Board From Damage

Caution: All computer boards are sensitive to electrostatic discharge. Handle all static-sensitive boards and components at a static-safe work area, and observe anti-static precautions at all times.

6.3.2 Unpacking the Product

Caution: Do not remove the board from the antistatic packaging until you are ready to install it.

Observe proper anti-static precautions at all times. Inspect the packaging for any signs of damage that may have occurred during transit. In the event of damage or missing items, notify both the carrier and the supplier immediately.

6.4 Installing the Board into a Chassis

Note: The AdvancedTCA standard supports “hot insertion” and “hot swap” of IXB2850 boards. The chassis can either be powered on or off when inserting the board. The following procedure describes how to install the board in a live, “hot” system. Installation in a “cold” system is similar but without the LED indications.

Install the main board in the chassis as follows:

1. Ensure that the board clamps are in the open (unlocked) position (away from the front panel).
2. Use the chassis guide rails to carefully align the board’s backplane connectors with the AdvancedTCA chassis backplane.
3. Gently insert the board into the chassis backplane.
4. Lock the board clamps.
5. Observe that the blue H/S indicator lights, then after a few seconds it goes out and the green HEALTH indicator is lit indicating that the board is now ready for operation.
6. Press in and tighten both of the board’s front panel retaining screws. When the screws are tightened, the board should be firmly connected to the chassis backplane.

6.5 Removing the Board from a Chassis

Remove the board from the chassis as follows:

1. Loosen both of the board’s front panel retaining screws.

Caution: Do not remove the board until the H/S indicator is lit continuously (that is, not flashing).

2. Open both of the board's clamps slightly to initiate removal of the board. Do not remove the board from the chassis backplane. The green HEALTH indicator goes out and after a few seconds the blue H/S indicator is lit.
3. Once the blue H/S indicator is lit, completely open the board clamps to disconnect the board from the chassis backplane slot.
4. Remove the board from the chassis.

6.6 Software Installation

IXB2850 boards are delivered without any software preinstalled (only firmware is preinstalled). It is the customer's responsibility to prepare the Linux kernel, file system and target application. This document provides information on downloading software to an IXB2850 board (that is, SSU functionality) and running software from a remote server. See Section 11.2, "Safe System Upgrade" on page 131 for more information.

6.7 Boot Monitor Operation

See Section 9.4, "Boot Monitor" on page 110 for a detailed description of the Boot Monitor (RedBoot*).

6.8 Power-On Self Test

The Board Management Controller (BMC) and the Network Processor (NP) both perform Power On Self Test (POST). See Section 8.1.2.1, "BMC Power On Self Test" on page 86 and Section 9.4.3, "NPU Power On Self Test" on page 112 for more information.

6.9 Diagnostics

The diagnostics are handled by a separate application (outside of the BootROM). By default, diagnostics are executed under the control of a diagnostic command line interface (DIAG> prompt). However, the diagnostics can be executed from the Boot Monitor. This is done by loading the diagnostics image into SDRAM from flash memory, TFTP server (Ethernet debug port), or through a serial (UART) port.

6.9.1 Running the Diagnostics

To run the diagnostics for the board, perform the following:

1. Enter the following command to ensure that you are using the proper file names for the diagnostic image:

```
RedBoot> fis list
```

2. Load the diagnostics image, as it appears in the fis list command output, from flash:

```
RedBoot> fis load -s diag
```

Note: If the "Cannot find valid SSU header in 0x01000000-0x012e0000" message is displayed, then either the diag image does not contain an SSU header or it is invalid. Verify that the image is valid (has the correct length, etc.), then try again.

Note: The diag diagnostics image name is shown for example purposes only. The diagnostics image for your IXB2850 board may have a different name. Ensure that you use the exact image name shown in the fis list command output (step 1 above).

3. Execute a valid diagnostics image as follows:

```
RedBoot> cond go
```

4. After the diagnostics executes, the diagnostics prompt appears:

```
DIAG>
```

5. Use any of the diagnostics commands as described in Section 6.9.2.
6. Exit the diagnostic image:

```
DIAG> exit
```

6.9.2 Diagnostics Commands

Table 31 gives a list of the diagnostics commands that are available for use at the diagnostics prompt (DIAG>). For details on each command, see Appendix C, "Diagnostics".

Table 31. IXB2850 diagnostics commands

Command	Description
banner	Print a hardware/software information banner.
blockcopy	Copy a memory block.
c	Change the contents of a memory address.
cfg	Read/write data from/to a PCI device.
doff	Disable data cache.
don	Create page table, enable data cache and enable wb.
exit	Exit the diagnostics image.
fill	Fill a block of memory with a specified value.
help	Print the help screen for a diagnostic command.
i2cread	Read from an I^2C device and display results.
i2cwrite	Write to an I^2C device.
io	Read/write data from/to a PCI IO.
ioff	Disable instruction cache.
ion	Enable data instruction cache.
mem	Read/write from/to memory.
p	Display contents of memory.
sethoe	Set the halt on error value.
setverbose	Set the verbose output level for all diagnostic commands.
test	Display a list of test commands within the diagnostic image.
tf	Enable test flags.
t bmc	Perform Board Management Controller tests.
t ethslow	Perform Ethernet slow port tests.
t generate	Starts the traffic generator for all ports configured by the t media command
t gpio	Perform GPIO tests.
t i2c	Perform I^2C bus tests.
t int	Perform interrupt controller tests.

Table 31. IXB2850 diagnostics commands (Continued)

Command	Description
t led	Perform LED tests.
t media	Perform Intel® IXF1104 and PHY interface test.
t mem	Perform memory tests.
t msf	Perform MSF tests.
t pci	Perform PCI tests.
t slowport	Perform slow port tests.
t uart	Perform UART tests.
t ueng	Perform a microengine test.
t xscale	Perform Intel XScale® scratchpad and core component tests.
wrv	Write, read and verify content in a specified memory address.

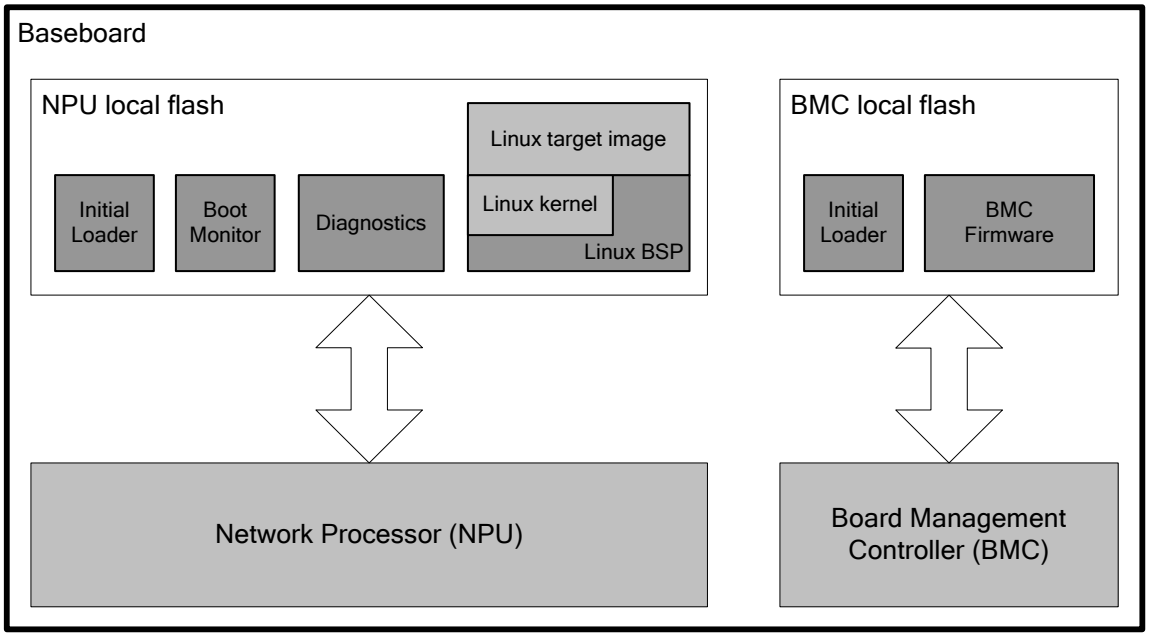
7.0 Firmware Overview

7.1 Firmware Components

IXB2850 boards are supplied with firmware for the Board Management Controller (BMC) and the Network Processor (NP).

Figure 28 shows the various firmware components and where they reside.

Figure 28. Firmware components

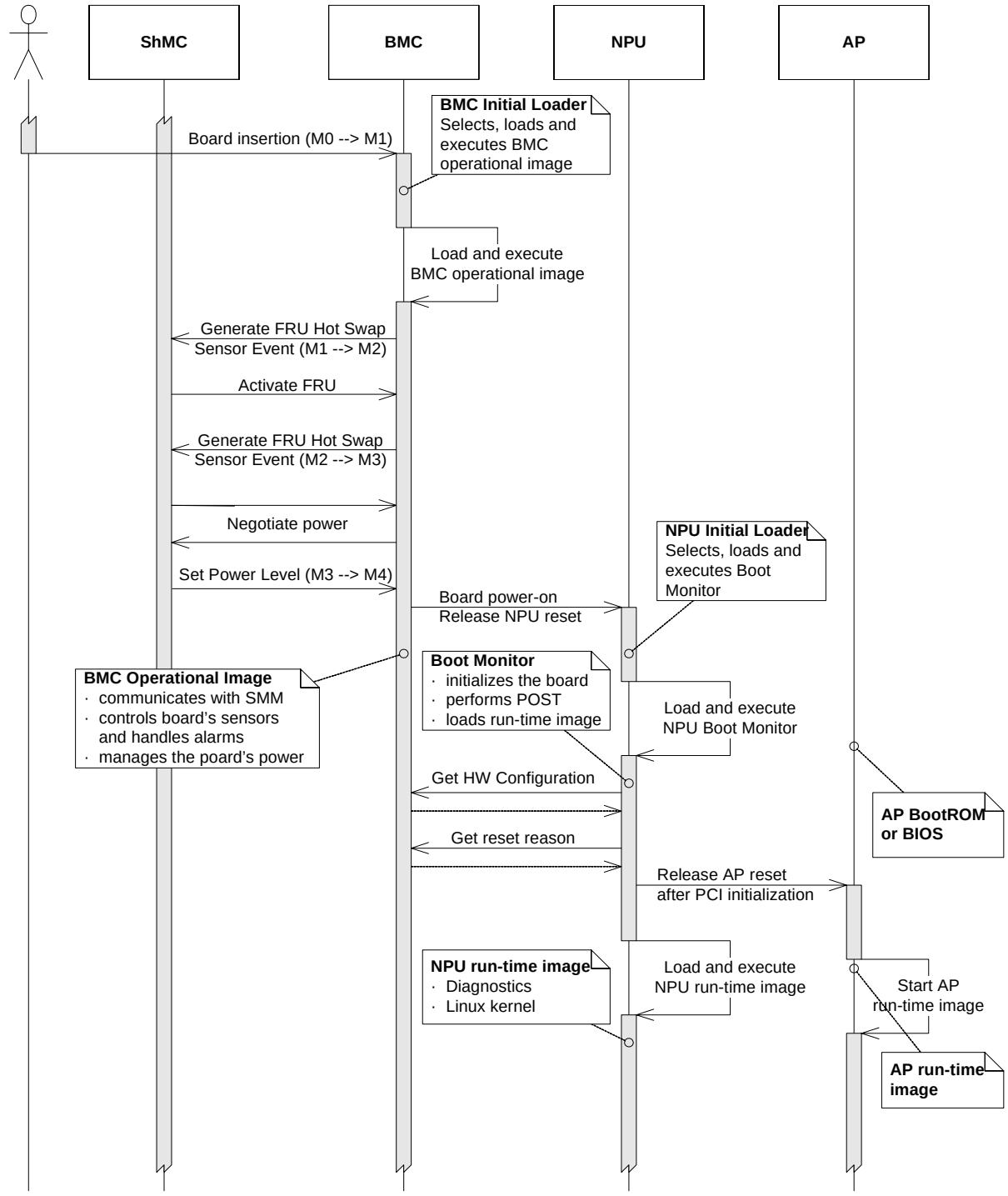


7.2 Boot Sequence

The IXB2850 boot sequence is shown in Figure 29. The first components that start operating after board power on are the Board Management Controller (BMC) and early power I^2C devices (ID EEPROMs and sensors) on the baseboard and associated extension boards, such as the Quad Gigabit Ethernet Mezzanine Card and the Fabric Interface Card (FIC). The BMC establishes communication with the chassis's Shelf Management Controllers (ShMCs) to provide each ShMC with information about the type and configuration of the board. The ShMC decides whether this board can operate in a particular slot of chassis and allow the BMC to switch on the power for the remaining components of the board. When power is switched on, the BMC releases an NPU reset signal.

The NPU starts execution of the Initial Loader code and controls the reset signals for all other components on the board.

Figure 29. IXB2850 boot sequence



When a PMC card, such as an Adjunct Processor card is detected, the NPU deasserts the AP reset signal after PCI bus initialization. The AP starts execution of boot ROM (or BIOS) code, while the boot ROM software on the NPU initializes execution of operating software (diagnostics or Linux kernel with target image). When AP Power On Self Test (POST) is completed, the AP initializes execution of its own run-time image.

Each processor runs its own specific software image independently. There is no built-in operating system synchronization mechanism for bootstrapping the two processors or system runtime support, that is, the operating system on each processor does not synchronize with the other's control. This design assumes that the application running on the system is to take full responsibility for any synchronizing system activities. The application can build its own synchronization using a shared-memory facility provided by the operating system (that is, the memory of the other processor can be accessed through the PCI bus; the range of shared memory depends on the PCI configuration).

7.3 Board Management Controller Firmware Overview

The Board Management Controller (BMC) is the first board component to start operating when power is provided to the board. The BMC is responsible for the following:

- **Access to I^2C devices**
  The BMC accesses ID EEPROMs and voltage/temperature sensors installed on the baseboard and associated cards. It can read and write ID EEPROM contents.
- **IPMI support for communication with the Shelf Management Controller (ShMC)**
  The ShMC obtains from the BMC information about the type of the board and all extension cards as well as information about sensors installed on the board (such as the sensor identifier, type, acceptable value range, and currently reported value).
- **Power Management and Hot Swap support**
- **IPMI support for board CPUs (NPU and AP)**
  The NPU and AP can communicate with the BMC to obtain the FRU Information as well as the board sensor readings and chassis slot number.

The BMC firmware resides in the BMC local flash memory. See Chapter 8.0, “Board Management Controller Firmware” for details on the organization and content of the BMC local flash memory.

7.4 Network Processor Firmware Overview

The Network Processor (NP) firmware comprises:

- Initial Loader
- Boot Monitor
- Diagnostics
- Linux* Support Package (LSP)

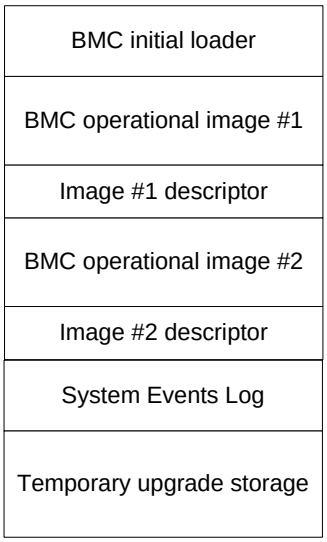
The NPU firmware resides in onboard flash memory. See Chapter 9.0, “Network Processor Firmware” for details on the organization and content of the NPU local flash memory.

8.0 Board Management Controller Firmware

8.1 BMC Flash Memory Layout and Content

On IXB2850 boards, the BMC firmware resides in the BMC local flash memory. The organization of the flash memory is shown in Figure 30. The content of the various sectors are described in the subsections that follow.

Figure 30. BMC flash memory organization



8.1.1 BMC Initial Loader

The first sector (128kB) of the BMC's local flash is occupied by the BMC Initial Loader. This is the first code executed by the BMC just after power is applied to the board. The BMC Initial Loader is responsible for:

- Basic BMC initialization (EEPROM, flash, RAM)
- Selecting, loading and executing one of the BMC operational images

The BMC Initial Loader provides additional functionality for BMC image recovery. It allows the loading of a new BMC image (using the Xmodem protocol) and the selection of an operational image.

The functionality enables the updating of BMC images in the event that the software does not operate correctly (that is, if it is not possible to load a new BMC image from the BMC console or to select the alternate operational image).

The user can enter the BMC recovery boot loader by entering the three characters, **==2,** during the first four seconds of board startup.

Note: The BMC recovery boot loader contains several options that are password protected. These options are **not** intended for customer use because their use can invalidate the software in a way that requires additional hardware for board recovery.

8.1.2 BMC Operational Images

The BMC's local flash contains two BMC operational images. The BMC allocates three sectors of flash memory for each operational image and its descriptor. One of these images is loaded by the BMC Initial Loader from flash to RAM and then executed. The selection of a proper operational image is done according to the Status Word stored in the BMC EEPROM memory.

Section 11.5, "Remote BMC Firmware Upgrade" on page 158 describes BMC dual image functionality and the format of the BMC status word.

Each of the two BMC operational images has an accompanying "properties image" descriptor containing image control information that needs to be verified before the image is stored on flash. See Section 11.5, "Remote BMC Firmware Upgrade" on page 158 for details.

8.1.2.1 BMC Power On Self Test

The Power-On Self Test (POST) performs basic tests on the BMC and the main baseboard components that the BMC depends on. These tests do not require any cables to be connected.

BMC POST can be executed as a set of short or long tests:

- The long POST set contains all possible tests the BMC can perform.
- The short POST executes only selected tests depending of the BMC's short POST configuration register included in the baseboard FRU Information. See Table 15, "BMC boot parameters definition" on page 50.

Note: The short POST option provides the ability to shorten BMC boot time.

The BMC can execute the following tests within the confines of POST:

- IRQ
- GPIO
- Flash
- Internal RAM
- External RAM
- IPMB
- local I^2C buses
- embedded UARTs
- external quad UART
- Keyboard Controller Style (KCS)
- LEDs
- Timers

Interrupt (IRQ) POST

Interrupt controller verification testing is available, including simulated interrupts.

General Purpose I/O (GPIO) POST

This POST verifies the subset of processor GPIO lines. Only a subset is chooses because the test is designed not to disrupt the normal operation of the board (for example, GPIO lines controlling power to the rest of the system are excluded).

Memory POST

The following memory types are installed and should be tested:

- Internal BMC RAM
- External RAM

The following tests are executed:

- Walking ones
- Walking zeros
- Known pattern tests:
  - 0x5a5a5a5a pattern
  - 0xa5a5a5a5 pattern
- Address bus test
- Incremental test

I^2C POST

The following I^2C buses should be tested:

- IPMB I^2C buses
- Local I^2C buses

The following I^2C tests are executed:

- Current read
- Random read
- Sequential read

In all other cases, only a simple test for detecting bus short circuit is executed.

UART POST

UART POST covers the initialization and test of the serial ports. There are two internal UARTs used as consoles and for communication with the NPU. The BMC also handles an external quad UART that is used for communication with processors installed on extension cards, for example Media Mezzanine Cards, FIC and PMC Adjunct Processor card. Each serial line register is written and then read for verification. An internal loopback test is also performed.

The following tests are executed using the UARTs in loopback mode:

- Register access
- Non-FIFO polling
- Non-FIFO interrupt
- FIFO polling
- FIFO interrupt

LED POST

LED POST performs simple LED tests. Each LED is lit in all the supported colors (for a noticeable amount of time).

KCS POST

At the stage POST is being executed, KCS peripherals are not yet powered. Only simple register access tests are performed.

Timer Test

There are four timers supported in the BMC. The following test is performed:

- Load the timer registers with a certain value and verify that an interrupt is generated after the time elapses.

8.1.3 System Event Log

The System Event Log (SEL) is a block of four sectors on the BMC local flash allocated to store IPMI events. Event handling, storage, and retrieval procedures are described in the following sections. Currently, the SEL is 128kB long and it is not circular (new events do not overwrite old ones, and are discarded if the SEL is full). The BMC SEL architecture is consistent with the definition in the IPMI 1.5 specification.

8.1.3.1 SEL Descriptor

The SEL Descriptor occupies the first 13 bytes of SEL storage memory and has the format given in Table 32. Information included in the SEL Descriptor is used to prepare a response to the **Get SEL Info** command.

Table 32. BMC SEL descriptor

Byte	Description
1:2	Number of log entries in the SEL, LS byte first
3:4	Free space in bytes, LS Byte first. FFFFh indicates that 65,535 or more bytes of free space are available.
5:8	Most recent addition timestamp†. LS byte first.
9:12	Most recent erase timestamp. The last time that one or more entries were deleted from the log; LS byte first.
13	Operation Support 7:7 – Overflow Flag. 1b = Events have been dropped due to lack of space in the SEL. 6:0 – Reserved. Write as 0b
† Timestamps used by SEL functionality have a format that conforms to the definition described in the IPMI 1.5 specification.	

8.1.3.2 Event Logging

The BMC stores the following events in the event log:

- All IPMI events generated by the BMC
- All IMPI events generated by the NPU (or AP) and forwarded by the BMC
- All reset events (entire board, NPU, or BMC)

IPMI events are stored in the SEL using the record format given in Table 33 (compare to Section 26 in the IPMI 1.5 specification).

Table 33. SEL record format used for IPMI events

Byte	Field	Description
1:2	Record ID	ID used for SEL Record access. The Record ID values 0000h and FFFFh have special meaning in the Event Access commands and must not be used as Record ID values for stored SEL Event Records.
3	Record Type	Record Type = System Event Record (02h)
4:7	Timestamp	Time when event was logged; LS byte first
8:9	Generator ID	Event Generator ID: Byte 1 • 7:1 – 7-bit I^2C BMC Slave Address • 0:0 – 0b - the ID of the IPMB Slave Address Byte 2 • 7:4 – 0h - Channel number • 3:2 – Reserved. Write as 00b • 1:0 – BMC LUN
10	EvM Re	Event Message format version = 04h
11	Sensor Type	Sensor Type Code taken from an event
12	Sensor #	Sensor number taken from an event
13	Event Dir | Event Type	Event Dir and Event Type taken from an event
14	Event Data 1	Data taken from an event Data 1 field
15	Event Data 2	Data taken from an event Data 2 field
16	Event Data 3	Data taken from an event Data 3 field

Reset events are stored in SEL using the record format given in Table 34 (compare to Section 26 in the IPMI 1.5 specification).

Table 34. SEL record format used for Reset events

Byte	Field	Description
1:2	Record ID	ID used for SEL Record access. The Record ID values 0000h and FFFFh have special meaning in the Event Access commands and must not be used as the Record ID values for stored SEL Event Records.
3	Record Type	Record Type = OEM System Event Record (C0h)
4:7	Timestamp	Time when event was logged; LS byte first
8 10	Manufacturer ID	Manufacturer ID = 5A3100h
11	Event Type	Event Type = Reset (00h)

Table 34. SEL record format used for Reset events

Byte	Field	Description
12	Reset Type	Reset Type: • 00h – Entire board reset • 01h – BMC reset • 02h – NPU reset
13	Reset Reason	Reset Reason defined as follows: Reset Type = Entire Board reset: • 00h – Operator initiated board power-down • 01h – BMC initiated board power-down performed under ShMC control • 02h – BMC controlled board power-down. Reset Info byte contains number of sensor that was a reason of the particular reset Reset Type = NPU reset: • 00h – Operator initiated NPU reset • 01h – NPU reset by BMC • 02h – NPU watchdog reset Reset Type = BMC reset: • 00h – Operator initiated BMC reset
14	Reset Info	Additional reset information

8.1.3.3 Timestamps

One of the internal BMC timers is used as a clock. Because this kind of clock does not maintain the its value during a board power-down period, at each startup, the BMC issues a **SEL Get Time** command to get the time from the ShMC and set the internal BMC timer. This ensures that the BMC can set the correct event time-stamps.

The timestamp format used by the BMC conforms to the definition described in the IPMI 1.5 specification.

8.1.4 Temporary Upgrade Storage

Temporary upgrade storage on the BMC local flash is a block of eight flash sectors used as temporary storage for image files used during the CPLD and FRU upgrade process.

8.2 IPMI Protocol Support

The BMC fully supports the IPMI 1.5 rev. 1.1. This protocol is used for communication with the ShMC over the IPMB. See Section 8.2.1, “IPMI 1.5 Command Support” on page 90 for more information.

The BMC also supports the PICMG 3.0 OEM mandatory command set. See Section 8.2.2, “PICMG 3.0 IPMI Command Support” on page 95 for details on the level of support.

In addition, the BMC supports a set of Intel OEM commands and board-specific commands. See Appendix D, “OEM IPMI Commands” for details.

8.2.1 IPMI 1.5 Command Support

Table 35 lists the commands defined in the IPMI 1.5 specification. Each command is grouped by general function, with an indication of whether the command is mandatory (M) or optional (O) from the PICMG point-of-view. There is also information on whether a given command is supported (S) or not supported (N) by the BMC.

Table 35. IPMI 1.5 command support

Command Name	NetFn	CMD	M/O	S/N	Comments
Chassis Device Commands					
Get Chassis Capabilities	Chassis	00h	O	S	
Get Chassis Status	Chassis	01h	O	N	
Chassis Control	Chassis	02h	O	N	
Chassis Reset	Chassis	03h	O	N	
Chassis Identify	Chassis	04h	O	N	
Set Chassis Capabilities	Chassis	05h	O	N	
Set Power Restore Policy	Chassis	06h	O	N	
Get System Restart Cause	Chassis	07h	O	S	
Set System Boot Options	Chassis	08h	O	N	
Get System Boot Options	Chassis	09h	O	N	
Get POH Counter	Chassis	0Fh	O	N	
Event Commands					
Set Event Receiver	S/E	00h	M	S	
Get Event Receiver	S/E	01h	M	S	
Platform Event (“Event Message”)	S/E	02h	M	S	
PEF and Alerting Commands					
Get PEF Capabilities	S/E	10h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Arm PEF Postpone Timer	S/E	11h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Set PEF Configuration Parameters	S/E	12h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Get PEF Configuration Parameters	S/E	13h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Set Last Processed Event ID	S/E	14h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Get Last Processed Event ID	S/E	15h	M	N	Additional constraints apply; see the IPMI 1.5 specification
Alarm Immediate	S/E	16h	O	N	Additional constraints apply; see the IPMI 1.5 specification
PET Acknowledge	S/E	17h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Sensor Device Commands					
Get Device SDR Info	S/E	20h	M	S	
Get Device SDR	S/E	21h	M	S	Additional constraints apply; see the IPMI 1.5 specification
Reserve Device SDR Repository	S/E	22h	M	S	Additional constraints apply; see the IPMI 1.5 specification
Get Sensor Reading Factors	S/E	23h	O	N	Additional constraints apply; see the IPMI 1.5 specification
M/O identifies if a command is Mandatory or Optional S/N identifies if a command is Supported or Not supported by the BMC					

Table 35. IPMI 1.5 command support (Continued)

Command Name	NetFn	CMD	M/O	S/N	Comments
Set Sensor Hysteresis	S/E	24h	O	S	
Get Sensor Hysteresis	S/E	25h	O	S	
Set Sensor Threshold	S/E	26h	O	S	
Get Sensor Threshold	S/E	27h	O	S	Additional constraints apply; see the IPMI 1.5 specification
Set Sensor Event Enable	S/E	28h	O	S	
Get Sensor Event Enable	S/E	29h	O	S	Additional constraints apply; see the IPMI 1.5 specification
Re-arm Sensor Events	S/E	2Ah	O	S	Additional constraints apply; see the IPMI 1.5 specification
Get Sensor Event Status	S/E	2Bh	O	S	
Get Sensor Reading	S/E	2Dh	M	S	
Set Sensor Type	S/E	2Eh	O	N	
IPM Device "Global" Commands					
Get Device ID	App	01h	M	S	
Cold Reset	App	02h	O	S	Additional constraints apply; see the IPMI 1.5 specification
Warm Reset	App	03h	O	N	
Get Self Test Results	App	04h	M	S	
Manufacturing Test On	App	05h	O	N	
Set ACPI Power State	App	06h	O	S	
Get ACPI Power State	App	07h	O	S	Additional constraints apply; see the IPMI 1.5 specification
Get Device GUID	App	08h	O	N	
Broadcast "Get Device ID"	App	01h	M	N	Additional constraints apply; see the IPMI 1.5 specification see the PICMG 3.0 specification
BMC Watchdog Timer Commands					
Reset Watchdog Timer	App	22h	M	S	
Set Watchdog Timer	App	24h	M	S	
Get Watchdog Timer	App	25h	M	S	
BMC Device and Messaging Commands					
Set BMC Global Enables	App	2Eh	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Get BMC Global Enables	App	2Fh	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Clear Message Flags	App	30h	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Get Message Flags	App	31h	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Enable Message Channel Receive	App	32h	O	N	
M/O identifies if a command is Mandatory or Optional S/N identifies if a command is Supported or Not supported by the BMC					

Table 35. IPMI 1.5 command support (Continued)

Command Name	NetFn	CMD	M/O	S/N	Comments
Get Message	App	33h	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Send Message	App	34h	O/M	S	Additional constraints apply; see the PICMG 3.0 specification
Read Event Message Buffer	App	35h	O	N	
Get BT Interface Capabilities	App	36h	O/M	N	Additional constraints apply; see the PICMG 3.0 specification
Get System GUID	App	37h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get Channel Authentication Capabilities	App	38h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get Session Challenge	App	39h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Activate Session	App	3Ah	O	N	Additional constraints apply; see the IPMI 1.5 specification
Set Session Privilege Level	App	3Bh	O	N	Additional constraints apply; see the IPMI 1.5 specification
Close Session	App	3Ch	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get Session Info	App	3Dh	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get AuthCode	App	3Fh	O	N	
Set Channel Access	App	40h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get Channel Access	App	41h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get Channel Info	App	42h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Set User Access	App	43h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get User Access	App	44h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Set User Name	App	45h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Get User Name	App	46h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Set User Password	App	47h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Master Write-Read	App	52h	O/M	S	Additional constraints apply; see the IPMI 1.5 specification
FRU Device Commands					
Get FRU Inventory Area Info	Storage	10h	M	S	
Read FRU Data	Storage	11h	M	S	
Write FRU Data	Storage	12h	M	S	
SDR Device Commands					
Get SDR Repository Info	Storage	20h	M	S	
M/O identifies if a command is Mandatory or Optional S/N identifies if a command is Supported or Not supported by the BMC					

Table 35. IPMI 1.5 command support (Continued)

Command Name	NetFn	CMD	M/O	S/N	Comments
Get SDR Repository Allocation Info	Storage	21h	O	N	
Reserve SDR Repository	Storage	22h	M	S	
Get SDR	Storage	23h	M	S	Additional constraints apply; see the PICMG 3.0 specification
Add SDR	Storage	24h	O/M	S	Additional constraints apply; see the PICMG 3.0 and IPMI 1.5 specifications
Partial Add SDR	Storage	25h	O/M	S	Additional constraints apply; see the PICMG 3.0 and IPMI 1.5 specifications
Delete SDR	Storage	26h	O	N	Additional constraints apply; see the IPMI 1.5 specification
Clear SDR Repository	Storage	27h	O/M	S	Additional constraints apply; see the PICMG 3.0 and IPMI 1.5 specifications
Get SDR Repository Time	Storage	28h	O/M	N	Additional constraints apply; see the IPMI 1.5 specification
Set SDR Repository Time	Storage	29h	O/M	N	Additional constraints apply; see the IPMI 1.5 specification
Enter SDR Repository Update Mode	Storage	2Ah	O	N	Additional constraints apply; see the IPMI 1.5 specification
Exit SDR Repository Update Mode	Storage	2Bh	M	N	Additional constraints apply; see IPMI 1.5 specification
Run Initialization Agent	Storage	2Ch	O	S	Additional constraints apply; see IPMI 1.5 specification
SEL Device Commands					
Get SEL Info	Storage	40h	M	S	
Get SEL Allocation Info	Storage	41h	O	N	
Reserve SEL	Storage	42h	O	S	Additional constraints apply; see the IPMI 1.5 specification
Get SEL Entry	Storage	43h	M	S	
Add SEL Entry	Storage	44h	M	S	Additional constraints apply; see the IPMI 1.5 specification
Partial Add SEL Entry	Storage	45h	M	S	Additional constraints apply; see the IPMI 1.5 specification
Delete SEL Entry	Storage	46h	O	S	
Clear SEL	Storage	47h	M	S	
Get SEL Time	Storage	48h	M	S	
Set SEL Time	Storage	49h	M	S	
Get Auxiliary Log Status	Storage	5Ah	O	N	
Set Auxiliary Log Status	Storage	5Bh	O	N	Additional constraints apply; see the IPMI 1.5 specification
M/O identifies if a command is Mandatory or Optional S/N identifies if a command is Supported or Not supported by the BMC					

8.2.2 PICMG 3.0 IPMI Command Support

Table 36 lists the commands defined in the PICMG 3.0 specification. These commands use NetFn = 2Ch/2Dh. The table indicates whether the command is mandatory (M) or optional (O) and whether a given command is supported (S) or not supported (N) by the BMC.

Table 36. PICMG 3.0 IPMI command support

Command Name	CMD	R	S	Comments
Get PICMG Properties	00h	M	S	
Get Address Info	01h	M	S	The BMC is required only to support a subset of this command; see PICMG 3.0 specification
Get Shelf Address Info	02h	O	N	
Set Shelf Address Info	03h	O	N	
FRU Control	04h	M	S	
Get FRU LED Properties	05h	M	S	
Get LED Color Capabilities	06h	M	S	
Set FRU LED State	07h	M	S	
Get FRU LED State	08h	M	S	
Set IPMB State	09h	M	S	
Set FRU Activation Policy	0Ah	M	S	
Get FRU Activation Policy	0Bh	M	S	
Set FRU Activation	0Ch	M	S	
Get Device Locator Record ID	0Dh	M	S	
Set Port State	0Eh	O/M	S	Mandatory for boards implementing E-Keying-governed interfaces
Get Port State	0Fh	O/M	S	Mandatory for boards implementing E-Keying-governed interfaces
Compute Power Properties	10h	M	S	
Set Power Level	11h	M	S	
Get Power Level	12h	M	S	
Renegotiate Power	13h	O	N	
Get Fan Speed Properties	14h	M	N	Command required by the BMC controlling Shelf fans
Set Fan Level	15h	O/M	N	Command required by the BMC controlling Shelf fans
Get Fan Level	16h	O/M	N	Command required by the BMC controlling Shelf fans
Bused resource	17h	O/M	S	Mandatory for boards implementing E-Keying-governed shared bus interfaces
Get IPMB Link Info	18h	O/M	N	Mandatory for BMC incorporating an IPMB-0 Hub

8.2.3 IPMI Bridging

CPUs on the IXB2850 board can communicate with the ShMC using the IPMI protocol. This communication is necessary for several functions (for example, Safe System Upgrade). The BMC is responsible for forwarding these IPMI messages appropriately.

The BMC provides a mechanism of bridging IPMI messages between the ShMC located on the IPMB and the board CPUs connected to the BMC via a UART or KCS connection. The **Send Message** command with response tracking should be used for this purpose.

Table 37 gives format of the **Send Message** request sent to the BMC, and Table 38 shows response message received by the requester.

Table 37. Send message request format

Field	Description
rsSA	Responder's Slave Address for the Send Message command
NetFn/RsLUN	NetFn=App (06h) and Responder's LUN for the Send Message command
Checksum 1	Checksum 1 for the Send Message command
rqSA	Requester's Slave Address for the Send Message command
rqSeq/rqLUN	Requester's Sequence Number and Requester's LUN for the Send Message command
CMD	Command = Send Message (34h)
Data 1	Track request (01b) and the Channel Number Channel Numbers: 0h – IPMB 4h – NPU 5h – AP 6h – AP_KCS
Data 2	Responder's Slave Address for remote message
Data 3	NetFn/Responder's LUN for remote message
Data 4	Checksum 1 for remote message
Data 5	Requester's Slave Address for remote message
Data 6	Requester's Sequence Number and Requester's LUN for remote message
Data 7	Command for remote message
Data 8:N	Data for remote message
Data N+1	Checksum 2 for remote message
Checksum 2	Checksum 2 for Send Message command

Table 38. Send message response format

Field	Description
rqSA	Requester's Slave Address for the Send Message command
NetFn/rqLUN	NetFn=App (07h) and Requester's LUN for the Send Message command
Checksum 1	Checksum 1 for the Send Message command
rsSA	Responder's Slave Address for the Send Message command
rqSeq/rsLUN	Requester's Sequence Number and Responder's LUN for the Send Message command
CMD	Command = Send Message (34h)
Completion Code	Completion Code for the Send Message command
Data 1	Requester's Slave Address for remote message
Data 2	NetFn / Requester's LUN (00b) for remote message
Data 3	Checksum 1 for remote message
Data 4	Responder's Slave Address for remote message
Data 5	Requester's Sequence Number and Responder's LUN for the remote message
Data 6	Command for the remote message

Table 38. Send message response format (Continued)

Field	Description
Data 7	Completion Code for the remote message
Data 8:N	Data for the remote message
Data N+1	Checksum 2 for the remote message
Checksum 2	Checksum 2 for the Send Message command

8.3 Communications with Other Processors

The BMC can communicate with other IXB2850 CPUs using the following methods:

- Over a UART connection
- Over the KCS interface

8.3.1 Serial Connections

An IXB2850 board contains a number of serial lines that connect the BMC with other CPUs on the baseboard and extension cards. The BMC can communicate over serial connections with the following CPUs:

- NPU
- Adjunct Processor on the PMC card

The BMC uses the IPMI protocol for communication with other CPUs over serial connections. Section 8.2, “IPMI Protocol Support” on page 90 describes in detail all the IPMI commands supported during communication over serial lines.

The encoding of the IPMB packets on a character-based serial interface is performed and understood by both the BMC and CPU. The encoding follows these rules:

- Each byte sent on serial interface consists of two fields (7 bits of data) and 1 bit start/stop message marker.
- Start/stop bit message marker; if 1, this byte is the first or the last in the IPMB packet
- 7 data bits; data taken from an IPMB message

8.3.2 KCS Connection

The PrPMC standard that defines the connectors of the PMC card does not include a UART connection as one of the mandatory connectors (J1, J2, and J3). The user-defined fourth connector (J4) can be used as a serial connection between processors on the baseboard and PMC card. The usage of this connector on the PMC card is vendor-dependent. The BMC is able to communicate with the CPU on a PMC card that is pin-compatible with the IXB2850 baseboard.

Vendor-independent communication with the CPU on the PMC card is possible only via the PCI bus. The BMC supports a KCS interface to allow this kind of connection. The IPMI protocol is used for communication with the PMC Adjunct Processor over the KSC interface. Section 8.2, “IPMI Protocol Support” on page 90 describes all IPMI commands supported during communication over the KCS interface.

The Linux operating system running on the PMC card can use the BMC Access module to communicate with the BMC.

8.4 Electronic Keying

IXB2850 boards, following the PICMG 3.0 specification, supports the Electronic Keying (E-Keying) mechanism on the following backplane interfaces:

- Fabric interface
- Base interface
- Update interface
- Clock interface

Since the BMC does not control access to these interfaces directly, it performs E-Keying operations via the NPU, which can set up backplane interfaces. To allow this cooperation and fully support the E-Keying functionality, the BMC maintains an E-Keying Status Word and forwards E-Keying IPMI messages to the NPU.

Cooperation between the BMC and NPU can be divided into the following phases:

- Phase 1 – NPU not powered
- Phase 2 – NPU runs Boot Monitor or diagnostics code
- Phase 3 – NPU runs the Linux operating system

Phase 1

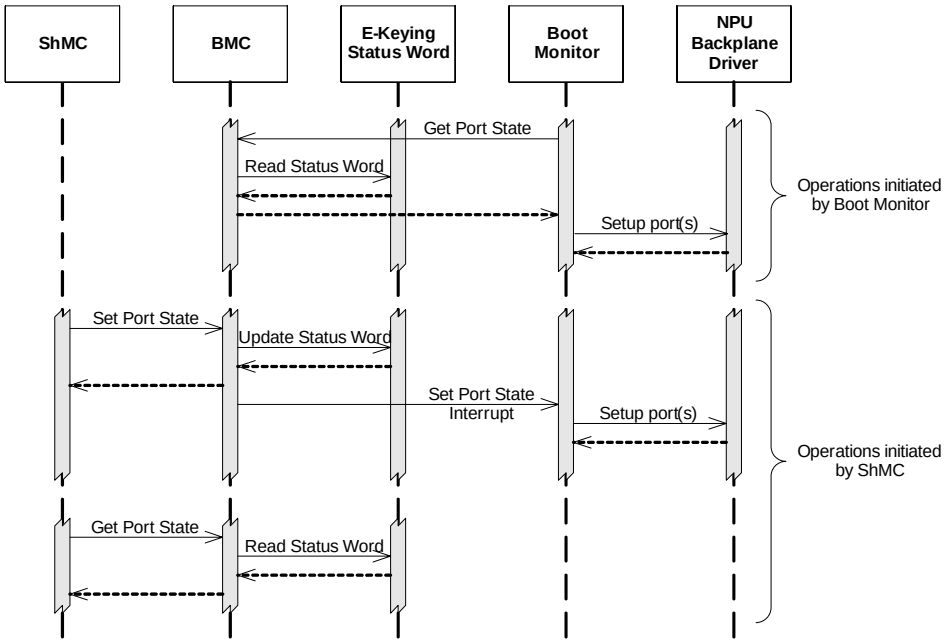
In phase 1, the NPU is not powered on and the board backplane interfaces are disabled. If the BMC receives a **Set Port State** or **Bus Resource Control** command in this phase, it modifies the
E-Keying Status Word accordingly. Since the NPU is not operational, no other action is performed.

If an IXB2850 board works without the chassis and the ShMC is not present, then the BMC modifies the E-Keying Status Word to indicate that all ports belong to the point-to-point interfaces (base interface, fabric interface and update interface) are enabled.

Phase 2

In phase 2, the NPU runs Boot Monitor or diagnostics code. In this phase, the NPU is not able to receive IPMI messages that are sent asynchronously by the BMC. It can only send a request message to the BMC and receive a response. In this phase, only the NPU can start message exchange between the NPU and BMC. Figure 31 is a UML diagram illustrating the cooperation between the NPU and BMC in this phase.

Figure 31. E-keying support in phase 2 (BMC and NPU cooperation)

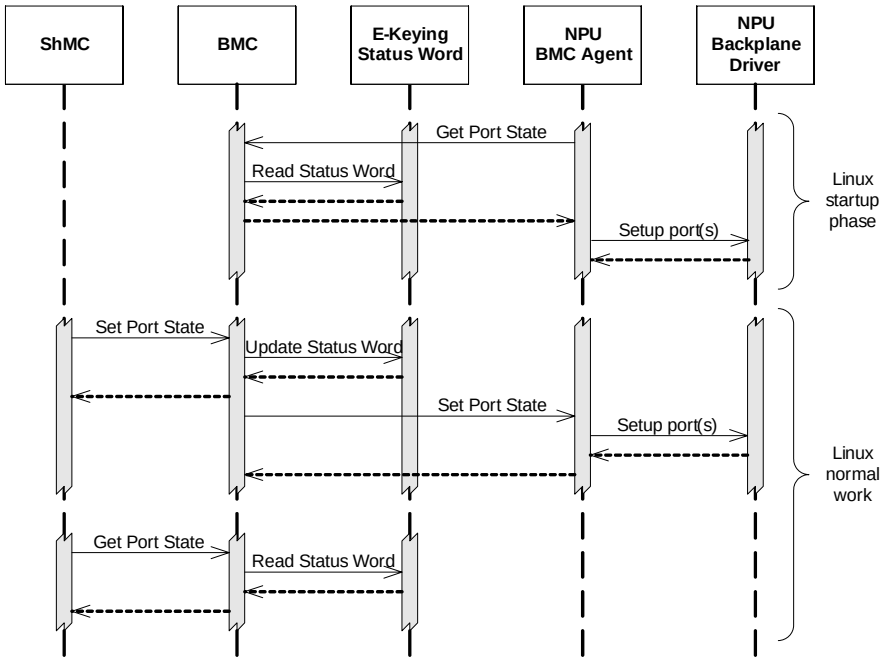


If the NPU needs to perform any operation on board backplane interfaces, it should first obtain the status of these interfaces from the BMC by sending a **Get Port State** or **Bus Resource Control** command. If the BMC receives a **Set Port State** or **Bus Resource Control** command from the ShMC, it generates an NPU interrupt and the Boot Monitor configures the backplane interfaces accordingly.

Phase 3

In phase 3, the board is fully operational and running Linux. In this phase, the NPU is running the BMC Agent application, which is able to receive E-Keying IPMI messages from the BMC. Figure 32 is a UML diagram illustrating cooperation between the NPU and the BMC in this phase.

Figure 32. E-keying support in phase 3 (BMC and NPU cooperation)



In the Linux startup phase, the NPU BMC Agent application configures all board backplane interfaces according to information received from the BMC. During normal Linux operation, the BMC updates the E-Keying Status Word according to commands received from the ShMC and forwards these as **Set Port State** and **Bus Resource Control** commands to the NPU BMC Agent, which is responsible for handling these commands. See Section 9.6.10, “BMC Agent” on page 127 for details.

8.4.1 E-Keying Status Word

The E-Keying Status Word contains information about the current state of all board backplane interfaces. This status word is maintained by the BMC and stored in a BMC EEPROM, since the state of backplane interfaces must not change during accidental BMC reset.

Note: The E-Keying Status Word is used internally by the BMC. It is not accessible to external applications.

Table 39 describes the format of the E-Keying Status Word.

Table 39. E-keying status word format

Bit	Name	Description
0	FI Ch 1 Port 0	State of Port 0 from fabric interface channel 1 (enabled or disabled)
1	FI Ch 1 Port 1	State of Port 1 from fabric interface channel 1 (enabled or disabled)
2	FI Ch 1 Port 2	State of Port 2 from fabric interface channel 1 (enabled or disabled)
3	FI Ch 1 Port 3	State of Port 3 from fabric interface channel 1 (enabled or disabled)
4	FI Ch 2 Port 0	State of Port 0 from fabric interface channel 2 (enabled or disabled)
5	FI Ch 2 Port 1	State of Port 1 from fabric interface channel 2 (enabled or disabled)
6	FI Ch 2 Port 2	State of Port 2 from fabric interface channel 2 (enabled or disabled)

Table 39. E-keying status word format (Continued)

Bit	Name	Description
7	FI Ch 2 Port 3	State of Port 3 from fabric interface channel 2 (enabled or disabled)
8	BI Ch 1 Port 0	State of Port 0 from base interface channel 1 (enabled or disabled)
9	BI Ch 2 Port 0	State of Port 0 from base Interface channel 2 (enabled or disabled)
10	UI Ch 1 Port 0	State of Port 0 from update interface channel 1 (enabled or disabled)
11	CLK Gr 1	State of synchronization clock interface group 1 – 1A/B (control bus or not)
12	CLK Gr 2	State of synchronization clock interface group 2 – 2A/B (control bus or not)
13	CLK Gr 3	State of synchronization clock interface group 3 – 3A/B (control bus or not)
14 to 16	Reserved	Reserved for the future use

8.4.2 Point-to-Point Connectivity Records

The Electronic Keying mechanism requires a description of board backplane interface configurations to be a part of the board FRU Information. The FRU Information for IXB2850 board hardware modules includes the Point-to-Point Connectivity Records describing the backplane interfaces supported by the hardware modules.

8.4.2.1 Baseboard Point-to-Point Connectivity Record

Table 40 describes the baseboard Point-to-Point Connectivity Record. An IXB2850 baseboard supports the following backplane interfaces:

- Fabric interface – one channel with four ports
- Base interface – two channels with one port

Table 40. Baseboard point-to-point connectivity record

Offset	Length (bytes)	Value	Definition
0	1	C0h	Record Type ID
1	1	02h	End of List/Version: • 7:7 – End of List = 0h (No) • 6:4 – Reserved = 0h • 3:0 – Record format version =2h
2	1		Record Length
3	1		Record Checksum. Holds the zero checksum of the record.
4	1		Header Checksum. Holds the zero checksum of the header.
5	3	5A3100h	Manufacturer ID. Written as the three byte ID assigned to PICMG.
8	1	14h	PICMG Record ID for the Board Point-to-Point Connectivity Record
9	1	00h	Record Format Version
10	1	0	OEM GUID Count
11	8		Link Descriptor list. See: • Table 41, “Link descriptors for fabric interface channel 1” on page 102 • Table 42, “Link descriptors for base interface channels” on page 102.

Table 41. Link descriptors for fabric interface channel 1

Field	Value	Description
Link Designator	111101000001b	Channel Number = 1; Interface = Fabric; Port 0,1,2,3 Enabled
Link Type	02h	Link Type = PICMG 3.1 Ethernet
Link Type Extension	0000b	Link Type Extension = 1000BASE-BX
Link Grouping ID	00h	Link Grouping ID = Independent

Table 42. Link descriptors for base interface channels

Field	Value	Description
Link Designator	000100000001b	Channel Number = 1; Interface = Base; Port 0 Enabled
Link Type	01h	Link Type= PICMG 3.0 Base Interface
Link Type Extension	0000b	Link Type Extension
Link Grouping ID	00h	Link Grouping ID = Independent
Link Designator	000100000010b	Channel Number = 2; Interface = Base; Port 0 Enabled
Link Type	01h	Link Type= PICMG 3.0 base interface
Link Type Extension	0000b	Link Type Extension
Link Grouping ID	00h	Link Grouping ID = Independent

8.4.2.2 FIC Point-to-Point Connectivity Record

Table 43 describes the Fabric Interface Card (FIC) Point-to-Point Connectivity Record. The FIC supports the following backplane interfaces:

- Fabric interface – one channel with four ports
- Fabric interface – one channel with two ports
- Fabric interface – one channel with one port

Table 43. FIC point-to-point connectivity record

Offset	Length (bytes)	Value	Definition
0	1	C0h	Record Type ID
1	1	02h	End of List/Version: • 7:7 – End of List = 0h (No) • 6:4 – Reserved = 0h • 3:0 – Record format version =2h
2	1		Record Length
3	1		Record Checksum. Holds the zero checksum of the record.
4	1		Header Checksum. Holds the zero checksum of the header.
5	3	5A3100h	Manufacturer ID. Written as the three byte ID assigned to PICMG.
8	1	14h	PICMG Record ID for the Board Point-to-Point Connectivity Record
9	1	00h	Record Format Version
10	1	0	OEM GUID Count
11	8		Link Descriptor list. See Table 44, "Link descriptors for fabric interface channel 2 (example for four ports)" on page 103.

Table 44. Link descriptors for fabric interface channel 2 (example for four ports)

Field	Value	Description
Link Designator	111101000010b	Channel Number = 2; Interface = Fabric; Port 0,1,2,3 Enabled
Link Type	02h	Link Type = PICMG 3.1 Ethernet
Link Type Extension	0000b	Link Type Extension = 1000BASE-BX
Link Grouping ID	00h	Link Grouping ID = Independent

8.5 Board UART Configuration

IXB2850 boards contain one RJ45 connector located on the front panel that is used for a console connection. The baseboard and extension cards contain processors that support one or more UART devices. These are:

- NPU – internal and external UARTs
- BMC – internal UART and external quad UART
- PMC – one UART

The front panel RJ45 console connector can work in single or dual mode (requires a special cable, IXB3GDEBUGCABLE, see Section 12.8.1, “Debug Console Cable Specification” on page 165 for details). In single mode, this connector works like a Cisco* single UART with full modem signaling (RTS, CTS, DTR and CD). In dual mode, there are two UARTs available on this connector, but the second line only supports RX and TX.

The console port is preconfigured (no DIP switches) to access the NPU and BMC internal UARTs only. See Section 8.5.1 following for more details.

8.5.1 UART Hardware Configuration

The front panel console connector is not hardware or software configurable, but is preset to select the NPU and BMC internal UARTs.

8.5.2 CPLD Internal UART Cross-Connections

The CPLD also establishes internal cross-connection between following UARTs:

- BMC external UART #1 – MMC #1 UART, not applicable to IXB2850 boards
- BMC external UART #2 – MMC #2 UART
- BMC external UART #3 – PMC UART
- BMC external UART #4 – FIC UART

These cross-connections are established if none of the UARTs in a pair is connected to the front panel console connector.

8.6 Non-Volatile Storage Programming

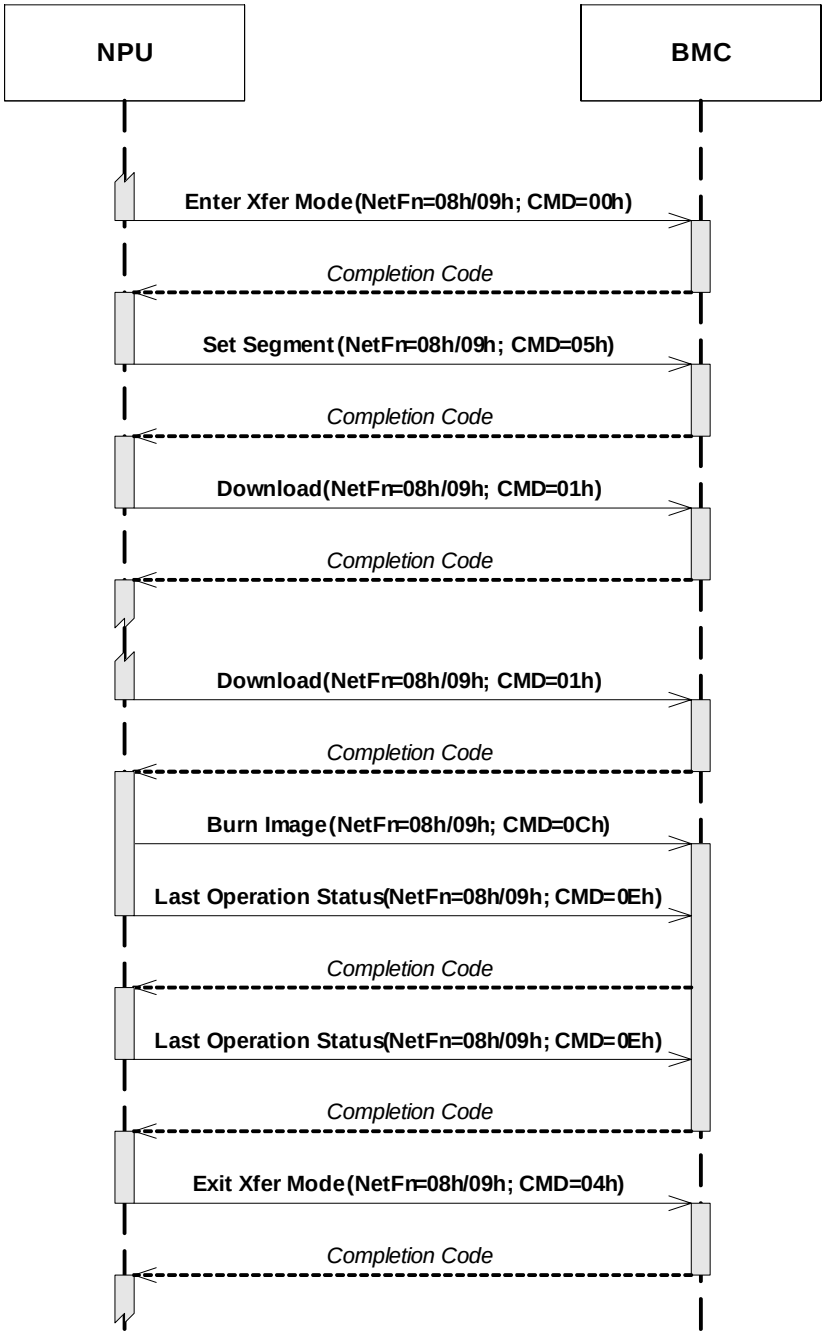
Table 45 gives the non-volatile memory areas that the BMC can access and update.

Table 45. Non-volatile memory available to the BMC

Segment ID	Description	Non-volatile storage
1	Baseboard FRU Information	BB ID EEPROM
2	Mezzanine 1 FRU Information	MMC1 ID EEPROM; not applicable to IXB2850 boards
3	Mezzanine 2 FRU Information	MMC2 ID EEPROM
4	Mezzanine 1 MIC FRU Information	MIC1 ID EEPROM; not applicable to IXB2850 boards
5	Mezzanine 2 MIC FRU Information	MIC2 ID EEPROM
6	FIC FRU Information	FIC ID EEPROM
7	NPU Module FRU Information	NP MODULE ID EEPROM
8	RTM FRU Information	RTM ID EEPROM; not applicable to IXB2850 boards
9	PMC FRU Information	PMC ID EEPROM
10	BMC Operational Code	Flash
11	BMC Image 1	Flash
12	BMC Image 2	Flash
13	SEL repository	Flash
14	Baseboard SDR Information	BB ID EEPROM
15	Mezzanine 1 SDR Information	MMC1 ID EEPROM; not applicable to IXB2850 boards
16	Mezzanine 2 SDR Information	MMC2 ID EEPROM
17	Mezzanine 1 MIC SDR Information	MIC1 ID EEPROM; not applicable to IXB2850 boards
18	Mezzanine 2 MIC SDR Information	MIC2 ID EEPROM
19	FIC SDR Information	FIC ID EEPROM
20	NPU Module SDR Information	NP MODULE ID EEPROM
21	RTM SDR Information	RTM ID EEPRO; not applicable to IXB2850 boards M
22	PMC SDR Information	PMC ID EEPROM

The BMC can update the content of all these areas using the IPMI protocol. The UML diagram in Figure 33 shows the IPMI messages that are exchanged during non-volatile storage programming.

Figure 33. Non-volatile storage programming



Note: The ShMC can also use the method shown in Figure 33 to upgrade non-volatile storage on the BMC.

9.0 Network Processor Firmware

9.1 NP Local Flash Memory Organization

On IXB2850 boards, the onboard flash memory on the Network Processor (NP) is divided into Flash Image System (FIS) partitions. Each partition layout is constant, that is, it cannot be automatically changed during software upgrade. Information in flash is stored in the last partition, the “FIS directory”.

Each FIS partition contains an image and an image property header (Figure 34) at the end of the partition:

Figure 34. FIS partition image header structure

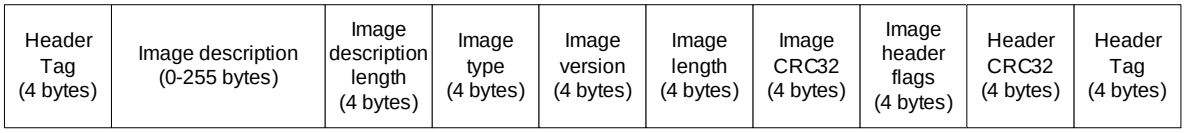


Table 46 gives the fields of the image header in decode order, starting from the end.

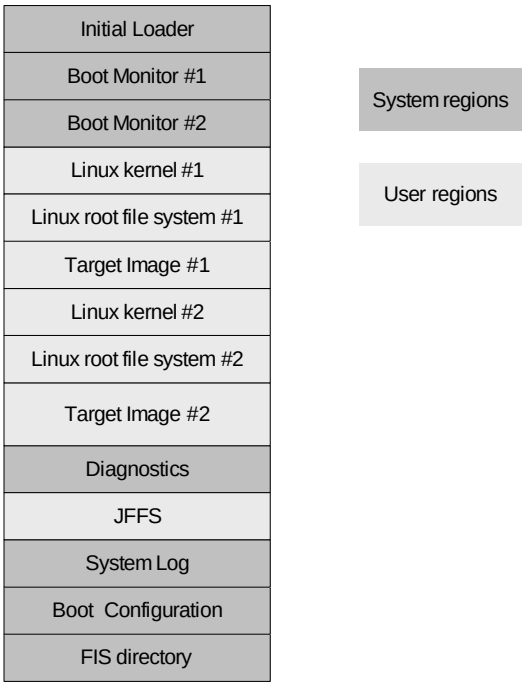
Table 46. NPU local flash memory fields

Name	Bytes	Purpose
Header tags	4	There are two header tags, one at the beginning of the header and one at the end. Header tags are used for validating the image header. The header tag value is 0x12345678. If the header tags have a different value, it is assumed that the image and its header are corrupted.
Header CRC32	4	The image header checksum used for verifying header contents (all header field except header CRC and header tags)
Image header flags	4	A bit field for marking additional features support
Image CRC32	4	The image checksum used for image verification
Image length	4	The real length of the image
Image version	4	Image version information
Image type field	4	Type of the image
Image description length	4	Length of the image description field
Image description field	0 to 255	Image version information. This field is aligned to 4 octets.

All 16/32-bit values are stored in the system native Endian type; all fields with variable length are stored using a strings-like method (byte after byte) starting from the lowest address.

The NPU local flash memory layout is shown in Figure 35.

Figure 35. Logical NPU flash memory layout



The NPU flash contains two types of regions:

- System Space – These flash sectors are occupied by system blocks necessary for IXB2850 board operation and cannot be used for other purposes. See Table 47 for details.

Table 47. NPU flash memory system regions

Sector	Purpose
Initial Loader	Stored in the first FIS partition. The partition should not exceed one or two flash blocks and should be locked to prevent accidental erase and program operations. It is responsible for preliminary board initialization, selecting Boot Monitor image, loading it to RAM memory and starting it. By default the Initial Loader does not provide CLI, however it is possible to stop the booting procedure and use the loader's command line to update a Boot Monitor image for example. This is done mostly in emergency cases.
Boot Monitor	Responsible for board initialization and for selecting and starting the Linux Kernel. The Boot Monitor provides CLI. In case of any error, the user can perform additional diagnostics and recovery actions.
Diagnostics	Allows the performing of detailed hardware tests and is helpful in determining the overall condition of the board.
System Log partition	A local data storage for Boot Monitor and Diagnostics.
Boot Configuration partition	Handles all necessary information that is used during system start, for example, IP address configuration (static IP configuration or configuration using BOOTP requests) and run-time image loading script.
FIS directory	Contains information on all flash entries.

- User Space – These flash sectors are designed for user purposes and their definitions, placements, and sizes can be modified according to particular user requirements. See Table 48 for details.

Table 48. NPU flash memory user regions

Sector	Purpose
Kernel partition	Contains a Linux kernel image.
Linux root file system partition	Contains a Linux root file system.
Target image	A compressed read-only file system (CRAMFS) that is mounted during boot time by the Linux kernel.
JFFS2 partition	An optional partition that may be used when Linux applications require a local read-write file system. This partition is always mounted, but the appropriate Memory Technology Device (MTD) exists only if the partition is manually created from the boot PROM level.

9.2 Boot Configuration

The NPU Boot Configuration occupies one flash memory sector and contains the parameters described in Table 49.

Table 49. NPU boot configuration parameters

Parameter	Description
POST type	A flag indicating the type of Power-On Self Test (POST) performed by the Boot Monitor during board startup. This parameter can have one of two values: • short POST – short POST set will be performed • long POST – long POST set will be performed
Soft Reset	A flag indicating if a soft reset should be performed. This flag is used by the Initial Loader to properly configure memory controllers and by POST to perform the appropriate POST set. This flag can have one of two values: • normal startup • soft reset ***Note:*** Soft reset does **not** perform SRAM write tests so that the content of SRAM can be preserved.
Run-time image loading	A flag indicating whether the Boot Monitor will load and execute the run-time image for the NPU. This flag can have one of two values: • image load – The Boot Manager attempt to load and execute run-time image code • command prompt – Boot Manager does not attempt to load run-time image code, displays a command prompt and waits for operator action (for example, diagnostics execution).
Ethernet BOOTP request number	The number of BOOTP requests that are sent by the Boot Manager over Ethernet interfaces to obtain the IP configuration.
SRAM initialization	Depending on the flag, the SRAM memory is either rewritten with its own content or filled with zeros.
Script timeout	Number of seconds of delay before launching the boot script.
Ethernet tracing	A flag indicating whether traffic that comes over Ethernet interfaces will be traced on the Boot Monitor console.
Debug Ethernet IP configuration type	A flag indicating the type of IP configuration used for the debug Ethernet interface during run-time image loading. This flag can have one of two values: • automatic – IP configuration is obtained via the BOOTP protocol • static – IP configuration stored as boot configuration parameter are used

Table 49. NPU boot configuration parameters (Continued)

Parameter	Description
Debug Ethernet IP configuration	The static IP configuration for debug Ethernet interface. The following information is stored here: • IP address • Subnet mask • Default gateway
Base Interface #1 † IP configuration type	A flag indicating the type of IP configuration used for the base interface #1 during run-time image loading. This flag can have one of two values: • automatic – IP configuration is obtained via the BOOTP protocol • static – IP configuration stored as a boot configuration parameter is used
Base Interface #1 † IP configuration	The static IP configuration for base interface #1. The following information is stored here: • IP address • Subnet mask • Default gateway
Base Interface #2 † IP configuration type	A flag indicating the type of IP configuration used for the base interface #2 during run-time image loading. This flag can have one of two values: • automatic – IP configuration is obtained via the BOOTP protocol • static – IP configuration stored as a boot configuration parameter is used
Base Interface #2 † IP configuration	The static IP configuration for base interface #2. The following information is stored here: • IP address • Subnet mask • Default gateway
Run-time image searching sequence	A script containing a set of commands that are executed by the Boot Manager to load the run-time image for the NPU.
† By default, RedBoot* routes the Intel® 82546 Dual Port Gigabit Ethernet Controller via the crosspoint switch and the baseboard PHYs to base interfaces. See Figure 7, "IXB2850 board block diagram" on page 26.	

The Boot Configuration parameters can be changed using dedicated Boot Monitor commands. The BMC and ShMC are also able to change these parameters via a dedicated OEM FRU record.

9.3 Initial Loader

The Initial Loader occupies the first sector (or sectors) of flash memory associated with a particular processor, that is, the boot sector. The boot sector must be locked (write-protected) to avoid the possibility that a user accidentally overwrites the boot sector with an incorrect image, which would render the board unusable and requires reparation with hardware tools such as a JTAG flash programmer.

The Initial Loader is the first code that the processor executes from flash after it is released from the reset state (note that the BMC controls reset of the NPU). This code performs the following actions:

- Slow port initialization
- Slow port DIP switch detection (allows executing alternate boot code such as VxWorks* boot)
- PLL configuration (DRAM, SRAM and Advanced Peripheral Bus [APB], an internal bus used for accessing UART, timers, GPIOs, slow port, etc.)
- NPU internal UART initialization
- GPIO initialization
- One of the Boot Monitor codes is selected and executed from flash

9.3.1 Soft Reset Procedure

The IXB2850 board supports the “Soft Reset” startup procedure. This feature has been designed to ensure that SRAM and DRAM memory content is preserved during this kind of boot procedure. The following actions of the standard boot procedure are **not performed** during the Soft Reset startup procedure:

- Reset and initialization of SRAM memory controller
- PLL configuration for SRAM memory controller
- All destructive SRAM test performed by POST

The Initial Loader and Boot Monitor can recognize that the Soft Reset boot procedure is being performed using the Soft Reset flag in the NPU boot parameters stored on flash memory (see Table 49, “NPU boot configuration parameters” on page 108 for details) and by checking whether the memory controllers are already initialized.

9.4 Boot Monitor

The Boot Monitor is RedBoot from Red Hat*. See the *RedBoot User’s Guide* (http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html) for details concerning usage and configuration.

The Boot Monitor contains the following components:

- Hardware initialization code
- Power-On Self Test (POST)
- Boot Manager
- Utilities, which include:
  - Flash Memory tool – enables the writing/reading to/from a flash memory
  - Xmodem – tool used for downloading from serial port (UART)
  - BOOTP client – tool that allows the retrieval of the IP configuration (local IP address, subnet mask, default gateway IP address and default server’s IP address) for debug Ethernet interface and AdvancedTCA* base interface
  - TFTP client – tool for download code from a server using the TFTP protocol
  - Command Line Interface – used to control and manage boot process and run other utilities
  - System Events Log – stores events that appear during Boot Monitor operation
  - IPMI agent – enables communication between the NPU and BMC over a UART connection
- Drivers, which include:
  - UART drivers – for a set of UARTs available (internal NPU UART used for console, external UARTs used for debug console, and for communication with BMC).
  - Ethernet controller driver – a driver for the Cirrus Logic* CS8900A Ethernet controller dedicated to console and debug purposes. The Boot Manager can use such an interface for TFTP code downloading.
  - Gigabit Ethernet controller driver – a driver for the Intel 82546 Dual Port Gigabit Ethernet Controller connected to the NPU via the PCI bus. This controller enables the NPU to access the AdvancedTCA base interface. The Boot Manager can use the AdvancedTCA base interface for TFTP code downloading.

- PHY driver – a driver for the Marvell* Alaska 1011 Gigabit Ethernet PHY. This PHY is one of several devices connecting the 82546 Dual Port Gigabit Ethernet Controller to the AdvancedTCA base interface.
- Crosspoint driver – a driver for the Analog Devices* AD8152 crosspoint switch. This switch has direct connection to the AdvancedTCA base interface and should be properly configured to allow the 82546 Dual Port Gigabit Ethernet Controller to connect to the base interface.

9.4.1 Hardware Initialization

The first operation performed by the Boot Monitor is basic hardware component initialization and tests. These operations constitute a short Power On Self Test set. The Boot Monitor initializes and performs basic tests on all hardware components that do not contain a dedicated Linux driver, performs reinitialization during Linux startup and performs other tasks that are essential for Boot Monitor work. The hardware initialization tasks include:

- DRAM memory controller initialization
- Scratchpad memory test
- DRAM memory test – limited fragment of DRAM, used by Boot Monitor, is tested with pattern tests of each memory location within tested area
- DRAM scrubbing, ECC enabling
- SRAM memory controller initialization
- PCI bus initialization
- Standard RedBoot initialization procedure, including:
  - 16550 UART device initialization
  - Flash initialization
  - Debug Ethernet (CS8900A) controller

After the basic hardware initialization, the Boot Monitor establishes connection with the BMC over a UART connection and sends to the BMC the “Boot started” IMPI event message (see Table 52, “IPMI event data for boot class” on page 116 for details). This event informs the BMC that the NPU is currently running the Boot Monitor.

9.4.2 FRU Information Retrieval

The Boot Monitor is responsible for retrieving, from the BMC, “FRU Information” about each hardware module on the IXB2850 board. This information is used by the Boot Monitor to properly initialize and test baseboard and extension cards. FRU Information is also stored in EEPROM memory for future use by the Linux operating system (see Section 9.7, “Baseboard Driver” on page 128).

FRU Information contains records describing the MAC addresses for all blade Ethernet devices. The Boot Monitor verifies the MAC addresses configured in board Ethernet devices (CS8900 and 82546 Dual Port Gigabit Ethernet Controller) with the contents of FRU Information at every startup and changes a device’s configuration if MAC addresses has been modified.

One of the baseboard FRU Information MultiRecord entries contains NPU boot parameters. If the Boot Monitor detects a new boot parameters definition (a dedicated flag in this record – see Section 4.6.2.5, “Processors Boot Parameters Record” on page 48 for details), it modifies the non-active set of NPU boot parameters (Boot Monitor Config) stored in flash memory with the content of the appropriated FRU Information record and modifies the Boot Monitor Config Status Word to run the new Boot Monitor Configuration in “safe mode” (see Section Section 11.2.2.2, “Image

Upgrade and Automatic Rollback" on page 133 for details). The Boot Monitor also updates the NP Module FRU Information to clear the flag that indicates the presence of a new NPU boot parameter set. These operations are performed only if the Boot Monitor Config Status Word indicates that none of Boot Monitor Configurations are marked to run in "safe mode". If the Boot Monitor Config Status Word indicates that one of the Boot Monitor Configurations is marked to run in "safe mode", no action is performed and the Boot Monitor Config upgrade procedure is postponed until the next NPU startup.

9.4.3 NPU Power On Self Test

The Power-On Self Test (POST) performs initialization and basic tests on the Intel XScale® core and the main baseboard components that the XScale core depends on. These tests are designed so that cabling to the board is not required.

POST is divided into two phases, the first of which is always executed. This first phase cannot be repeated on user request (the only way to repeat the tests are by rebooting the board). POST covers these basic tests:

- XScale core scratchpad memory
- RAM
- I^2C (EEPROM) Interface
- I^2C (TCAM) Interface
- DRAM memory
- QDR SRAM and TCAM cards (if present)
- PCI bus

The first phase of POST is executed by ROM code, but the main part of POST is the first component of Boot Monitor executed after copying to DRAM.

Note: The Boot Monitor starts operating from flash, performs basic POST, then copies itself from flash to DRAM, and begins executing code from DRAM.

The second stage can be optionally executed and covers more specific tests. The configuration is changed by setting long POST in flash config using the fconfig command and setting the "skip POST option" to false.

Note: Tests belonging to the second stage can be repeated on user request using the command line interface when the user breaks RedBoot execution by pressing ^C. The reason for making the second stage of POST optional is to shorten the boot time (by default, this phase of POST is disabled).

The second POST test phase covers:

- UART
- DRAM
- QDR SRAM
- PCI bus
- BMC
- Ethernet slow port
- Microengines
- Slow port
- Interrupts

- XScale core
- MSF
- GPIO
- I^2C
- LEDs
- Mezzanine with Gigabit Ethernet
- FIC (IXB28504XGBEFSx boards only)
- Telecom clock test

The POST results are displayed on the console and saved in the System Events Log (SEL). After the boot process is finished, the results can be retrieved and used by an operating system.

Every POST error is reported to the ShMC via an IPMI event message sent to BMC as described in Section 9.4.7, “IPMI Event Messaging” on page 115.

See Appendix B, “Power On Self Test” for details of each POST test.

9.4.4 Boot Manager

The Boot Manager is the main part of Boot Monitor. It is responsible for loading and executing the run-time image for the NPU. The Boot Manager attempts to load and execute run-time image code according to the Boot Configuration parameter – run-time image loading. If this parameter is set to the command prompt value, the Boot Manager displays a command prompt only and waits for operator action. Otherwise, the Boot Manager reads the run-time image searching sequence script, a pointer to which is stored as one of the Boot Configuration parameters on flash memory and the operations in this script are performed. If Boot Manager is unable to load the run-time image from one of the sources, then it tries to use the next one. If all run-time image code loading attempts failed, the Boot Monitor displays an error message on the console, stores an appropriate event in the System Log and sends the event to the ShMC.

The run-time image can be loaded from the following sources:

- flash memory – The run-time image can be loaded from local flash memory using the Flash Memory tool.
- UART – The run-time image can be loaded over serial interface using Xmodem utility
- debug Ethernet – The run-time image can be loaded from an external server over the debug Ethernet interface using the TFTP protocol. The interface IP configuration necessary for the TFTP protocol can be obtained from two sources:
  - Boot Configuration parameter - debug Ethernet IP Configuration
  - BOOTP

  The selection of the appropriate configuration is determined by the Boot Configuration parameter - debug Ethernet IP Configuration type. If an automatic configuration is selected, then the Boot Manager attempts to obtain the IP configuration for the debug Ethernet interface via the BOOTP protocol. The maximum number of BOOTP requests sent by the Boot Manager is determined by another Boot Configuration parameter - debug Ethernet BOOTP request number. If it is not possible to obtain the IP configuration, then the loading of the run-time image over the debug Ethernet interface fails and the Boot Manager attempts to use another source.

- base interface - run-time image can be loaded from the external server over the base interface on the AdvancedTCA backplane using TFTP protocol. The 82546 Dual Port Gigabit Ethernet Controller is used by the NPU to access the AdvancedTCA base interface. The Boot Manager loads a network driver for this controller and then configures MAC, PHY and an analog crosspoint switch on baseboard. The base interface IP configuration necessary for TFTP protocol can be obtained from two sources:
  - Boot Configuration parameter - base interface IP configuration
  - BOOTP

  The selection of the appropriate configuration is determined by the Boot Configuration parameter - base interface IP configuration type. If an automatic configuration is selected, the Boot Manager attempts to obtain the IP configuration for the base interface via BOOTP. The maximum number of BOOTP requests sent by the Boot Manager is determined by the Boot Configuration parameter - base interface BOOTP request number.

The Boot Manager work can be interrupted at any stage via console (Ctrl-C). In this case, the Boot Manager finishes the currently executed command, then displays a console prompt and waits for operator action.

9.4.5 Boot Monitor Console

The Boot Monitor uses the console as an output port following UART initialization. The console is used to report the status (progress and errors) of the POST phases that follow.

The console can also be used as an interactive Command Line Interface (CLI) that allows the user to manage the boot process or run utilities built into the Boot Monitor.

The CLI is activated (that is, the Boot Monitor breaks normal execution and displays the console prompt) in the following cases:

- After completion of POST, if the Boot Configuration parameter – run-time image loading is set to the command prompt value
- When the user types Ctrl-C on the console during POST execution (after UART ports have been initiated and tested). In this case, the currently executing phase of POST is completed and then the Boot Monitor enters the console prompt, waiting for operator action.
- When all run-time imaged code loading attempts failed.
- When Ctrl-C was pressed during execution of a startup script.
- When Ctrl-C is received from the serial port at any time during Xmodem download.

The CLI provided by Boot Monitor allows:

- **Execution of download utilities –** The user can download a file into a selected memory area using Xmodem (via serial port) or TFTP (via Ethernet interface).
- **Code execution –** The user can execute the code located in the selected address of RAM or flash.
- **Execution of flash tools –** The user can write a selected area of memory into flash (for example, a previously downloaded file into RAM). The image can also be loaded from flash into RAM, or can be deleted from flash.
- **Definition of configuration –** The user can change the Boot Configuration parameters stored in a dedicated sector of flash memory.

A detailed specification of the console commands can be found in the *RedBoot User Guide* (see http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html).

9.4.6 Boot Monitor Device Drivers

The Boot Monitor device drivers are described in Table 50.

Table 50. Boot monitor device drivers

Driver	Description
UART device driver	Used for serial interface configuration and data transmission/reception over serial interfaces. This driver allows configuration of the following UART parameters: • Speed – default set to 115200 bps • Data bits – default is 8 • Parity – default is none • Stop bits – default is 1 • Flow control – default is none
Debug Ethernet device driver	Used for the CS8900A Ethernet MAC controller configuration and data transmission/reception over debug Ethernet interface.
PCI Gigabit Ethernet MAC device driver	Used for the 82546 Dual Port Gigabit Ethernet Controller configuration and data transmission/reception over PCI Ethernet interface connected to the AdvancedTCA base interface. This driver configures the following devices: • Marvell* Alaska 1011 Gigabit Ethernet PHY – Sets GBIC mode for interfaces connected to the AdvancedTCA base interface. • AD8152 crosspoint switch – Connects the 82546 Dual Port Gigabit Ethernet Controller to the AdvancedTCA base interface via the Alaska Gigabit Ethernet PHY according to the E-Keying base interface state.

9.4.7 IPMI Event Messaging

The NPU on an IXB2850 board acts as the CPU sensor and generates events destined for the BMC SEL or the ShMC. These events have the format given in Table 51. The Event Data has a format that complies with the IPMI 1.5 specification.

The Table 51 gives the format of the Event Data 1 field.

Table 51. Event data 1 field format

Bits	Value	Description
7:6	10b	Indicates that the Event Data 2 field contains an OEM code
5:4	10b	Indicates that the Event Data 3 field contains an OEM code
3:0		Offset from the Event/Reading Type Code from the SDR describing a given CPU sensor. This offset has the meaning of the event class. For details, see Table 22, “Processor SDR format” on page 56.

The Event Data 2 and Event Data 3 fields contain additional information about events belonging to the appropriated event class. The Table 52 gives contents of the Event Data 2 and Event Data 3 fields for the Boot class used by the NPU acting as sensor.

Table 52. IPMI event data for boot class

Data 2 (Event)	Data 3 (Event Additional Information)	Description
00h – Boot started	Reset type: • 00h – Hard reset • 01h – Soft reset	Event generated by the Boot Monitor during startup procedure.
01h – POST result	POST result: • 00h – Pass • 01h – Failed	Event generated by the Boot Monitor after POST execution. This event message contains overall POST result.
02h – Run-time image error	Not used (FFh)	Event generated by the Boot Monitor if no run-time image can be loaded and executed. See Section 9.4.4, “Boot Manager” on page 113 for details.
03h – Boot completed	Not used (FFh)	Event generated by the BMC Agent during startup. See Section 9.6.10.1, “BMC Agent Startup Activity” on page 127 for details.

Table 53 gives the contents of the Event Data 2 and Event Data 3 fields for the POST class used by the NPU acting as sensor. These IPMI events messages are generated by POST.

Table 53. IPMI event data for the POST class

Data 2 (Event)	Data 3 (Additional Information)	Description
00h – UART	Test ID / UART#	UART POST errors. Test IDs (4 MSb) and UART numbers are listed in Section B.1.1, “UART POST” on page 223.
01h – DRAM	Test ID	DRAM POST errors. Test IDs (8 bits) are listed in Section B.1.2, “Memory POST” on page 224.
02h – QDR SRAM	Test ID	QDR SRAM POST errors. Test IDs (8 bits) are listed in Section B.1.2, “Memory POST” on page 224.
04h – PCI	Test ID / Bridge#	PCI POST errors. Test IDs (4 MSb) are listed in Section B.1.3, “PCI POST” on page 224. PCI bridge numbers (4LSb) start with 0.
05h – BMC	Test ID	BMC POST errors. Test IDs (8 bits) are listed in Section B.1.4, “BMC POST” on page 224.
06h – Slow port Ethernet	Test ID	Slow port Ethernet POST errors. Test IDs (8 bits) are listed in Section B.1.5, “Slow Port Ethernet POST” on page 225.
07h – PCI Ethernet	Test ID / Port#	PCI Ethernet POST errors. Test IDs (4 MSb) are listed in Section B.1.6, “PCI Ethernet POST” on page 225. Port numbers (4LSb) start with 0.
08h – ME#0 to 17h – ME#15	Test ID / Thread#	Microengines POST errors. Test IDs (4 MSb) are listed in Section B.1.7, “Microengines POST” on page 225. Thread numbers (4LSb) start with 0.
18h – Slow port	Test ID	Slow port POST errors. Test IDs (8 bits) are listed in Section B.1.5, “Slow Port Ethernet POST” on page 225.
19h – Interrupts	Test ID	Interrupts POST errors. Test IDs (8 bits) are listed in Section B.1.9, “Interrupt POST” on page 226.
1Ah – XScale core	Test ID	XScale core POST errors. Test IDs (8 bits) are listed in Section B.1.10, “XScale Core POST” on page 226.
1Bh – MSF	Test ID	MSF POST errors. Test IDs (8 bits) are listed in Section B.1.11, “MSF POST” on page 226.
1Ch – GPIO	Test ID	GPIO POST errors. Test IDs (8 bits) are listed in Section B.1.12, “General Purpose I/O (GPIO) POST” on page 226.

Table 53. IPMI event data for the POST class (Continued)

Data 2 (Event)	Data 3 (Additional Information)	Description
1Dh – I^2C	Test ID	I^2C POST errors. Test IDs (8 bits) are listed in Section B.1.13, "I2C POST" on page 227.
1Eh – LEDs	Test ID	LEDs POST errors. Test IDs (8 bits) are listed in Section B.1.14, "LED POST" on page 227.
1Fh – Baseboard Media Access	Test ID / Port#	Baseboard Media Access POST errors. Test IDs (4 MSb) are listed in Section B.1.15, "Media Access POST" on page 227. Port numbers (4LSb) start with 0.
20h – FIC Media Access (IXB28504XGBEFSx boards only)	Test ID / Port#	FIC Media Access POST errors. Test IDs (4 MSb) are listed in Section B.1.15, "Media Access POST" on page 227. Port numbers (4 LSb) start with 0.
21h – MMC#1 Media Access – mode 1	Test ID / Port#	Media Access POST errors for MMC#1 working in mode 1 – GE or POS. Test IDs (4 MSb) are listed in Section B.1.15, "Media Access POST" on page 227. Port numbers (4LSb) start with 0. Not applicable to IXB2850.
23h – MMC#2 Media Access – mode 1	Test ID / Port#	Media Access POST errors for MMC#2 working in mode 1 – GE or POS. Test IDs (4 MSb) are listed in Section B.1.15, "Media Access POST" on page 227. Port numbers (4LSb) start with 0.
25h – Baseboard Media Traffic	Test ID / Port#	Baseboard Media Traffic POST errors. Test IDs (4 MSb) are listed in Section B.1.16, "Media Traffic POST" on page 228. Port numbers (4LSb) start with 0.
26h – FIC Media Traffic (IXB28504XGBEFS boards only)	Test ID / Port#	FIC Media Traffic POST errors. Test IDs (4 MSb) are listed in Section B.1.16, "Media Traffic POST" on page 228. Port numbers (4LSb) start with 0.
27h – MMC#1 Media Traffic – mode 1	Test ID / Port#	Media Traffic POST errors for MMC#1 working in mode 1 – GE. Test IDs (4 MSb) are listed in Section B.1.16, "Media Traffic POST" on page 228. Port numbers (4LSb) start with 0. Not applicable to IXB2850.
29h – MMC#2 Media Traffic – mode 1	Test ID / Port#	Media Traffic POST errors for MMC#1 working in mode 1 – GE or POS. Test IDs (4 MSb) are listed in Section B.1.16, "Media Traffic POST" on page 228. Port numbers (4LSb) start with 0.
2Bh – Telecom Clocks	Test ID	Telecom Clocks POST errors. Test IDs (8 bits) are listed in Section B.1.17, "Telecom Clock POST" on page 228.

The Table 53 gives the contents of the Event Data 2 and Event Data 3 fields for the Diagnostics class used by the NPU acting as sensor. These IPMI events messages are generated by the extended diagnostics.

Table 54. IPMI event data for the diagnostics class

Data 2 (Event)	Data 3 (Additional Information)	Description
00h – UART	Test ID / UART#	UART tests errors. Test IDs (4 MSb) and UART numbers are listed in Section C.1.5, "Serial/UART Tests" on page 231.
01h – DRAM	Test ID	DRAM tests errors. Test IDs (8 bits) are listed in Section C.1.4, "Memory Tests" on page 230.
02h – QDR SRAM	Test ID	QDR SRAM tests errors. Test IDs (8 bits) are listed in Section C.1.4, "Memory Tests" on page 230.
04h – PCI	Test ID / Bridge#	PCI POST errors. Test IDs (4 MSb) are listed in Section C.1.6, "PCI Tests" on page 232. PCI bridge numbers (4LSb) start with 0.
05h – BMC	Test ID	BMC tests errors. Test IDs (8 bits) are listed in Section C.1.2, "BMC Tests" on page 230.

Table 54. IPMI event data for the diagnostics class (Continued)

Data 2 (Event)	Data 3 (Additional Information)	Description
06h – Slow port Ethernet	Test ID	Slow port Ethernet tests errors. Test IDs (8 bits) are listed in Section C.1.10, "Slow Port Ethernet Tests" on page 233.
07h – PCI Ethernet	Test ID / Port#	PCI Ethernet tests errors. Test IDs (4 MSb) are listed in Section C.1.11, "PCI Ethernet Tests" on page 233. Port numbers (4LSb) start with 0.
08h – ME#0 to 17h – ME#15	Test ID / Thread#	Microengines tests errors. Test IDs (4 MSb) are listed in Section C.1.14, "Microengine Tests" on page 234. Thread numbers (4LSb) start with 0.
18h – Slow port	Test ID	Slow port tests errors. Test IDs (8 bits) are listed in Section C.1.9, "Slow Port Tests" on page 233.
19h – Interrupts	Test ID	Interrupts tests errors. Test IDs (8 bits) are listed in Section C.1.12, "Interrupt Tests" on page 233.
1Ah – XScale core	Test ID	XScale core tests errors. Test IDs (8 bits) are listed in Section C.1.3, "XScale Core Tests" on page 230.
1Bh – MSF	Test ID	MSF tests errors. Test IDs (8 bits) are listed in Section C.1.13, "MSF Tests" on page 234.
1Ch – GPIO	Test ID	GPIO tests errors. Test IDs (8 bits) are listed in Section C.1.7, "GPIO Tests" on page 232.
1Dh – I^2C	Test ID	I^2C tests errors. Test IDs (8 bits) are listed in Section C.1.8, "I2C Tests" on page 232.
1Eh – LEDs	Test ID	LEDs tests errors. Test IDs (8 bits) are listed in Section C.1.1, "LED Tests" on page 230.
1Fh – Baseboard Media Access	Test ID / Port#	Baseboard Media Access tests errors. Test IDs (4 MSb) are listed in Section C.1.15, "Media Access Tests" on page 234. Port numbers (4LSb) start with 0.
20h – FIC Media Access (IXB28504XGBEFSx boards only)	Test ID / Port#	FIC Media Access tests errors. Test IDs (4 MSb) are listed in Section C.1.15, "Media Access Tests" on page 234. Port numbers (4LSb) start with 0.
21h – MMC#1 Media Access – mode 1	Test ID / Port#	Media Access tests errors for MMC#1 working in mode 1 – GE. Test IDs (4 MSb) are listed in Section C.1.15, "Media Access Tests" on page 234. Port numbers (4LSb) start with 0. Not applicable to IXB2850.
23h – MMC#2 Media Access – mode 1	Test ID / Port#	Media Access tests errors for MMC#1 working in mode 1 – GE or POS. Test IDs (4 MSb) are listed in Section C.1.15, "Media Access Tests" on page 234. Port numbers (4LSb) start with 0.
25h – Baseboard Media Traffic	Test ID / Port#	Baseboard Media Traffic tests errors. Test IDs (4 MSb) are listed in Section C.1.16, "Media Traffic Tests" on page 235. Port numbers (4LSb) start with 0.
26h – FIC Media Traffic (IXB28504XGBEFSx boards only)	Test ID / Port#	FIC Media Traffic tests errors. Test IDs (4 MSb) are listed in Section C.1.16, "Media Traffic Tests" on page 235. Port numbers (4LSb) start with 0.
27h – MMC#1 Media Traffic – mode 1	Test ID / Port#	Media Traffic tests errors for MMC#1 working in mode 1 – GE. Test IDs (4 MSb) are listed in Section C.1.16, "Media Traffic Tests" on page 235. Port numbers (4LSb) start with 0. Not applicable to IXB2850.
29h – MMC#2 Media Traffic – mode 1	Test ID / Port#	Media Traffic tests errors for MMC#1 working in mode 1 – GE or POS. Test IDs (4 MSb) are listed in Section C.1.16, "Media Traffic Tests" on page 235. Port numbers (4LSb) start with 0.
2Bh – Telecom Clocks	Test ID	Telecom Clocks tests errors. Test IDs (8 bits) are listed in Section C.1.17, "Telecom Clock Tests" on page 236.

9.5 Diagnostics

Diagnostics are handled by an additional application separate from the Boot Monitor. The diagnostics are executed from the Command Line Interface (CLI). It is possible to run a single test or the complete diagnostics for the entire baseboard. Some tests can be executed with parameters that allow the user to modify test execution. For example, the user can determine tested memory size or interface speed. Some tests can be repeated a specified number of times or run until Ctrl-C is typed at the console.

Diagnostics consist of detailed and thorough test cases that verify the proper operation of all hardware components on the board. The diagnostics can be loaded and executed from the Boot Monitor. The diagnostics image can be loaded into DRAM from flash, TFTP server or through a serial line.

The diagnostic tests are grouped according to the hardware components they test and under each group, there may be multiple tests. The diagnostic test suite consists of the following:

- Xscale core tests
- LED tests
- QDR RAM tests
- DRAM tests
- UART tests
- PCI bus tests
- Interrupt tests
- GPIO tests
- Slow port tests
- I^2C tests
- Slow port Ethernet tests
- PCI Ethernet tests
- MSF tests
- Microengine tests
- Media tests
- FIC tests (IXB28504XGBEFSx boards only)
- Telecom Clock tests

Every diagnostic test error is reported to the ShMC via an IPMI event message sent to the BMC as described in Section 9.4.7, “IPMI Event Messaging” on page 115.

See Appendix C, “Diagnostics” for more detailed information about each diagnostic test.

9.5.1 Diagnostic Utilities Module

The Diagnostic Utilities module includes the following utilities:

- **Serial Console** - Serial console is used for the command line interface and for reporting status of executed tests.
- **Memory access** - Utilities for accessing flash, SRAM, and DRAM.
- **View System Log** - A utility for viewing the system log that is stored in flash memory.

9.5.2 Diagnostic Test Module

The Test Module consists of various test routines for the different hardware components on the board, such as the XScale core, memory, UART, debug Ethernet, PCI, and microengines, etc. See Appendix C, “Diagnostics” for details of each diagnostic test.

9.6 Linux Board Support Package

The embedded Linux* kernel must be modified to support IXB2850 boards. The modified version is referred to as the Linux Support Package (LSP).

The LSP kernel must support the following IXB2850 hardware-specific components:

- Intel® IXP2850 network processor-specific interrupts
- IXB2850 baseboard-specific interrupts
- IXB2850 board reset
- I^2C bus
- Standard PCI bus
- IXB2850 on-board banked flash memory
- IXP2850 network processor internal UART (serial console)
- Two on-board 16C550 UARTs (debugger console and BMC serial interface)
- On-board CS8900A 10 Mb Ethernet controller (debug interface)
- On-board 82546 Dual Gigabit Ethernet Controller (via PCI - backplane base interface)

The following subsections provides an overview of the LSP kernel drivers required for the above hardware components.

9.6.1 Hardware-Specific Interrupt Support

The LSP kernel provides support for interrupt sources specific to:

- IXP2850 network processor
- IXB2850 baseboard

The IXP2850 network processor interrupts are described in detail in *Intel IXP2800 Network Processor Programmer’s Reference Manual*.

IXB2850 baseboard interrupts are managed through a set of internal CPLD registers accessible through the NP slow port. The interrupt-related CPLD registers support setting interrupt mask, reading masked and unmasked interrupts, and simulating interrupts.

IXB2850 board interrupt sources are:

- 82546 Dual Port Gigabit Ethernet Controller
- 16C550 UARTs
- Media Mezzanine Cards (Quad Gigabit Ethernet Mezzanine Card)
- Board Management Controller (BMC)
- Thermal alarm
- Board installed at PMC site
- Onboard Ethernet PHYs

- Onboard SPI-3/4 Bridge chip

9.6.1.1 Reset Interrupt

IXB2850 boards support the “Soft Reset” startup procedure. The XScale core as well as SRAM and DRAM memory controller are not reset during this kind of boot procedure. To make the Soft Reset possible, the CPLD device responsible for the NPU hardware reset generates the NPU interrupt to allow a software controlled NPU reset. The Linux routine supporting this interrupt should perform the following operation:

- If the Soft Reset flag in the NPU boot parameters stored on flash memory is cleared, disable the reset timer in the CPLD device and perform Soft Reset startup procedure. The CPLD starts this timer along with NPU interrupt generation. If this timer is not disabled (for example, the NPU permanently hangs-up), the NPU is hardware reset after timer expiry.
- Clear XScale core cache memory

Note: Do not enable flash memory aliasing. Typically, the XScale core in the IXP2850 network processor is mapped to address 0xC400 0000 in the memory map. Flash memory aliasing is a network processor feature that allows the XScale core to see flash memory at address 0x0 in the memory map (when DRAM is physically connected). This feature allows the network processor to boot from flash memory.

- Start the Initial Loader from flash memory

9.6.2 IXB2850 Board Reset

IXB2850 board hardware components can be reset by the software through a special CPLD 32-bit reset register. See the *IXP2800 Network Processor Hardware Reference Manual* and the *IXP2800 Network Processor Programmer’s Reference Manual* for details.

By setting individual bits in this register, the LSP kernel can reset:

- Board Management Controller (BMC)
- Media Access Module SPI-3/4 Bridge chip
- Media Access Module Fork FPGA
- On-board 82546 Dual Port Gigabit Ethernet Controller
- On-board Intel® IXF1104 4x1 Gigabit Ethernet MAC
- On-board Marvell* Alaska 1011 Gigabit Ethernet PHY
- On-board CS8900A 10 Mb Ethernet controller
- On-board 16C550 UART controller
- Fabric Interface Card (FIC, IXB28504XGBEFSx boards only)
- Gigabit Ethernet Mezzanine Card in slot DB#2

The board reset procedure takes care of switching to bank 0 of the flash memory during system reset (see Section 9.6.4, “Banked Flash Memory Support” on page 122).

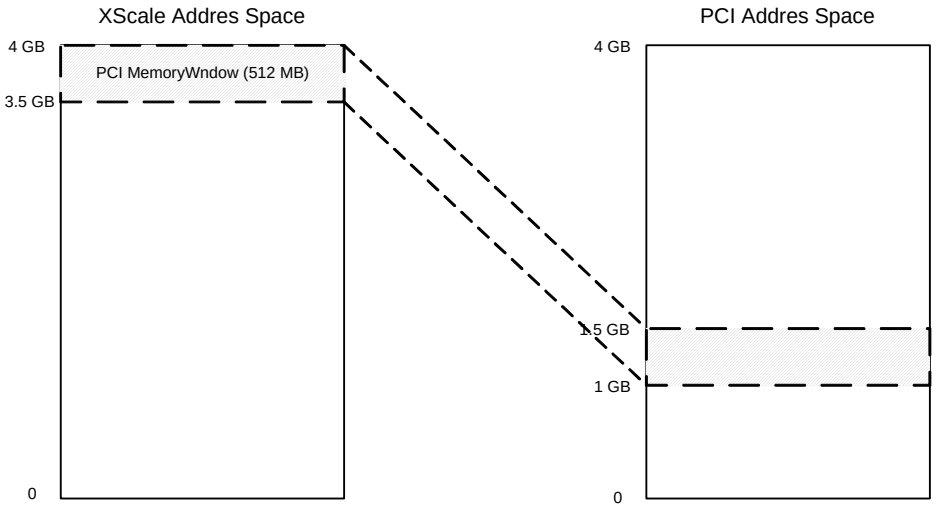
9.6.3 PCI Memory Remapping

The standard PCI memory address space spans 4 GB (32-bit addresses). The XScale core 4 GB physical address space reserves only 512 MB for the PCI memory window. Further addressing is possible by using the PCI_ADDR_EXT register. The Boot Monitor takes care to configure all possible devices in a single 512MB “block”. During initialization, the Linux kernel reads the content of the PCI_ADDR_EXT register and calculates the proper offset between the XScale core physical address and the PCI bus

address. Assuming that the RDRAM size is 768MB, and addresses of the IXP2850 registers and PCI devices belong to the 1 GB to 1.5 GB PCI range, the XScale core PCI memory window has to be remapped to that range. This process is shown in Figure 36.

Note: Using this PCI memory remapping scheme, the NPU can access 512 MB of Adjunct Processor (AP) memory over the PCI at one time.

Figure 36. PCI memory window remapping



During initialization, the PCI driver performs the following functions specific to the IXB2850 board hardware:

- Base PCI address calculation
- Mapping PCI interrupts to slot numbers

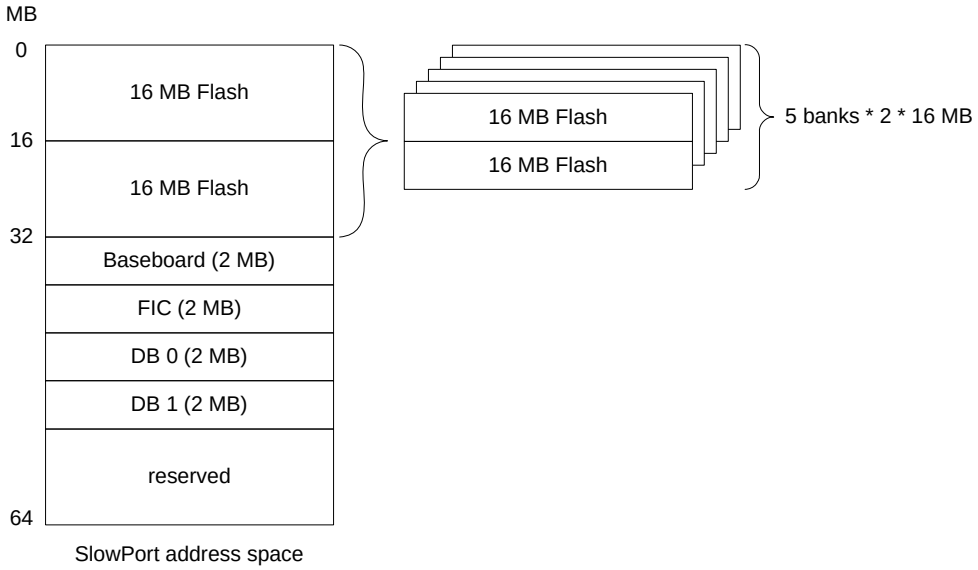
PCI memory remapping and PCI bus scanning and configuration is performed by the Boot Monitor. PCI bus scanning in the LSP kernel is disabled.

9.6.4 Banked Flash Memory Support

IXB2850 boards are equipped with 160 MB of flash memory. To enable access to this memory through the available 32 MB slow port window, it is divided into 5 banks, each comprising two 16 MB chips. The LSP kernel contains an MTD map driver modified to support appropriate bank switching.

Figure 37 shows the concept of flash memory bank switching.

Figure 37. Flash memory bank switching



The MTD map driver allows the use of the various flash file systems provided in the standard kernel (JFFS2, CRAMFS, etc.).

9.6.5 UART Driver

The LSP UART driver supports both the IXP2850 network processor internal serial port and the additional two on-board UARTs (16C550).

The internal serial port is used as the primary Linux console.

The on-board UARTs are used for:

- Secondary Linux console
- BMC (IPMI controller) communication

Due to hardware requirements, the Linux standard serial driver has been modified to support 32-bit data access.

9.6.6 10 Mb Ethernet Driver

The 10 Mb Ethernet Driver is an LSP Ethernet driver that supports the IXB2850 on-board CS8900A Ethernet controller. This Ethernet port is accessed through the slow port.

The I/O functions of a Linux standard Ethernet driver have been modified to:

- Remove PC/ISA-related dependencies
- Support 32-bit access (hardware requirement)
- Provide a workaround for microengine performance throttling during on-board port activity

9.6.7 Gigabit Ethernet Driver

A standard Linux Gigabit Ethernet driver is used for supporting the on-board 82546 Dual Port Gigabit Ethernet Controller, accessible via the PCI bus and supporting the backplane base interface.

Note: RedBoot configures the PHY in GBIC mode and also configures the crosspoint switch. Therefore, when Linux runs, the standard Linux Gigabit Ethernet driver can be used without modification.

9.6.8 Linux Support Package Image Loading

The Boot Monitor, RedBoot, is used as the OS loader for Linux. See Section 9.4, "Boot Monitor" on page 110 and the *RedBoot User's Guide* (see http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html) for further information about the Boot Monitor.

9.6.9 BMC Access

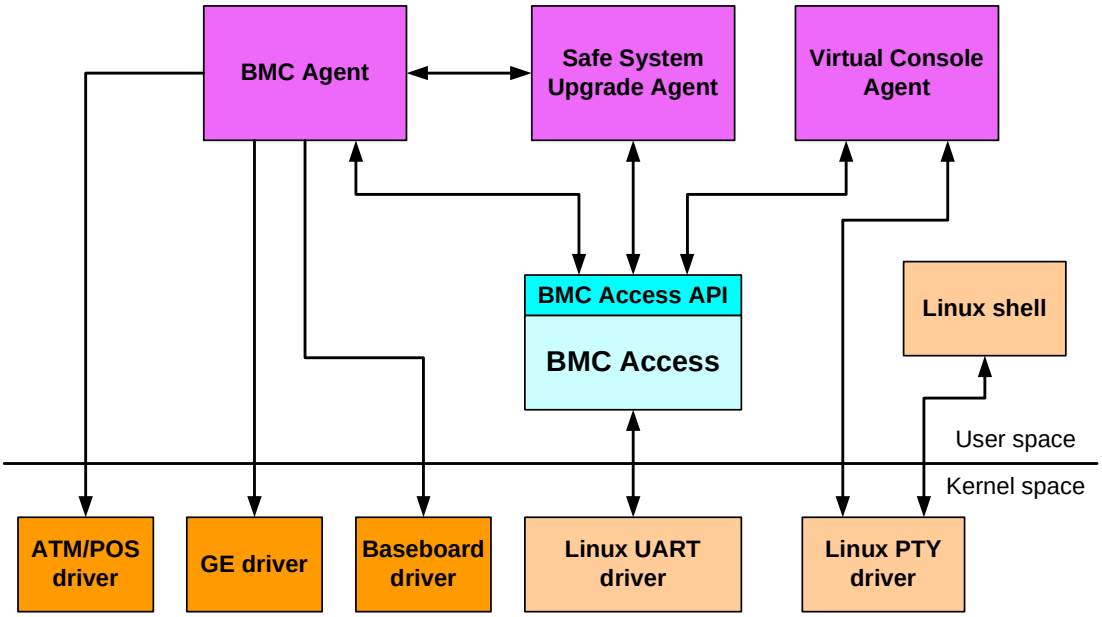
The Linux BMC Access is an application that is responsible for communication between the NPU and the BMC. It uses a UART connection to send and receive IPMI messages between the BMC and ShMC (see Section 8.3.1, "Serial Connections" on page 97 for details).

The BMC Access exposes an API to other Linux application that allows:

- Registration for the reception of IPMI messages based on NetFn/CMD information
- Reception of selected IMPI messages
- Sending responses to received IMPI messages
- Sending IPMI event messages to ShMC

The BMC Access interfaces are shown in Figure 38.

Figure 38. BMC Access interfaces



9.6.9.1 BMC Access API

9.6.9.1.1 registerBMCRequest( )

Name

```
registerBMCRequest (int, int, int, CC_STATUS (*in_fn))
```

Definition

```
CC_STATUS
registerBMCRequest
(
    int     netFn,
    int     cmd,
    int     rsLUN,
    CC_STATUS(*in_fn(unsigned char payload*, int len))
)
```

Description

This routine registers a process in the BMC Access module. BMC Access calls the specified callback function on receiving a message with specified NetFn and CMD.

Parameters

Type	Name	Description
IN	netFn	IPMI net function
IN	cmd	IPMI command
IN	rsLUN	Responder's IPMI Logical unit number
IN	In_fn	Pointer to callback function invoked after a message is received
IN	payload	Data decapsulated from the received IPMI message, that is, the message without the header information.
IN	len	Length of payload

Returns

Return code	Description
CC_OK	operation completed successfully
CC_ERROR	operation error
CC_BUSY	resource already registered by other process

9.6.9.1.2 unregisterBMCRequest( )

Name

```
unregisterBMCRequest (int, int, int)
```

Definition

```
CC_STATUS
unregisterBMCRequest
(
    int netFn,
    int cmd,
    int rsLUN
)
```

Description

This routine unregisters a process in the BMC Access module.

Parameters

Type	Name	Description
IN	netFn	IPMI net function
IN	cmd	IPMI command
IN	rsLUN	Responder's IPMI Logical unit number

Returns

Return Code	Description
CC_OK	operation completed successfully
CC_ERROR	operation error

9.6.9.1.3 sendIpmiMsg( )

Name

```
sendIpmiMsg(char*, int*, int, int, int)
```

Definition

```
CC_STATUS
sendIpmiMsg
(
    char*   pData,
    int*    pLength,
    int     netFn,
    int     cmd,
    int     rqLUN
)
```

Description

This routine sends an IPMI message and returns the payload of the response message.

Parameters

Type	Name	Description
IN/OUT	pLength	payload length in bytes
IN/OUT	pData	pointer to message's payload, that is, the message without IPMI header information. ASCIIZ is not required.
IN	netFn	IPMI net function
IN	cmd	IPMI command
IN	rqLUN	Requester's IPMI Logical unit number

Returns

Return Code	Description
CC_OK	operation completed successfully
CC_ERROR	operation error

9.6.10 BMC Agent

The BMC Agent is a Linux application responsible for:

- Receiving and handling **Set Port State** and **Bus Resource Control** IPMI messages concerning E-Keying functionality.
- Receiving and handling the **Set FRU LED State** IPMI message to force all Media Mezzanine Cards software controllable LEDs ON/OFF to perform a “lamp test”
- Receiving and handling IPMI messages that control the remote Safe System Upgrade process

The BMC Agent uses the BMC Access API to:

- Register for reception of all the above-mentioned IPMI messages
- Receive these messages
- Send IPMI messages to the BMC and the ShMC

9.6.10.1 BMC Agent Startup Activity

The BMC Agent performs the following steps during its startup:

- Sends to the ShMC an IPMI event about NPU readiness (the CPU is treated as a sensor described by a dedicated SDR). This event message indicates to the BMC and ShMC that the NPU is able to handle IPMI messages
- Retrieves from the BMC the E-Keying Status Word and sets the AdvancedTCA backplane accordingly
- Disable the BMC watchdog supervising the Linux startup process (send **Set Processor Watchdog** command to BMC)

9.6.10.2 E-Keying

The BMC Agent supports E-Keying functionality by setting the appropriated state of the backplane interfaces. It uses the following drivers API for this purpose:

- Gigabit Ethernet driver, to set the state of the Gigabit Ethernet PHY ports connected to the AdvancedTCA base interface
- Baseboard driver, to set state of crosspoint switch ports connected to the AdvancedTCA fabric interface
- Clock driver, to set state of Zarlink* buses connected to the AdvancedTCA Clock Interface

9.6.10.3 LED handling

The BMC Agent uses the baseboard driver’s LED management API to force all MMC LEDs ON/OFF to support “lamp test” functionality.

9.6.11 Safe System Upgrade Agent

The SSU Agent provides the framework for Safe System Upgrade functionality. This framework allows remote upgrade of the following IXB2850 board firmware:

- Boot Monitor
- Linux kernel
- Linux root file system
- Linux target application
- BMC firmware

9.7 Baseboard Driver

The LSP baseboard driver is a special firmware driver covering all system components not controlled by the media drivers, in particular:

- SPI-3/4 Bridge
- SPI3/UTOPIA Bus Fork FPGA
- Configuration of crosspoint switch responsible for connecting baseboard and FIC Gigabit Ethernet MAC (IXF1104) to the fabric interface channels

The driver provides access to the hardware through its API functions (the preferred way) or IOCTL (I/O Control) functions.

Basic functions of the driver include:

- Fork FPGA image loading
- SPI-3/4 Bridge configuration
- Crosspoint switch configuration
- MSF and data path configuration
- LED support (including control of LEDs on MICs, mezzanine cards and NP module)
- Detection of device types inserted into the baseboard
- Handling "**get FRU parameter**" and "**get slot number**" requests
- Telecommunication Clocks (Zarlink) configuration

The driver is initially set up using the command line. It configures and manages baseboard components through various registers accessible through the NP slow port.

The baseboard driver must be loaded and initialized before the media drivers, because the baseboard driver performs system media configuration functions (for example, MSF initialization, SPI lane routing) that the other drivers require to operate.

Note: Baseboard driver initialization does not reset connections between the base interface, PHY, crosspoint switch or 82546 Dual Port Gigabit Ethernet Controller devices on the board.

The baseboard driver uses information that is maintained by the board management controller (BMC). Under Linux, the driver does not need IPMI access to the BMC since all the values required to start the board are read by the Boot Monitor and stored in EEPROM before the driver starts.

9.7.1 Configuration Settings

The Baseboard driver initialization parameters in Table 55 must be set up during driver initialization. For Linux systems, this is done in the command line with **insmod**.

See Figure 8, "Media Access Module block diagram" on page 28 for elements that are referenced in the baseboard driver initialization parameters.

Table 55. Baseboard driver initialization parameters

Parameter	Values	Description	Valid Values
dev0	Mode can be: • spi3 (GbE) Default = spi3 Parity can be: • odd • even • none Default = odd	**dev0** is DB#1 (unused) Mode and parity values are separated by a colon.	dev0=spi3:odd
dev1	Mode can be: • spi3 (GbE) Default = spi3 Parity can be: • odd • even • none Default = odd	**dev1** is DB2#2 (the Quad Gigabit Ethernet Mezzanine Card) Mode and parity values are separated by a colon	dev1=spi3:odd
dev2	Mode can be: • spi3 (GbE) Default = spi3 Parity can be: • odd Default = odd	**dev2** is the IXF1104 MAC Device on the baseboard or the Fabric Interface Card Mode and parity values are separated by a colon	dev2=spi3:odd
noFabricPrepend (in firmware version 1B2/1B1G or later)	Mode can be: • 0 • 1 Default = 0	**noFabricPrepend** disables the use of the 64-bit prefix used to distinguish between packets coming from/going to the baseboard IXF1104 or FIC, as explained in Section 3.3.1, "Ethernet Packet Formats Over the Backplane" on page 31. By default, the use of the 64-bit prefix is enabled. Specifying **noFabricPrepend**=1 disables the use of the prefix.	**noFabricPrepend**=1

Note: Gigabit Ethernet devices require the parity setting to be synchronized with the software. The current default setting for the Gigabit Ethernet driver requires odd parity, but the parity can be changed within the Gigabit Ethernet driver. You must set the same parity for both drivers.

Note: Quad Gigabit Ethernet device mode must be SPI-3.

10.0 Using IXB2850 Boards with the IXA SDK

10.1 Introduction

IXB2850 boards can be used with the Intel® Internet Exchange Architecture Software Development Kit (Intel® IXA SDK) to enable application development and in-depth testing to validate data paths, device functionality, system functionality, device driver behavior and the Linux Support Package (LSP) software. For more information about the IXA SDK, see the *Intel® Internet Exchange Architecture Software Development Kit (IXA SDK) Software Framework Getting Started Guide*.

10.2 IXA SDK Support

Refer to the IXB2850 board software *Release Notes* for the version of the IXA SDK supported by the latest board software.

10.2.1 IXA SDK Patch

An IXA SDK patch must be installed before any development using the IXA SDK can occur. The IXA SDK patch image can be downloaded from the board support web site. See Section 1.1, “Product Description” on page 9 for a pointer to the product web site.

Refer to the README file that accompanies the patch for installation instructions. If necessary, contact your Intel support representative for assistance.

10.3 Supported IXA SDK Application

IXB2850 boards support an IPv4 Forwarding application (called the IXDP2801 IPv4 4xOC12 POS / 6xGB Ethernet Forwarding Application).

Note: The primary purpose of this application is to validate that the IXA SDK Framework (RM, System Application, etc.) is working correctly on IXB2850 boards.

10.3.1 IPv4 Forwarding Application Patch

A patch must be applied before the IPv4 Forwarding application can be used. The patch can be downloaded from the board support web site. See Section 1.1, “Product Description” on page 9 for a pointer to the product web site.

Refer to the README file that accompanies the patch for installation instructions. If necessary, contact your Intel support representative for assistance.

11.0 Maintenance

11.1 Firmware Upgrade

On IXB2850 boards, Safe System Upgrade (SSU) controls the board software upgrade process (that is, upgrade from a local repository). SSU comprises two components:

- Safe System Upgrade Agent (SSUA) - Provided as part of the board firmware.
- Safe System Upgrade Manager (SSUM) - A customer application that can be developed using OEM SSU IPMI messages (see Section 11.4.1, “Safe System Upgrade IPMI Commands” on page 152).

SSU provides the following functionality:

- Different upgrade types:
  - — Selective upgrade:
    - Boot Monitor only (including FPGA image)
    - Linux kernel only
    - Linux file system
    - Target application image only – microcode, core components, proprietary drivers
    - BMC image
  - — Full upgrade – all upgradeable images can be updated at once
- Local and remote upgrade types – local upgrade is performed in the Boot Monitor, it requires operator intervention and the image is loaded via the debug interface or serial console. Remote upgrade is performed remotely using the IPMI protocol, without any operator intervention near the system.
- Automatic Rollback upon upgrade failure – each upgradeable image is validated by CP confirmation request. In the case of a board reboot without this request, automatic restart of the previous stable software configuration is performed.

11.2 Safe System Upgrade

Safe System Upgrade is functionality responsible for updating images stored in onboard flash memory. The upgrade process is supervised and the results must be confirmed by the management module, the SSUM. If the software upgrade fails or is not confirmed, the SSU undoes all modifications (restoring the previous stable software version of the board) and the system state remains unchanged – this operation is called rollback.

11.2.1 Role and Functionality

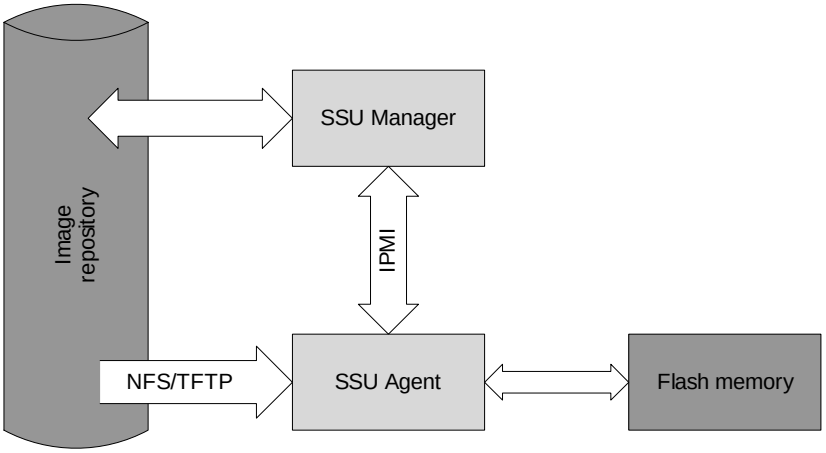
SSU provides the functionality to upgrade images stored on flash (Boot Monitor, Linux kernel, target image etc.). Note that all images can be upgraded at once or the upgrade process may be performed for the selected image.

The SSU module includes two main elements (see Figure 39):

- SSU Agent (SSUA)

- SSU Manager (SSUM)

Figure 39. Safe System Upgrade architecture overview



The SSU Agent (SSUA) controls SSU functionality at the board level. The SSUA:

- Monitors and controls all SSU operations on the board (including BMC upgrade)
- Upgrades software on the flash memory - stores new software images on the NPU or BMC flash memory.
- On SSUM request, the SSUA reads and sends properties of the software stored in flash memory and running on the board (including the querying of the BMC for image properties on its local flash).

The SSU Manager (SSUM) is responsible for the SSU functionality at the chassis/system level. SSUM receives, dispatches and handles SSU requests from the operator, transfers image properties to the board and handles the status of the SSU operations on board.

The image repository is a storage area that is accessible to the SSUA and SSUM. The image repository contains a set of images that can be used for upgrading board software and for booting.

SSU functionality offers two methods of image upgrade, remote and local:

- Local image upgrade enables manual image load to local memory using the TFTP or Xmodem utilities. After the image is loaded, the operator flashes the image using a simple commands. It is important that the correct image type (operational code and accompanying header) are use. The SSU ensures that the loaded image is written into the appropriate area of flash. In the case of a local image upgrade, the Boot Monitor console is used, for both NPU and BMC images.
- Remote image(s) upgrade is performed using the IPMI protocol, images are loaded into local memory using the NFS or TFTP utilities, then the image is written to flash.

11.2.2 SSUA Operation

11.2.2.1 Dual Image Support

In order to support the automatic rollback mechanism in case of image upgrade failure (invalid image, part of image hangs, kernel fatal error during initialization, target image does not work properly, etc.), a dual image concept must be supported. The concept is based on the following assumptions:

- For each image of flash that is subject to SSU functionality, there must be a second image storage. One image is active, the second is intended for writing a new image.
- There is a well-known location (I^2C EEPROM) for storing and updating the status of the upgrade operation. This location is also used for selecting the image to be executed during system startup.

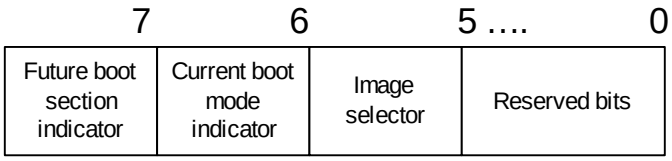
Note: To start the kernel with the appropriate file system, one of two Boot Monitor start scripts is selected. The contents of this script are modified by the SSUA.

- In the case of BMC upgrade, the status of the upgrade operation is stored on NPU I^2C EEPROM and on BMC local flash (this is for booting reasons, that is, the BMC Initial Loader operates before the power is present on board).
- Image validation after upgrade takes place only on SSUM request. That is, the SSUA marks the upgraded image as a valid (and active) image only after receiving a confirmation from the SSUM.

11.2.2.2 Image Upgrade and Automatic Rollback

To enable the SSU to provide Automatic Rollback, the status word shown in Figure 40 has been implemented.

Figure 40. SSU status word definition



Status words are maintained for:

- Boot Monitor
- BMC
- Kernel startup script (executed by the Boot Monitor and containing information about the Linux kernel, file system, and target image)

Table 56. Boot monitor image status word

Bit	Name	Description
7	Future image switchover indicator	This bit is set by the SSUA to force the Initial Loader to load the second image in safe mode. If bit is set to 1, the Initial Loader clears this bit and loads the alternative image to the one specified by the “Image selector”.
6	Current boot mode indicator	This bit indicates that current the Boot Monitor image is started in safe mode; used only as an indication of the boot status and set by the Initial loader immediately after clearing bit 7.
5	Image selector	This bit selects the active image to be loaded (either the first or the second copy). This bit may be modified only by the SSUA on a control plane validation request.
4 to 0	Reserved for future use	Unused

Table 57. Boot monitor config image status word (including kernel startup script)

Bit	Name	Description
7	Future Image switchover indicator	This bit is set by the SSUA to force the Boot Monitor to use the second copy of the configuration script in safe mode. If bit is set to 1, the Boot Monitor clears this bit and loads the alternative configuration record to the one specified by the “Image selector”.
6	Current boot mode indicator	This bit indicates that the current Boot Monitor Config (which specifies the kernel and/or root filesystem) is started in safe mode; used only as an indication of boot status and set by the Boot Monitor immediately after clearing bit 7.
5	Image selector	This bit selects the active configuration record to be loaded (either the first or second copy). This bit may be modified only by the SSUA on a control plane validation request.
4 to 0	Reserved for future use	Unused

A description of the image status word for the BMC can be found in Section 11.5.1, “Image Upgrade on BMC” on page 158.

Figure 41 shows the states of each image status word.

Figure 41. SSU status word states

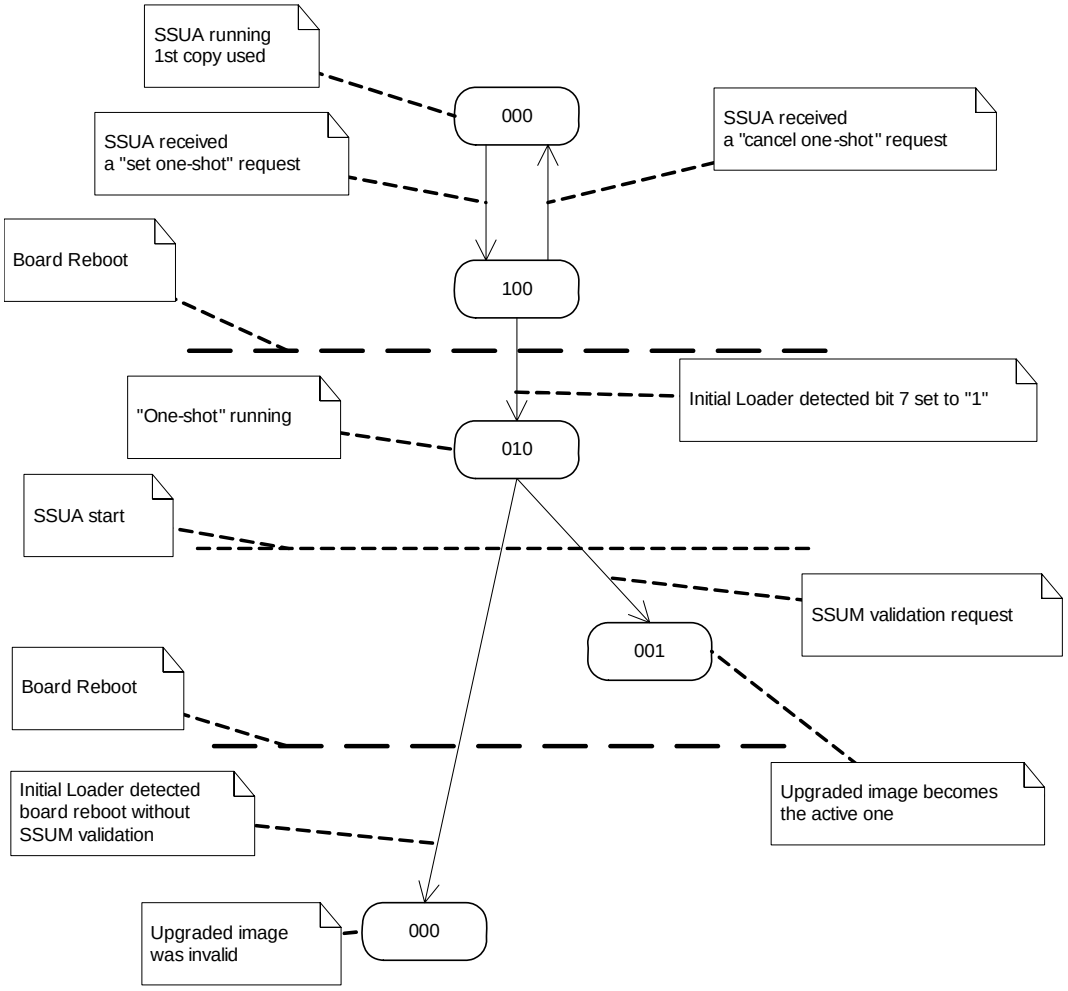
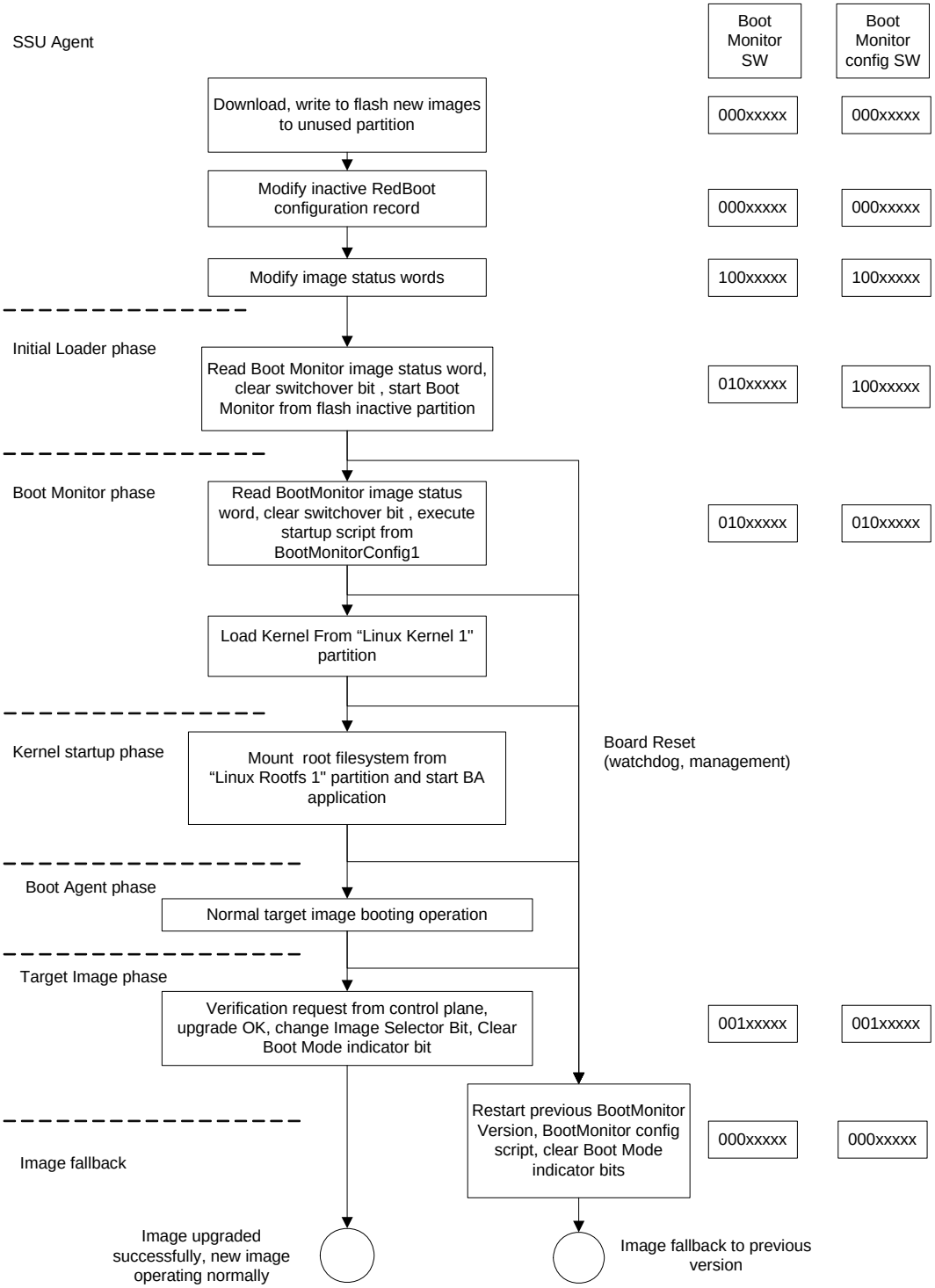


Figure 42 presents the Boot Monitor image upgrade procedure, upgrade failure, and automatic rollback operations.

Figure 42. Boot Monitor upgrade and automatic rollback operations



To update an image stored in flash, the following operations are performed:

- **Download new image**
  Downloading the image to the board is covered either by the NFS client or by the TFTP utility. The image is downloaded into NPU RAM also in the case of the BMC upgrade process.
- **Check if the loaded image is valid**
  The upgrade procedure requires an image file to be verified before writing it to flash. Verification is based on image header validation. The image descriptor is transferred to the board together with the image itself and provides all the necessary information for verifying image type and integrity.
- **Erase flash partition**
  Prior to flashing a new image, the SSUA must clear the selected storage on the flash partitions. The erase operation removes the image and image header file permanently. It cannot be undone.
- **Flash (store) new image**
  Image flashing is the longest part of the upgrade operation. During that operation, the SSUA stores a new image in the selected flash partition and verifies if the stored image matches the original one. Image storing is a simple operation of writing to the MTD device. Image verification performed by the SSUA is an operation of reading from the MTD device combined with checksum calculation.

11.2.2.3 Image Configuration (Switch Images)

An image switchover request is used to run the image from an inactive partition. It is mostly used after the image flash update process to test and validate the new image, but can be also be issued to switch between two images of the same type (active and inactive), for example, to test the alternate image.

This request causes the inactive image (or set of images) to start after the next board reboot. To make this state permanent, the user has to send a validation request. If there is no validation request, on the following reboot, the previous set of images are automatically restored.

11.2.2.4 Image Validation

Board reboot after image switchover request start the system in a so-called “safe mode” or “one-shot-test mode”. If the image is operational, the CP sends a validation request that marks the upgraded image as active.

Restart without control plane confirmation of the new image results in a restart the uses the previous image.

11.2.2.5 Automatic Rollback

The SSUA supports automatic rollback operation. A SSUM may request to switchover to an inactive image, and if after the next reboot there is no validation request (either because operator does not want to validate this type image or the image is not operational), after the next reboot, the previous stable version of image is automatically restored.

11.2.2.6 Read Image Version and Type

It is possible to read from the SSUM the attributes of each image stored in flash. The SSUA reads this information from the image header that has been verified and added after the image update. It is possible to retrieve an image header for one image or a group of images.

11.2.2.7 SSUA Initialization

An SSUA startup script is added to the data plane initialization scripts.

11.3 Local Software Upgrade

It is possible to upgrade software components locally via RedBoot*. This type of upgrade can be performed via the debug Ethernet port (using TFTP) or via the serial console (using Xmodem) and requires operator intervention.

To make this operation as simple as possible, it is performed in the Boot Monitor (the booting procedure has to be stopped on the Boot Monitor's command line interface). To perform a single image upgrade (note that only this kind of update is possible when upgrading locally), the operator has to perform two operations:

- Load the selected image to local RAM
- Issue a command to write this image into flash

Since writing the image to flash can be performed in a number of ways in the Boot Monitor, it is very important that this operation does not disrupt any of the SSUA functionality. To ensure that local software upgrade operations are transparent to the SSU functionality, some dedicated commands have been introduced:

- **ssu upgrade -** Selects image storage by checking the status word for the selected image
- **ssu get info -** Returns properties of the selected image type
- **ssu setcfg -** Runs the upgraded image in safe mode after the next board reboot
- **ssu validate -** Validates the configuration running in safe mode

Local software upgrade of the BMC image is supported by the NPU. The same upgrade commands are used to load the image and start the upgrade process for the BMC code. After loading a BMC image to NPU RAM, communication with the BMC is started and the image is passed to the BMC controller local memory. The BMC operational code selects and writes the image to the inactive partition on its flash. This is possible because there is a local copy of the image status words present on the BMC. After successful storage of the image, the operator has to issue a "switch image" command to run this image in one-shot mode after the next reboot.

11.3.1 Local Software Upgrade Procedure

This procedure covers the upgrade of the firmware on IXB2850 boards that contain an earlier version of the firmware. Check the software Release Notes for a list of the components to be upgraded for a specific release.

The procedure relies on the Local SSU mechanism (as described above in Section 11.3). The TFTP protocol is used to upload files to the board, therefore the debug network interface should be operational and configured.

Note: Substitute the appropriate XModem load commands when upgrading via the serial line if preferred by the system setup.

The procedure for upgrading from earlier versions of the firmware is basically the same, but the components to be upgraded are slightly different. Table 58 and Table 59 show the upgradable components and the order in which they should be upgraded for earlier generation and newer boards respectively.

Table 58. Components to be upgraded for earlier generation boards[1]

No.	Component (Name)	Hardware P/N (Spin)	Image File Locations
1	Bootloader	N/A	Firmware/IXB28x0/RedBoot/bootmonitor/bin/redboot.ssu
2	Recovery Loader	N/A	Firmware/IXB28x0/RedBoot/bootmonitor/bin/appimage_loader.bin
3	Diagnostics	N/A	Firmware/IXB28x0/RedBoot/ixdp2801/bin/diag_ixdp2801_ssu.bin
4	BMC Firmware	N/A	Firmware/IXB28x0/BMC/LION_ssu.bin
5	BMC Starter	N/A	Firmware/IXB28x0/BMC/starter.bin
6	NPU (IXNPM2800)	95-001-05 (3A)	**FRU**: Firmware/IXB28x0/BMC/ixnpm2800_3a_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixnpm2800_3a_sdr.bin
7	Baseboard (IXMB2801)	90-002-05 (3A)	**FRU**: Firmware/IXB28x0/BMC/ixmb2801_3a_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixmb2801_3a_sdr.bin
		90-002-07 (4)	**FRU**: Firmware/IXB28x0/BMC/ixmb2801_4_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixmb2801_4_sdr.bin
8	FIC (FIC_V2)	94-002-01 (1)	**FRU**: Firmware/IXB28x0/BMC/FIC_V2_1_fru.bin **SDR**: Firmware/IXB28x0/BMC/FIC_V2_1_sdr.bin
9	Gigabit Ethernet Mezzanine Card (IXD4GEA1F-M)	91-004-04 (1A)	**FRU**: Firmware/IXB28x0/BMC/ixd4gea1f_M_1a_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixd4gea1f_M_1a_sdr.bin
10	MIC, Fiber (MIC-F4XGB MEZZ)	93-002-01 (1)	**FRU**: Firmware/IXB28x0/BMC/mic_f4gb_mezz_1_fru.bin **SDR**: Firmware/IXB28x0/BMC/mic_f4gb_mezz_1_sdr.bin
		93-002-02 (2)	**FRU**: Firmware/IXB28x0/BMC/mic_f4gb_v2_mezz_1_fru.bin **SDR**: Firmware/IXB28x0/BMC/mic_f4gb_mezz_1_sdr.bin
---	NPU [2] (IXNPM2800)	95-001-05 (3A)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixnpm2800_3A.bin
11	Baseboard [3] (IXMB2801)	90-002-05 (3A)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixmb2801_3A.bin
		90-002-07 (4)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixmb2801_4.bin
---	FIC [2] (FIC_V2)	94-002-01 (1)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixdfic2800_1.bin
12	Gigabit Ethernet Mezzanine Card (IXD4GEA1F-M)	91-004-04 (1A)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixd4get0c_1A.bin
13	MIC, Fiber (MIC-F4XGB MEZZ)	93-002-01 (1)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixd4get0f_mic_1.bin
		93-002-02 (2)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixd4get0f_v2_mic_1.bin

Notes:

Use the cfg read boot monitor console command to check hardware versions.

[1] "Earlier generation boards" include IXB28504XGBEFS boards, which may contain restricted substances.

[2] Shown here for completeness only; upgrade not always required; check the software Release Notes for a list of components to upgrade.

[3] Required when upgrading from version 1A3 only.

Table 59. Components to be upgraded for newer boards[1]

No.	Component (Name)	Hardware P/N (Spin)	Image File Locations
1	Bootloader	N/A	Firmware/IXB28x0/RedBoot/bootmonitor/bin/redboot.ssu
2	Recovery Loader	N/A	Firmware/IXB28x0/RedBoot/bootmonitor/bin/appimage_loader.bin
3	Diagnostics	N/A	Firmware/IXB28x0/RedBoot/ixdp2801/bin/diag_ixdp2801_ssu.bin
4	BMC Firmware	N/A	Firmware/IXB28x0/BMC/LION_ssu.bin
5	BMC Starter	N/A	Firmware/IXB28x0/BMC/starter.bin
6	NPU (IXNPM2800)	95-001-07 (3A)	**FRU**: Firmware/IXB28x0/BMC/ixnpm2800_3a_rohs5_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixnpm2800_3a_rohs5_sdr.bin
7	Baseboard (IXMB2801)	90-002-08 (4)	**FRU**: Firmware/IXB28x0/BMC/ixmb2801_4_rohs5_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixmb2801_4_rohs5_sdr.bin
8	FIC (FIC_V2)	94-002-02 (1)	**FRU**: Firmware/IXB28x0/BMC/FIC_V2_1_rohs5_fru.bin **SDR**: Firmware/IXB28x0/BMC/FIC_V2_1_rohs5_sdr.bin
9	Gigabit Ethernet Mezzanine Card (IXD4GEA1F-M)	91-004-05 (1A)	**FRU**: Firmware/IXB28x0/BMC/ixd4gea1f_M_1a_rohs5_fru.bin **SDR**: Firmware/IXB28x0/BMC/ixd4gea1f_M_1a_rohs5_sdr.bin
10	MIC, Fiber (MIC-F4XGB MEZZ)	93-002-03 (2)	**FRU**: Firmware/IXB28x0/BMC/mic_f4gb_mezz_2_rohs6_fru.bin **SDR**: Firmware/IXB28x0/BMC/mic_f4gb_mezz_2_rohs6_sdr.bin
---	NPU [2] (IXNPM2800)	95-001-07 (3A)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixnpm2800_3A_rohs5.bin
11	Baseboard [2] (IXMB2801)	90-002-08 (4)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixmb2801_4_rohs5.bin
---	FIC [2] (FIC_V2)	94-002-02 (1)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixdfic2800_1_rohs5.bin
12	Gigabit Ethernet Mezzanine Card (IXD4GEA1F-M)	91-004-05 (1A)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixd4get0c_1A_rohs5.bin
13	MIC, Fiber (MIC-F4XGB MEZZ)	93-002-03 (2)	**CPLD**: Firmware/IXB28x0/RedBoot/cpld_images_2801_v2/bin/ixd4get0f_mic_2_rohs6.bin

Notes:
Use the cfg read boot monitor console command to check hardware versions.
[1] "Newer boards" include the IXB28504XGBEFSW and IXB28504XGBEFSR, which are lead-free to second level interconnect.
[2] Shown here for completeness only; upgrade not always required; check the software Release Notes for a list of components to upgrade.

For the purpose of this document, the files are assumed to be present in the `/tftpdir/<release_version>` directory, where `<release_version>` is the actual version of the release, for example "SRA_2.1".

The following topics provide more information on upgrading specific components:

- Upgrading RedBoot
- Upgrading the Recovery Loader
- Upgrading Diagnostics
- Upgrading the BMC Firmware
- Upgrading the BMC Starter
- Upgrading FRU Information and SDRs
- Upgrading CPLDs

11.3.1.1 Upgrading RedBoot

1. Download the RedBoot image to RAM using the command:

```
RedBoot> load -r -v -m TFTP -b 0x3000000 /tftpdir/<release_version>/
redboot.ssu
-
Raw file loaded 0x03000000-0x03300000
```

2. Upgrade the flash partition:

```
RedBoot> ssu upgrade -b 0x03000000 -l 0x300000
... Check unlock from 0xc4020000-0xc4320000:
 * CAUTION * one or more flash sectors locked
... Unlock from 0xc4020000-0xc4320000: ........................
... Erase from 0xc4020000-0xc4320000: ........................
... Program from 0x03000000-0x03300000 at 0xc4020000: ..............
... Unlock from 0xcdfe0000-0xce000000: .
... Erase from 0xcdfe0000-0xce000000: .
... Program from 0x2ff9f000-0x2ffbf000 at 0xcdfe0000: .
... Lock from 0xcdfe0000-0xce000000:
```

3. Switch the active RedBoot:

```
RedBoot> ssu oneshot -i bootmonitor
Safe mode for image : "RedBoot" of type <<bootmonitor>> activated !

RedBoot> reset np
```

4. Validate the new RedBoot image:

```
RedBoot> ssu validate -i  bootmonitor
Image "RedBoot" of type <<bootmonitor>> passed SSU validation !
```

5. Repeat steps 1 to 4 to upgrade the second image.

In the event of failure, see Section 11.3.1.1.1, “Troubleshooting RedBoot Upgrade Problems” on page 141 following.

11.3.1.1.1 Troubleshooting RedBoot Upgrade Problems

In case of problems with upgrading the RedBoot using the SSU mechanism, one of the following procedures can be used:

When one Redboot image is operational:

1. Check the console log to determine which RedBoot partition is active. The information is displayed during loader startup, for example:

```
Recovery Loader version 1.2.0.2 (build AI2800_03_03_11_018_GLC)
Starting regular RedBoot version
Watchdog armed for 250ms
Executing boot monitor from image: RedBoot
```

2. Download the RedBoot image to RAM using the command:

```
RedBoot> load -r -v -m TFTP -b 0x3000000 /tftpdir/<release_version>/
redboot.ssu
-
Raw file loaded 0x03000000-0x03300000
```

3. Manually flash the partition corresponding to the inactive one (RedBoot or _RedBoot):

```
RedBoot> fis create -b 0x03000000 -l 0x300000 _RedBoot
```

4. Activate the new partition and reset the blade:

```
RedBoot> trb
Active boot monitor image have just been switched to: "_RedBoot"
RedBoot> reset
```

5. Repeat steps 1 to 4 above for the second partition.

When both Redboot images are **not** operational:

1. Copy the `appimage_redboot.bin` file (available in the firmware package in the `ixa_sdk_4.1\Firmware\IXB28x0\RedBoot\bootmonitor\bin\` directory) to a convenient location.
2. At board startup, enter the Recovery Loader by typing **==1** at the board startup.
3. Type `load -r -m xmodem -b 0x3000000`.
4. Type `go`.
5. At the `Appimage>` prompt, type `fis image`.
6. Reset the board and follow the steps in the “When one Redboot image is operational” procedure above.

11.3.1.2 Upgrading the Recovery Loader

1. Download the appimage loader with the command:

```
RedBoot> load -r -v -m TFTP -b 0x3000000 /tftpdir/<release_version>/
         appimage_loader.bin
-
Raw file loaded 0x03000000-0x0303810c
```

2. Execute the loader:

```
RedBoot> go
```

```
Booting from BM
...
Appimage>
```

3. Upgrade the Recovery Loader:

```
Appimage> fis image
Your action requires changes to BootMonitor blocks
Continue - are you sure (y/n)? y
* CAUTION * about to program FLASH
              at 0xc4000000..0xc401ffff from 0x0001cebc - are you sure (y/n)? y
... Unlock from 0xc4000000-0xc4020000: .
... Check unlock from 0xc4000000-0xc4020000: .
... Erase from 0xc4000000-0xc4020000: .
... Program from 0x0001cebc-0x0003cebc at 0xc4000000: .
... Lock from 0xc4000000-0xc4020000: .
```

4. Reset the NPU:

```
Appimage> reset
Recovery Loader version 1.2.0.2 (build AI2800_03_03_11_018_GLC)
...
```

11.3.1.3 Upgrading Diagnostics

1. Download the Diagnostics image to RAM using the command:

```
RedBoot> load -r -v -m TFTP -b 0x1000000 /tftpdir/<release_version>/
diag_ixdp2801_ssu.bin
-
Raw file loaded 0x01000000-0x012E0000
```

2. Manually flash the partition:

```
RedBoot> fis create -b 0x01000000 -l 0x2E0000 diag
```

Note: If the partition already exists but has a different size or base address, it should be deleted (using the fis delete command) prior to issuing the fis create command.

11.3.1.4 Upgrading the BMC Firmware

1. Download the BMC image to RAM using the command:

```
RedBoot> load -r -v -m TFTP -b 0x3000000 /tftpboot/<release_version>/
LION_ssu.bin
-
Raw file loaded 0x03000000-0x030 40000
```

2. Upgrade the BMC image (backup copy) using the following command. Notice that image start address and image length (in bold text) come from the load status message displayed in step 1.

```
RedBoot> ssu upgrade -b 0x03000000 -l 0x40000
-
.BMC image updated.
```

Note: This operation takes approximately 10 minutes.

3. Activate the new BMC image as follows:

```
RedBoot> ssu oneshot -i bmc
Waiting for SSU info from BMC ......
Safe mode for image : "bmc" of type <<bmc image>> activated !

RedBoot> reset bmc
BMC reset succeeded !

RedBoot> ssu validate -i bmc
Waiting for SSU info from BMC ......
Image "bmc" of type <<bmc image>> passed SSU validation !
```

4. Repeat steps 1 to 3 to upgrade the second BMC image.

11.3.1.5 Upgrading the BMC Starter

Note: The BMC loader is necessary for SSU upgrade if the BMC firmware is not operational. The BMC loader can be loaded only from BMC console.

1. Using a terminal emulation program, for example `minicom`, connect to the BMC console. The following text is displayed:

```
Welcome to minicom 2.00.0

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n
Compiled on Jan 25 2003, 00:15:18.

Press CTRL-A Z for help on special keys
```

2. On the BMC console, press the Enter key. If the main menu is not displayed, press **0** until the main menu is displayed as follows:

```
BUILD Date:Nov 09 2006 Time:12:49:23
BMC Dual Image ver: 03.03.00.021 (RELEASE) Addr: 0x88 Slot: 4
Board Type: IXMB28X1 Spin3A or Spin4; CPLD version: 9
IMAGE_1 is active
3: burn-in FRUs menu
9: burn-in SDRs menu
l: Start simple CLI console
p: display HW Config
r: IXP (Network processor) reset
R: IXP (Network processor) soft reboot
s: BMC reset!
t: test 1 menu
w: service menu 1 (power management, dual image & registers)
z: service menu 2 (general)
q: CPLD (load/update/version) menu
+: validate image running in safe mode
```

3. On the BMC console main menu, press **z** (to enter Service Menu 2), then **Y** to acknowledge the warning and confirm:

```
You are going to Service 2 Menu. Inapropriate use commands from this menu can
cause board DAMAGE. Are you sure (Y/N)?
BMC Service 2 menu:
1: load image using xmodem
2: Burn-in BOOT part of flash with loaded data
0: Exit this menu
```

4. On Service Menu 2, press **1** to load the image using xmodem. The following text is displayed:

```
Start loading...
```

5. Using your terminal, choose the file to upload with the xmodem protocol (**Ctrl+A S** in minicom).

```
|--[Upload--]
|  zmodem    |
|  ymodem    |
|  xmodem    |
|  kermit    |
|  ascii     |
-------------
```

6. On Service Menu 2, press **2** (to upgrade the image to flash), then **Y** to acknowledge the warning and proceed with the operation. The following messages are displayed:

```
Burn-in code to BOOT part after code download (command 1). Are you sure (Y/N)?Y
Erasing BOOT code part of flash...
Burning BOOT code to flash...
Burning finished successfully !
```

7. Press **0** to return to the main menu.

```
BUILD Date:Nov 09 2006 Time:12:49:23
BMC Dual Image ver: 03.03.00.021 (RELEASE) Addr: 0x88 Slot: 4
Board Type: IXMB28X1 Spin3A or Spin4; CPLD version: 9
IMAGE_1 is active
3: burn-in FRUs menu
9: burn-in SDRs menu
l: Start simple CLI console
p: display HW Config
r: IXP (Network processor) reset
R: IXP (Network processor) soft reboot
s: BMC reset!
t: test 1 menu
w: service menu 1 (power management, dual image & registers)
z: service menu 2 (general)
q: CPLD (load/update/version) menu
+: validate image running in safe mode
```

8. Press **s** to reset the BMC, then press **Y** to confirm:

```
    Reset BMC. Are you sure (Y/N)?
```

9. After restart, the following line is displayed during bootup:

```
BMC reset

------------------------------
BMC Loader version 2.06
------------------------------
```

11.3.1.6 Upgrading FRU Information and SDRs

The following subsections provide information about upgrading FRU information and Sensor Device Records (SDRs).

When upgrading FRU information and SDRs, the following switches must be set as indicated:

- S1 on the baseboard: 1,2,3,4,5,6,7,8 = ON (default settings)
- SW1 on the Gigabit Ethernet Mezzanine card: 1,2,3,4 = ON (default settings)
- SW1 on the FIC: 1,2,3,4 = ON (default settings)

11.3.1.6.1 Upgrading FRU Information

The FRU upgrade process preserves the following content in FRUs:

- Serial numbers
- MAC addresses
- Custom OEM records (id=1 and id=2, version 0.0)
- All other unknown type records

If the unknown type records (or Custom OEM records) do not already exist in the FRU, the upgrade process adds them.

The preservation process can be skipped by using the -o option when using the update fru or fis fru_upgrade command.

To determine which upgrade files to use for your hardware, see Table 58, “Components to be upgraded for earlier generation boards1” on page 138 or Table 59, “Components to be upgraded for newer boards1” on page 139.

1. Load the baseboard FRU using a command such as the following:

```
RedBoot> load -r -v -m TFTP -h 172.28.57.135  -b 0x1000000 /tftpdir/
<release_version>/ ixmb2801_4_rohs5_fru.bin
/
Raw file loaded 0x01000000-0x01000155
```

2. Upgrade the baseboard FRU using a command such as the following:

```
Redboot> update fru -c bb -b 0x1000000 -l 0x155
Warning! Updating the FRU will overwrite HW version number for this FRU - are
you sure (y/n)? y
---- Backing up: MAC adress record data
          Interface  Interface         MAC
Num     Type       Number        Address
0       0           0           00:00:00:00:00:00
1       0           1           00:00:00:00:00:01
2       3           0           00:00:00:00:03:00
3       3           1           00:00:00:00:03:01
4       6           0           00:00:00:00:06:00
5       6           1           00:00:00:00:06:01
6       6           2           00:00:00:00:06:02
7       6           3           00:00:00:00:06:03
---- Backing up: Serial number
Serial Number: 000001
-------------------------------------
Parsing new FRU to configuration data
---- Restoring: MAC adress record data
---- Restoring: Serial number
Converting updated FRU to binary
...................
..
FRU update ended.

TURN OFF AND ON POWER
```

Note: The procedures for upgrading the FRU information for other components is similar and therefore not repeated here.

11.3.1.6.2 Upgrading Sensor Device Records (SDRs)

To determine which upgrade files to use for your hardware, see Table 58, “Components to be upgraded for earlier generation boards1” on page 138.

1. Load the baseboard Sensor Device Records (SDR) file using a command such as the following:

```
RedBoot> load -r -v -m TFTP -h 172.28.57.135  -b 0x1000000 /tftpdir/
<release_version>/ixmb2801_4_rohs5_sdr.bin
-
Raw file loaded 0x01000000-0x010005c4
```

2. Upgrade the baseboard SDR using a command such as the following:

```
RedBoot> update sdr -c bb -b 0x1000000 -l 0x5c4
Update Sensor Data Record (SDR) definitions - are you sure (y/n)? y
.....................................................................
......
SDR update ended.

TURN OFF AND ON POWER
```

3. Repeat steps 1 and 2 above for the NP module, Gigabit Ethernet Mezzanine Card, MIC, and FIC SDRs as necessary. See Table 58, “Components to be upgraded for earlier generation boards1” on page 138 or Table 59, “Components to be upgraded for newer boards1” on page 139 for the names of the relevant SDR files.
4. Reboot both the BMC and NPU as follows:

```
RedBoot> reset bmc
BMC reset succeeded !

RedBoot> reset np
```

Note that when the old RedBoot starts on a board following FRU and SDR upgrade, but before RedBoot itself is upgraded, the media tests will fail with the following indications:

```
MB with GBE test .......... FAILED
DB with GBE test .......... FAILED
FIC_V2 test ............... FAILED
```

These failure indications are due to a lack of support for the current board hardware in versions of the bootloaders prior to firmware version 1B1/1B1G. This is normal behavior and these indications can be disregarded until the 1B1/1B1G or later version of Redboot is loaded (see Section 11.3.1.1, “Upgrading RedBoot” on page 140).

11.3.1.7 Upgrading CPLDs

IXB2850 boards contains several in-system programmable Complex Programmable Logic Devices (CPLDs):

- Baseboard NPU CPLD
- Baseboard BMC CPLD
- Baseboard JTAG router CPLD
- NPU module CPLD

- MMC #1 and #2 CPLDs (MMC #1 is not applicable to IXB2850 boards)
- MIC #1 and #2 CPLDs (MIC #1 is not applicable to IXB2850 boards)
- FIC CPLD

All mentioned above devices are programmable by the BMC with the exception of the BMC CPLD and JTAG router CPLD. These two latter CPLD are vital to the overall board operation and can be programmed only using the JTAG interface.

When upgrading CPLDs, the following switches must be set as indicated:

- S1 on the baseboard: 1,2,3,4,5,6,7=OFF, S8=ON
- SW1 on the Gigabit Ethernet Mezzanine card:
  - For the Mezzanine Card upgrade: 1=Unchanged, 2,3=ON, 4=OFF
  - For the MIC upgrade: 1=Unchanged, 2=OFF, 3,4=ON
- SW1 on the FIC: 1=Unchanged, 2,3=ON, 4=OFF

The CPLDs may be upgraded using the BMC console.

1. Enter the CPLD section:

```
q

y
```

2. Upload the baseboard CPLD:

```
l

<upload ixmb2801_4_rohs5.bin by XModem>
```

3. Upgrade the baseboard CPLD:

```
u
0
```

4. Upload the NP module CPLD:

```
l
<upload ixnpm2800_3A_rohs5.bin by XModem>
```

5. Upgrade the NP Module CPLD:

```
u
1
```

Note: The procedures for upgrading CPLDs on the Gigabit Ethernet Mezzanine card, MIC and FIC are similar and therefore not repeated here.

11.3.2 Local Software Upgrade of All Images at One Time

Software upgrade may be performed using the IXP28x0_upgrade.bin appimage that contains all upgradeable images.

The image should be loaded into RAM and run using the commands:

```
RedBoot> load -m tftp -r -b 0x10000000 /tftpdir/IXB28x0_upgrade.bin

RedBoot> go 0x10000000
```

The user may choose to upgrade all components using the fis all command or upgrade specific components using other fis commands as described in the following subsections.

Note: After performing the fis all or fis ssu_upgrade -a -o commands, the SSU images should be validated using the specified command sequence (see Section 11.3.2.1 and Section 11.3.2.5).

11.3.2.1 fis all

The fis all command is equivalent to performing the following sequence of commands:

```
Appimage> fis fru_upgrade -a
Appimage> fis sdr_upgrade -a
Appimage> fis fis_upgrade -i diag
Appimage> fis ssu_upgrade -a -o
```

If the Recovery Loader needs to be upgraded also, the following additional command sequence must be issued:

```
Appimage> fis image
Appimage> reset
```

Issuing the fis all command results in oneshot mode operation. After executing the fis all command, the following command sequence should be used to validate the operation and check the firmware components configuration:

```
Appimage> reset
....
RedBoot> reset bmc
RedBoot> ssu validate -i all
RedBoot> cfg verify
```

11.3.2.2 fis manifest

The fis manifest command lists all images that are contained in the currently running appimage. The following is a sample output:

```
Appimage> fis manifest
SSU-capable images:

LION_ssu.bin
redboot.ssu

Other FLASH images:

diag_ixdp2801_ssu.bin

FRU binaries:

ixmb2801_3a_fru.bin             for bb  (Main board FRU, spin 3A with FIC)
ixmb2801_4_fru.bin              for bb  (Main board FRU, spin 4 with FIC)
ixmb2801_4_rohs5_fru.bin        for bb  (Main board FRU, spin 4 with FIC (RoHS 5/6))
ixmb2801_3a_nofic_fru.bin       for bb  (Main board FRU, spin 3A without FIC)
ixmb2801_4_nofic_fru.bin        for bb  (Main board FRU, spin 4 without FIC)
```

```
ixmb2801_4_rohs5_nofic_fru.bin  for bb  (Main board FRU, spin 4 without FIC (RoHS 5/6))
ixnpm2800_3a_fru.bin            for npmod  (NP module FRU, spin 3A)
ixnpm2800_3a_rohs5_fru.bin      for npmod  (NP module FRU, spin 3A (RoHS 5/6))
ixd4gea1f_M_1_fru.bin           for db1 db2  (Mezzanine FRU, spin 1)
ixd4gea1f_M_1a_fru.bin          for db1 db2  (Mezzanine FRU, spin 1A)
ixd4gea1f_M_1a_rohs5_fru.bin    for db1 db2  (Mezzanine FRU, spin 1A (RoHS 5/6))
mic_f4gb_mezz_1_fru.bin         for mic1 mic2  (MIC FRU, 4-port fiber, spin 1)
mic_c4gb_mezz_1_fru.bin         for mic1 mic2  (MIC FRU, 4-port copper, spin 1)
mic_f4gb_v2_mezz_1_fru.bin      for mic1 mic2  (MIC FRU, 4-port fiber, spin 1, v2)
mic_f4gb_mezz_2_rohs6_fru.bin   for mic1 mic2  (MIC FRU, 4-port fiber, spin 2 (RoHS 6/6))
FIC_V2_1_fru.bin                for fic  (FIC FRU)
FIC_V2_1_rohs5_fru.bin          for fic  (FIC FRU (RoHS 5/6))

SDR binaries:

ixmb2801_3a_sdr.bin             for bb  (Main board SDR, spin 3A)
ixmb2801_4_sdr.bin              for bb  (Main board SDR, spin 4)
ixmb2801_4_rohs5_sdr.bin        for bb  (Main board SDR, spin 4 (RoHS 5/6))
ixnpm2800_3a_sdr.bin            for npmod  (NP module SDR, spin 3A)
ixnpm2800_3a_rohs5_sdr.bin      for npmod  (NP module SDR, spin 3A (RoHS 5/6))
ixd4gea1f_M_1a_sdr.bin          for db1 db2  (Mezzanine SDR, spin 1A)
ixd4gea1f_M_1a_rohs5_sdr.bin    for db1 db2  (Mezzanine SDR, spin 1A (RoHS 5/6))
mic_f4gb_mezz_1_sdr.bin         for mic1 mic2  (MIC SDR, 4-port fiber, spin 1)
mic_f4gb_mezz_1_sdr.bin         for mic1 mic2  (MIC SDR, 4-port copper, spin 1)
mic_f4gb_mezz_1_sdr.bin         for mic1 mic2  (MIC SDR, 4-port fiber, spin 1, v2)
mic_f4gb_mezz_2_rohs6_sdr.bin   for mic1 mic2  (MIC SDR, 4-port fiber, spin 2 (RoHS 6/6))
FIC_V2_1_sdr.bin                for fic  (FIC SDR)
FIC_V2_1_rohs5_sdr.bin          for fic  (FIC SDR (RoHS 5/6))
```

11.3.2.3 fis fru_upgrade

The full syntax of the command is:

```
fis fru_upgrade [-a] | [-f] | -i <imagename> [-c <bb|npmod|fic|db1|db2|mic1|mic2>]
[-o]
```

This command is used to update FRU information on all (-a option) or specific (-c option) components. The recognized parameters are:

- -a - upgrade all visible components using their FRU to determine the image to be used for update
  OR
  -c upgrade a specific component, that is one of the following:
  - bb - Baseboard
  - npmod - NPU module
  - fic - Fabric Interface Card (FIC)
  - db1 - Media Mezzanine Card (MMC) # 1 (not applicable to IXB2850 boards)
  - db2 - Media Mezzanine Card (MMC) #2
  - mic1 - Media Interface Card (MIC) #1 (not applicable to IXB2850 boards)
  - mic2 - Media Interface Card (MIC) #2
- -i - Use a specific image for the update (see Section 11.3.2.2, “fis manifest” on page 148 for the names of the images)
- -f - Use FRU information to determine which image should be used for update
- -o - Skip the preservation of serial numbers, MAC addresses, custom OEM records (id=1 and id=2, version 0.0) and all other unknown type records.

11.3.2.4 fis sdr_upgrade

The full syntax of this command is:

```
fis sdr_upgrade [-a] | [-f] | -i <imagename> [-c <bb|npmod|fic|db1|db2|mic1|mic2>]
```

This command is used to update the Sensor Data Record (SDR) on all (-a option) or specific (-c option) components. The recognized parameters are:

- -a - Upgrade all visible components using their FRU information to determine the image to be used for the update
  OR
  -c - Upgrade a specific component
- -i - Use a specific image for update (see Section 11.3.2.2, “fis manifest” on page 148 for the names of the images)
- -f - Use the FRU information to determine which image should be used for the update

11.3.2.5 fis ssu_upgrade

The full syntax of this command is:

```
fis ssu_upgrade [-o] -a | -i <bmc|redboot|linux|rootfs|target>
```

Upgrade an element using the Safe System Upgrade (SSU) mechanism. The images currently present are printed in “SSU-capable images” section of fis manifest command (see Section 11.3.2.2, “fis manifest” on page 148). The recognized parameters are:

- -o - Set upgraded component(s) in SSU_ONESHOT mode
- -a - Upgrade all available components
  OR
  -i - Upgrade a specific component (if present in the appimage).

After executing the fis ssu_upgrade command, the following command sequence should be used to validate the operation and check the firmware components configuration:

```
Appimage> reset
....
RedBoot> reset bmc
RedBoot> ssu validate -i all
RedBoot> cfg verify
```

11.3.2.6 fis fis_upgrade

The full syntax of this command is:

```
fis fis_upgrade -i <redboot|linux|rootfs|target|diag> [partition]
```

This command burns the specified image to the specified partition. The currently present images are printed in “SSU-capable images” and “Other images” sections of fis manifest command. The only recognized option is:

- -i - Identifies the image to be burned in flash. Possible images are:
  - redboot - Redboot* boot monitor
  - linux - Linux
  - rootfs - Linux root file system

  - target - Target application image
  - diag - Diagnostics image
- partition - The name of the partition into which the specified image is to be burned

11.3.2.7 fis image

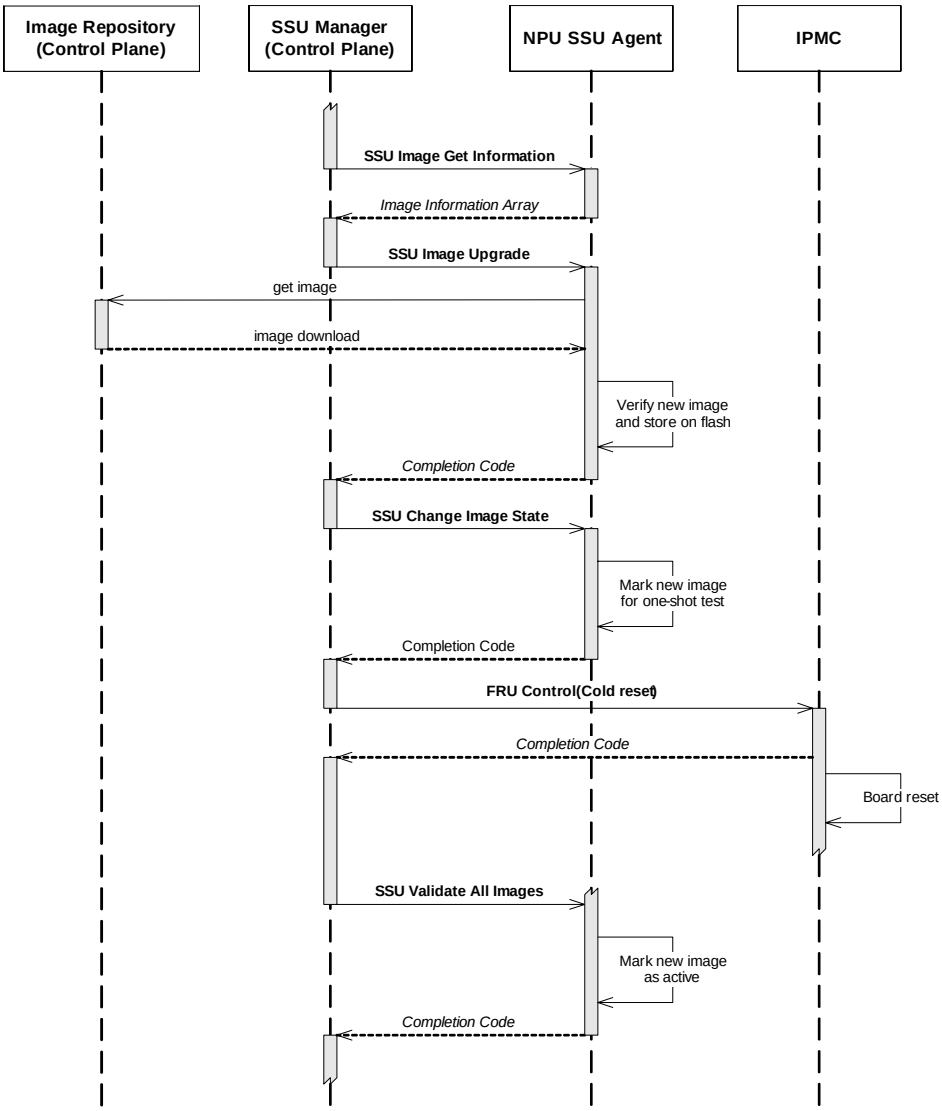
The fis image command is used to upgrade the Recovery Loader.

11.4 Remote NPU Firmware Upgrade

This section describes the IPMI interface between the SSU Agent and the SSU Manager that is used during the remote safe software upgrade procedure. The sequence diagram in Figure 43 shows a typical image upgrade procedure:

1. Get information about images currently stored and/or running on the network processor.
2. Request an upgrade of the selected image by downloading a new image file from a remote location.
3. Mark the newly downloaded image for one-shot test mode.
4. Reboot the board.
5. Verify that the new image has booted correctly and validate the new image.

Figure 43. SSU usage model



11.4.1 Safe System Upgrade IPMI Commands

The Safe System Upgrade (SSU) IPMI commands include:

- SSU Get Image Info
- SSU Image Upgrade
- SSU Change Image State (One-Shot)
- SSU Validate All Images
- SSU Upgrade Operation Status Command

11.4.1.1 SSU Get Image Info

SSU Get Image Info **NetFn**: Firmware (08h/09h) **CMD**: 25h		
Description: The response returns information about image stored in the particular flash partition.		
Data	**Byte**	**Data field**
Request	1	**Image Type** (see Table 61)
	2	**Partition Type** (see Table 60)
Response	1	**Completion Code** (see below)
	2	**Image State** (see Table 62)
	3	Image version length
	4:N	Image version
	4+N	Image description length
	5+N:M	Image description

Information about the image is stored in the local flash memory partition. The image version string is the same as that in the CRAMFS files when the image was downloaded.

Table 60 shows the types of partitions storing software images.

Table 60. SSU flash partition types

Type	Value	Description
SSU_PARTITION_BACKUP	0	Flash backup partition
SSU_PARTITION_ACTIVE	1	Flash active partition

Table 61 shows the types of upgradeable software images stored in NPU or BMC flash memory.

Table 61. SSU upgradeable image types

Type	Value	Description
SSU_BOOTLOADER	0	NPU Boot Monitor
SSU_KERNEL	1	NPU Linux kernel
SSU_ROOTFS	2	NPU Root File System
SSU_TARGET	3	NPU Target application image
SSU_BMC	4	BMC firmware

The board can store two versions of each image type in local flash. There is always one active and validated image, while the second placeholder can be occupied by a new image downloaded for a safe upgrade. A typical path for a new image to become the active one is:

- state = SSU_EMPTY
- action = download new image
- state = SSU_INACTIVE
- action = mark image for one-shot test
- state = SSU_ONESHOT
- action = reboot board

- state = SSU_ONESHOT_RUNNING
- action = validate current configuration
- state = SSU_ACTIVE_RUNNING

Details of other transitions and error conditions are provided in the descriptions in Table 62.

Table 62. SSU image state

State	Value	Description
SSU_EMPTY	0	No image (flash partition empty). Initial state of the flash partition if no image is downloaded. The partition also enters this state when the image download fails because of a CRC error. Allowed state transitions from SSU_EMPTY: • action = successful download of a new image -> new state = SSU_INACTIVE
SSU_INACTIVE	1	Inactive flash image. Image state after a successful download from a remote repository. An image also reverts to this state after explicit canceling of one-shot test mode or after system reboot if an image running in one-shot test was not validated. Allowed state transitions from SSU_INACTIVE: • action = mark image for one-shot test mode -> new state = SSU_ONESHOT • action = failed attempt to download new image -> new state = SSU_EMPTY • action = successful download of a new image -> new state = SSU_INACTIVE
SSU_ONESHOT	2	Inactive flash image marked for one-shot test. An inactive image can be marked for one-shot test using SSU IPMI command. Marking for one-shot test does not imply automatic system reboot. Possible state transitions from SSU_ONESHOT: • action = cancel one-shot test mode using IPMI -> new state = SSU_INACTIVE • action = successful reboot (new image works) -> new state = SSU_ONESHOT_RUNNING • action = failed reboot (watchdog expires) -> new state = SSU_INACTIVE
SSU_ONESHOT_RUNNING	3	Flash image running in one-shot test mode. State entered by one-shot test image after system reboot. The running one-shot image must be explicitly validated using IPMI. Otherwise, it automatically reverts to inactive state after next reboot. Possible state transitions from SSU_ONESHOT_RUNNING: • action = validate current configuration -> new state = SSU_ACTIVE_RUNNING • action = cancel one-shot test mode using IPMI -> new state = SSU_INACTIVE • action = reboot without prior validation -> new state = SSU_INACTIVE
SSU_ACTIVE	4	Active flash image. A validated image stored in flash, but not currently running. An active flash image should not be running when a remote image is loaded (SSU_REMOTE_RUNNING) or if the second image is tested in one-shot mode (SSU_ONESHOT_RUNNING). The only way to leave this state is to validate the second image. The active image cannot be overridden, nor marked for one-shot mode. Possible state transitions from SSU_ACTIVE: • action = validate the other image -> new state = SSU_INACTIVE

Table 62. SSU image state

State	Value	Description
SSU_ACTIVE_RUNNING	5	Active flash image running. Validated image stored in local flash and currently running. No SSU operations are allowed during this state. An image may leave this state and enter SSU_ACTIVE after system reboot, if another image is selected to run (one-shot test or remote). Possible state transitions from SSU_ACTIVE_RUNNING: • action = select one-shot or remote image for running -> new state = SSU_ACTIVE

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_ BUSY	C0h	Responder busy (**SSU Image Upgrade** operation in progress)
COMPCODE_INVALIDFIELD	CCh	Invalid Image Type or Partition Type

11.4.1.2 SSU Image Upgrade

SSU Image Upgrade
NetFn: Firmware (08h/09h)
CMD: 26h

Description: Control plane SSU Manager sends this command to download new image(s) from a remote repository and store the image(s) in local flash.

A new image can only be written to a flash partition that is either empty (SSU_EMPTY) or stores an image in the SSU_INACTIVE state (see Table 62, “SSU image state” on page 154). If there is no such partition, the download request is discarded.

An application specifies location of the new image (CRAMFS file) and the transport method to download the file (for example, TFTP or NFS). After a successful download, the new image is stored as SSU_INACTIVE. If the image cannot be found, the partition content remains unchanged. If a downloaded file is corrupted, the partition is marked as SSU_EMPTY.

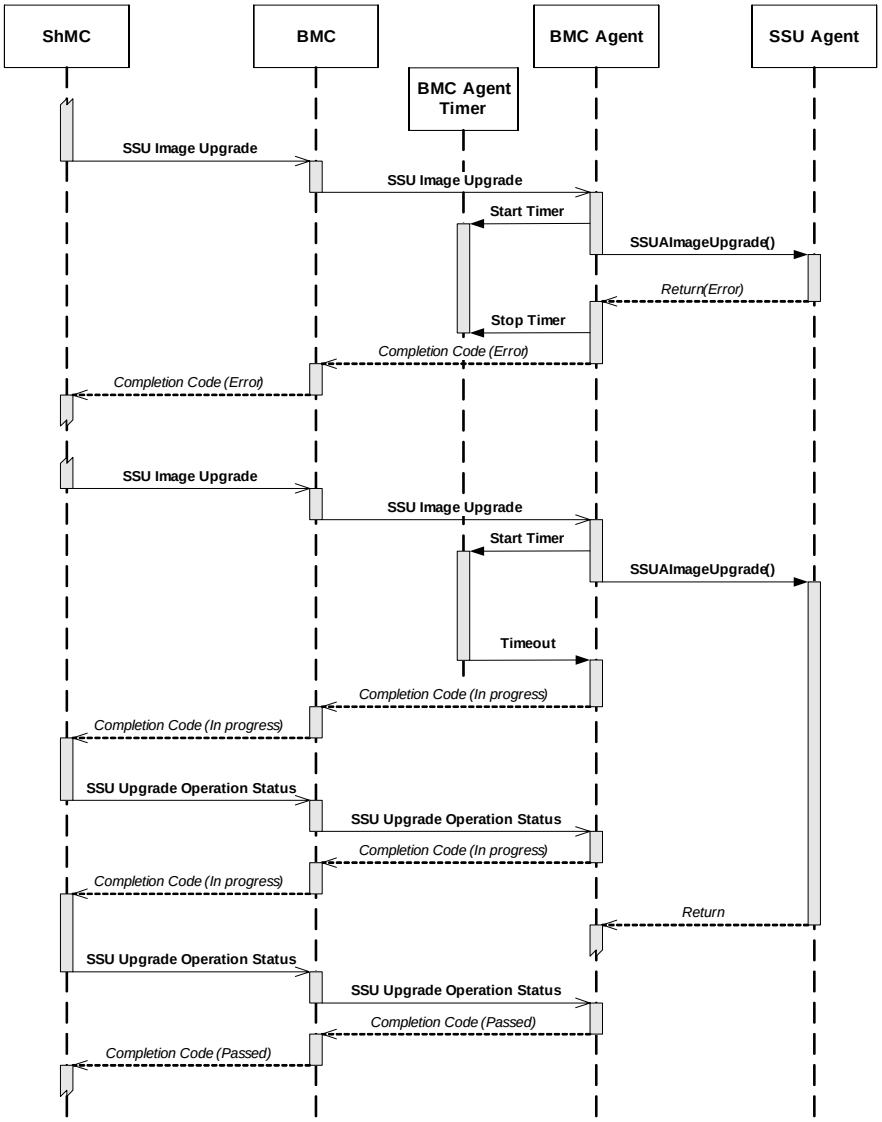
Data	Byte	Data field
Request	1	**Image Type** (see Table 61)
	2	**Transport protocol to download image**: • 00h – NFS • 01h – TFTP
	6:3	IP version 4 address of image repository server
	7	Length of path string
	8:N	Path in remote repository
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_FAILED	20h	Upgrade operation failed

Code	Value	Description
COMPCODE_INPROGRESS	21h	Upgrade operation is in progress. The final status of this operation should be obtained using **SSU Upgrade Operation Status** command (see Section 11.4.1.5). See Table 62 for details about obtaining **SSU Image Upgrade** operation final status.
COMPCODE_ BUSY	C0h	Responder busy (other **SSU Image Upgrade** operation in progress)
COMPCODE_INVALIDFIELD	CCh	Invalid Image Type or Transport Protocol. The new image is not downloaded and the images currently stored in local flash remain unchanged.

Figure 44. SSU image upgrade operation



11.4.1.3 SSU Change Image State (One-Shot)

SSU Change Image State **NetFn**: Firmware (08h/09h) **CMD**: 27h		
Description: This message is sent to set/clear the one-shot test mode. Setting one-shot mode is allowed only for images in the SSU_INACTIVE state (see Table 62, "SSU image state" on page 154). The selected image is marked as SSU_ONESHOT and it will be loaded after the next system reboot. Clearing one-shot mode is allowed only for images in the SSU_ONESHOT or SSU_ONESHOT_RUNNING states. The selected image enters the SSU_INVALID state.		
Data	**Byte**	**Data Field**
Request	1	**Image Type** (see Table 61)
	2	**Command**: • 00h – (SSU_ACTION_ONESHOT_CANCEL) Revert image state from one-shot test to inactive • 01h – (SSU_ACTION_ONESHOT_SET) Mark inactive image for one-shot test
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_ BUSY	C0h	Responder busy (**SSU Image Upgrade** operation in progress)
COMPCODE_OUTOFRANGE	C9h	Invalid Command (out of supported range)
COMPCODE_INVALIDFIELD	CCh	Invalid Image Type

11.4.1.4 SSU Validate All Images

SSU Validate All Images **NetFn**: Firmware 08h/09h) **CMD**: 28h		
Description: This command validates the currently running configuration – all images in the SSU_ONESHOT_RUNNING state move to the SSU_ACTIVE_RUNNING state. It is not possible to validate only one selected element and not validate others.		
Data	**Byte**	**Data Field**
Request		
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_ BUSY	C0h	Responder busy (**SSU Image Upgrade** operation in progress)

11.4.1.5 SSU Upgrade Operation Status Command

SSU Upgrade Operation Status **NetFn**: Firmware (08h/09h) **CMD**: 29h		
Description: Command used to track **SSU Image Upgrade** operation progress		
Data	**Byte**	**Data Field**
Request		
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	**SSU Image Upgrade** operation completed successfully
COMPCODE_FAILED	20h	**SSU Image Upgrade** operation failed
COMPCODE_INPROGRESS	21h	**SSU Image Upgrade** operation in progress
COMPCODE_IDLE	22h	**SSU Image Upgrade** operation not started

11.5 Remote BMC Firmware Upgrade

11.5.1 Image Upgrade on BMC

Upgrade of the BMC operational code is controlled by the SSUA running on the NPU, that is, the NPU upgrade agent retrieves a new BMC firmware image from the image repository and passes it to the BMC using the IPMI driver.

Next, the BMC operational image stores the received image in one of two locations on its flash as shown in Figure 30, “BMC flash memory organization” on page 85. The image selection procedure is based on the image status word. The image status word for the BMC is supported by the SSUA on the NPU, but its local flash copy must be present on the BMC for image selection purposes during the BMC Initial Loader phase. After every modification of this word by either BMC (both operational image and loader) or the SSUA, it has to be transferred to the other side as soon as communication between processors is established or possible. The SSUA on the NPU requests the BMC to read or write the status word.

Table 63. BMC image status word

Bit	Name	Description
7	Future image switchover indicator	This bit is set by the SSUA to force the BMC Loader to load the second image in safe mode. If bit is set to 1, the BMC Loader clears this bit and loads the opposite image to that specified by the “Image selector”.
6	Boot mode indicator	This bit indicates that current BMC image is started in safe mode, used only as an indication of boot status, and set by the BMC Loader immediately after clearing bit 7.
5	Image selector	This bit selects an active image to be loaded (either first or second copy). This bit may be modified only by the SSUA on a control plane validation request.
4 to 0	Reserved for future use	Unused

After storing the new image, the BMC sends the status of the upgrade operation to the SSUA on the NPU, and the SSUA passes this status to the SSUM.

The SSUM sends a message to the SSUA to run the newly upgraded image in one-shot mode. This message is passed to the BMC via a “write status word” request.

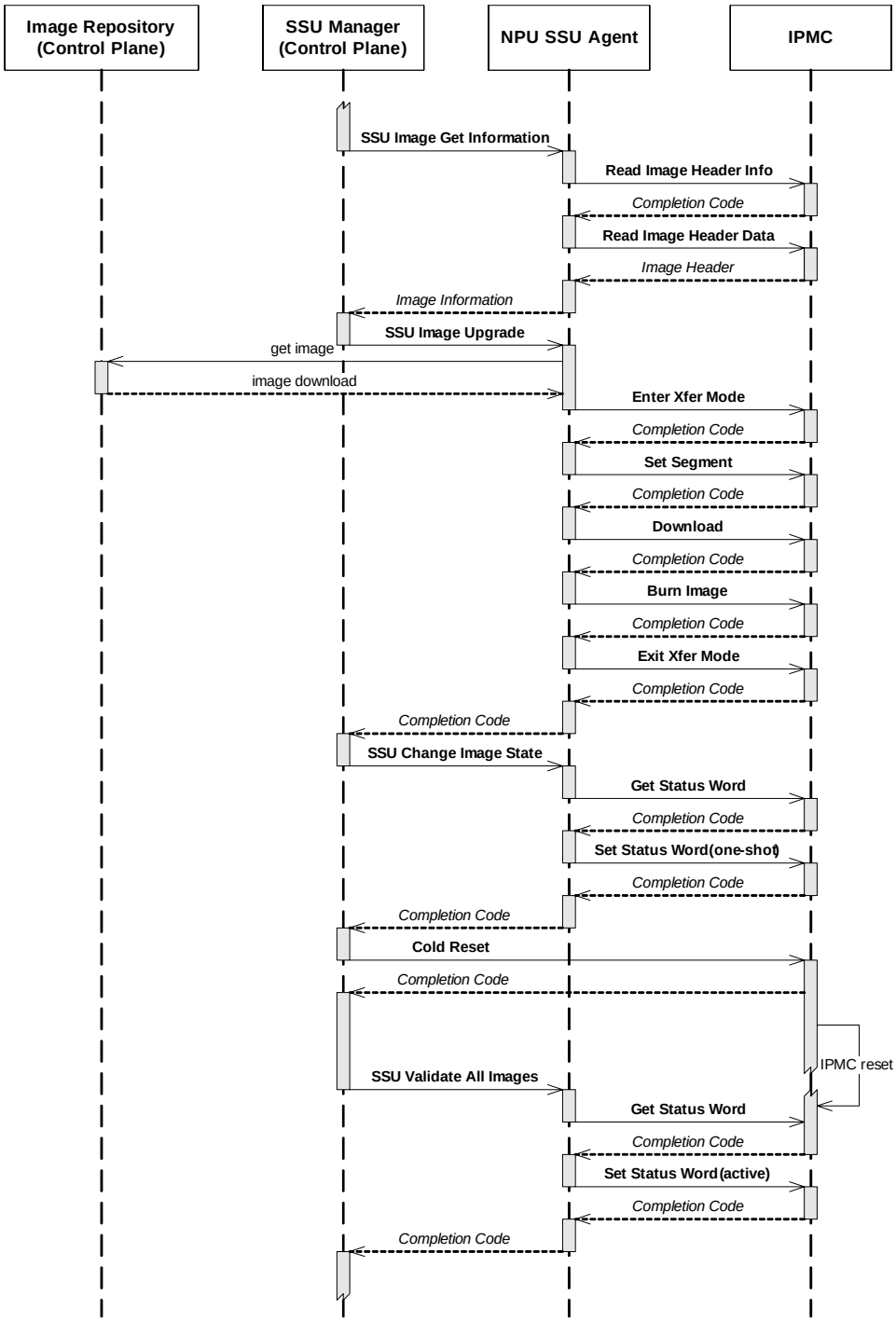
After setting one-shot mode, the SSUM requests to reboot the BMC firmware.

This is a very important part of the whole BMC upgrade process, because if an image running in one-shot mode fails to initialize its communication interfaces (with the NPU or ShMC), there is no possibility to send a reboot request that will cancel the one-shot mode and automatically restore the previous version of the BMC image. Therefore, there is a timeout for the establishment of a connection with the ShMC and NPU that is implemented in the BMC that ensures automatic reboot in case of such an error.

Next, after the control plane ensures that the new image is operational, the SSUM sends a validation request to the SSUA that accepts the current image selection for subsequent reboots.

Figure 45 presents interactions between the Control Plane (SSUM, image repository) and Data Plane (NPU SSUA and BMC) during a remote BMC image upgrade process.

Figure 45. Remote upgrade of BMC firmware



12.0 Specifications

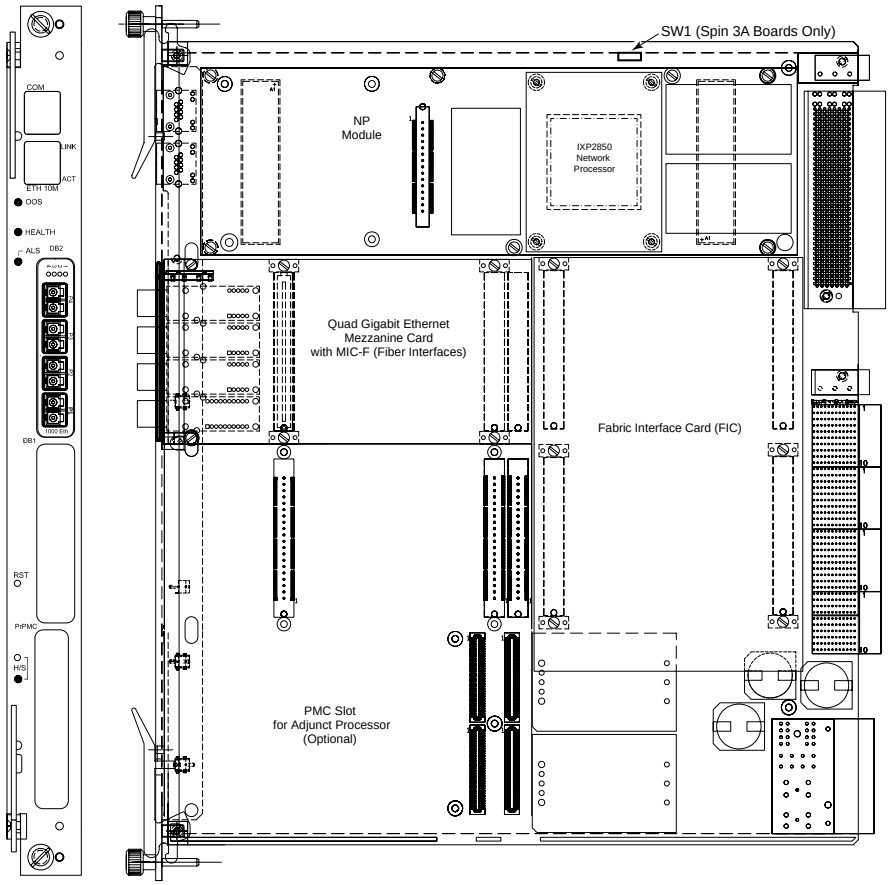
12.1 IXB28504XGBEFSx Mechanical Specifications

All mechanical specifications conform to PICMG 3.0 (AdvancedTCA). Key parameters for Intel NetStructure® IXB28504XGBEFSx board are:

- **Form factor**: AdvancedTCA
- **Dimensions**: 8U x 280 mm x 1.2 inch pitch

Figure 46 shows the mechanical layout for IXB28504XGBEFSx boards.

Figure 46. IXB28504xGbEFSx board mechanical layout



12.1.1 Quad Gigabit Ethernet Mezzanine Card Mechanical Specification

IXB28504XGBEFSx boards provide 4 x 1000BASE-CX fiber Gigabit Ethernet interfaces.

Figure 47 shows the physical arrangement of the Quad Gigabit Ethernet Mezzanine Card, comprising a Media Mezzanine Card (MMC) and a Fiber Media Interface Card (MIC-F), and the physical connections to the baseboard. The MIC-F card provides the fiber connections to the front panel. The dimensions of the card are 141 mm x 74 mm.

Figure 47. Quad Gigabit Ethernet Mezzanine Card (fiber) assembled to baseboard

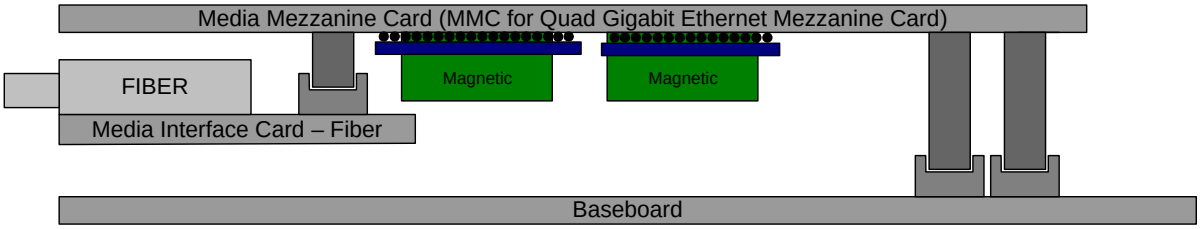
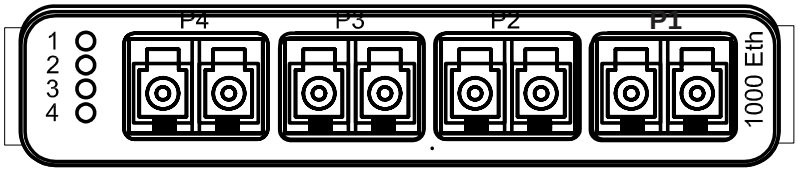


Figure 48 shows the front panel of the Quad Gigabit Ethernet Mezzanine Card with fiber interfaces.

Figure 48. Quad Gigabit Ethernet Mezzanine Card front panel (fiber)

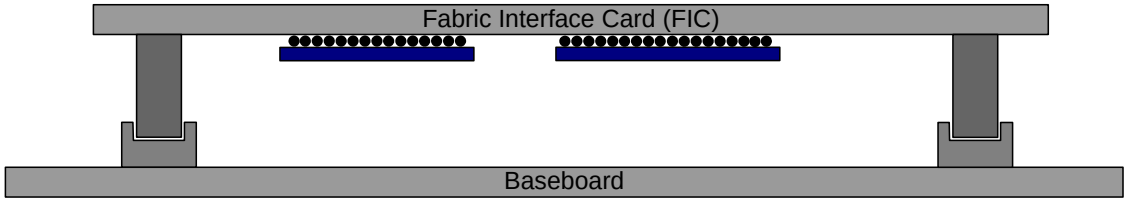


12.1.2 Fabric Interface Card Mechanical Specification

IXB28504XGBEFSx boards provide a Fabric Interface Card (FIC).

Figure 49 shows the physical connection of the Fabric Interface Card (FIC) to the baseboard. Four identical 152-pin Mictor* connectors are used for this connection. The dimensions of the card are 166 mm x 109 mm.

Figure 49. Fabric Interface Card assembled to baseboard



12.1.3 PCI Mezzanine Card (PMC) Site Mechanical Specification

All IXB2850 boards provides a site for the addition of a PCI Mezzanine Card (PMC) such as an Adjunct Processor (AP). The site conforms with the PMC standard, accommodating a card with dimensions 74 mm x 149 mm.

12.2 Processor Specification

Network Processor

- **Processor Type**: Intel® IXP2850 network processor with 16 v2 microengines capable of operating at 1.4 GHz and an Intel XScale® core capable of operating at 700 MHz
- **Encryption/Decryption Capability**: Two integrated and independent crypto blocks for accelerating DES, 3DES, AES, and SHA-1.

Interfaces

- **Packet**: SPI-4.2, 16-bit
- **PCI**: 64-bit @ 66 MHz
- **Slow Port**: 16-bit

Memory

- **RDRAM**: Three channels (768 MB) with Error Correcting Code (ECC)
- **SRAM**: Four QDR controllers (32 MB + TCAM) with parity
- **FLASH**: 160 MB

12.3 Interface Specification

AdvancedTCA Backplane Interface

- **Base Interface**: Two channels x 1 Ethernet, 1000BASE-T
- **Fabric Interface**: Two channels x 4 Ethernet, 1000BASE-BX
- **IPMB**: AdvancedTCA compatible, dual IPMI bus
- **System Clock Interface**: CLK 1A/B, CLK 2A/B
- **Reference Clock Interface**: CLK 3A/B

Quad Gigabit Ethernet Mezzanine Card

- **External Interfaces**: 4 x SFF 1000BASE-SX (fiber) on the front panel
- **Internal Interfaces**: SPI-3 32-bit MPHY, slow port, I^2C, JTAG

Fabric Interface Card (IXB28504XGBEFSx only)

- **External Interfaces**: 4 x 1000BASE-BX
- **Internal Interfaces**: SPI-3 4x8 SPHY, slow port, I^2C, JTAG

PCI Mezzanine Card (PMC) Site

- **External Interfaces**: Front Panel access
- **Internal Interfaces**: PCI (32-bit @ 66 MHz); two Gigabit Ethernet links compliant with the PICMG 2.15 specification.

12.4 Environmental Specification

Table 64 summarizes the environmental limits of IXB2850 boards, both operating and nonoperating.

Table 64. Environmental specifications

Parameter	Conditions	Specification Detail
Temperature (Ambient)	Operating	0 to 55 °C
	Storage	-40 to 70 °C
Airflow	Operating	30 cubic feet per minute (CFM) minimum
Humidity	Operating	15%-90% (non-condensing) at 55 °C
	Storage	5%-95% (non-condensing) at 40 °C
Altitude	Operating	4000 m (13123 ft.) ***Note:*** May require additional cooling above 1800 m (5905 ft.)
	Storage	15000 m (49212 ft.)
Unpackaged Vibration	Operating	Sine sweep: • 5 to 100 Hz: 1G @ 0.25 Octave/minute • 100 to 500 Hz: 1G @ 1 Octave/minute Random profile: • 5 Hz @ 0.01 g2 /Hz to 20 Hz @ 0.02 g2 /Hz (slope up) • 20 Hz to 500 Hz @ 0.02 g2 /Hz (flat) • 3.13 g RMS, 10 minutes per axis for all 3 axes
	Storage	5 to 50 Hz: 0.5G @ 0.1 Octave/minute 50 to 500 Hz: 3G @ 0.25 Octave/minute
Shock	Operating	30G/11 ms half sine
	Storage	50G, 170 inches/second trapezoidal

IXB2850 boards have been tested to Network Equipment Building System (NEBS) Level 3. A report documenting the tests and results is available on request. Contact your Intel representative for more information.

12.5 Reliability Specification

12.5.1 MTBF

The predicted MTBF values are calculated according to *Reliability Prediction for Electronic Equipment, SR-332 Issue 1, Telcordia Technologies, May 2001*, but scaled with suitable modifiers based on Intel field failure data for a similar product, to reflect performance in the field.

The MTBF for IXB2850 boards are as follows:

- IXB28504XGBEFSx - 353 K hours

12.5.2 MTTR

The target Mean Time to Repair (MTTR) for IXB2850 boards is 1 hour for a staffed CO, with replacement boards on-site. For an unstaffed Central Office (CO), the MTTR shall include an additional 3 hours for dispatch and travel time, and 1 hour for locating and replacing the board. This is a total of 4 hours.

12.5.3 Service Life Target

The service life target for IXB2850 boards should be greater than 10 years.

12.6 Power Consumption

The key power supply parameters are:

- **Supported Voltage**: Dual -48 VDC, supplied by redundant power rails
- **Maximum Power Draw**: 110 W to 135 W

12.7 Weight

The unpackaged weight of an IXB2850 board is approximately 8 lbs.

12.8 Cable Specifications

12.8.1 Debug Console Cable Specification

The debug console cable, IXB3GDEBUGCABLE, is a separately orderable item. Figure 50 shows the mapping between the RJ45 and RS-232 ports (DB9 female plug) for two serial connections. The cable is used for IXB2850 boards to access the Network Processor (NP) or the Intelligent Platform Management Controller (IPMC).

Note: The serial console cable must be shielded with a capacitance that does not exceed a value of 2500pF.

If constructing such a cable, the maximum length should be:

- 10 meters for transfer rates up to 115 kbit/s
- 15 meters for transfer rates up to 20 kbit/s.

Figure 50. NP/BMC console cable

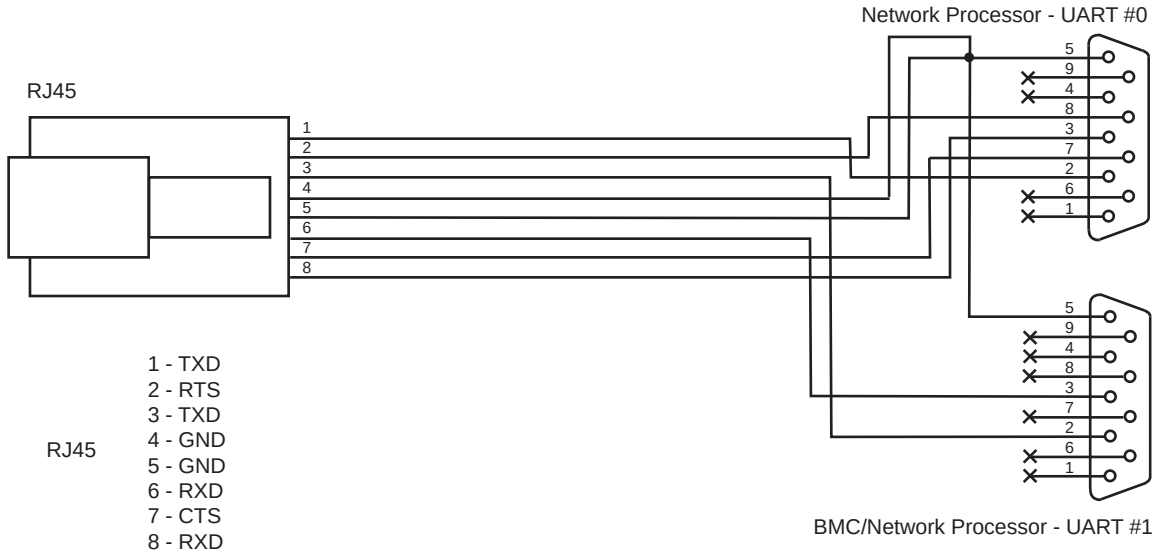


Table 65 shows the connections between the RJ45 and the two DB9 connectors.

Table 65. NP/BMC console cable connections

RJ45 Pin	Signal	DB9 UART #0 Pin	DB9 UART #1 Pin
1	TXD	2	-
2	RTS	8	-
3	TXD	-	2
4	GND	5	5
5	GND	5	5
6	RXD	-	3
7	CTS	7	-
8	RXD	3	-
Note: DB9 pin numbers not shown have no connection.			

Note: The cabling connections in Table 65 support full modem signaling on the UART #0 DB9 when only that DB9 is used. When both DB9 connectors are used, full model signaling is not available.

13.0 Component Technology

Intel NetStructure® IXB2850 boards use the following components:

- **Intel® IXP2850 network processor**
  A high-performance, multi-threaded network processor that encompasses 16 RISC-based microengines and an embedded Intel XScale® core control processor.
- **Intel® IXF1104**
  This Quad Gigabit Ethernet MAC Controller is connected to the NPU through the Media Access Module (for packet transmission/reception) and through the slow port for device configuration and management. Possible IXF1104 chip configurations:
  - Base/Fabric mode - two ports used to support the fabric interface and two other ports used for access to the base interface on the AdvancedTCA backplane
  - Fabric mode - all four ports are connected to the fabric interface (two ports to channel 1 and two ports to channel 2 or all 4 ports to channel 1)
- Marvell* Alaska 1011
  Single Quad Gigabit Ethernet PHY. Four of these devices work as a set of PHYs for the on-board IXF1104 and are connected to this chip through the GMII interface. The Alaska chip is configurable and manageable indirectly by the NP through the IXF1104.
- **Intel® 82546**
  This Dual Port Gigabit Ethernet Controller is fully accessible via the PCI bus; packet transmission/reception as well as device configuration and management. This MAC Controller is used to support the base interface on the AdvancedTCA backplane in the PICMG 3.1 configuration.
- **SPI-3/4 Bridge and Fork FPGA**
  These components make up the Media Access Module which switches packets/cells between IXP28x0 SPI-4 bus and four SPI-3/UTOPIA buses connected to the following devices: on-board IXF1104 (quad Gigabit Ethernet Controller), FIC, Quad Gigabit Ethernet Mezzanine Card, and a second Mezzanine Card (not used on IXB2850 boards). This module is configurable through the slow port.
- **Intel® 21154**
  Two PCI bridges configurable through the PCI bus.
- **Cirrus Logic* CS8900**
  A 10Mb Ethernet Controller that is connected to the NPU and used for debug purposes (console and debugger support). This chip is fully manageable (including packet transmission and reception) through the slow port.
- **16C550**
  A dual UART device dedicated to the NPU, one used for debugger support, and one for communication with the Board Management Controller (BMC). Note that there is also a third UART port embedded in the IXP28x0 that is used as a NPU console.

- **NPU Flash**
  160MB accessible through the slow port. Note that flash memory available for NPU is split into five banks of 32 MB each (2 x 16MB) due to limited NP slow port addressing capabilities. One bank is mounted on the NP module and remaining four banks are mounted on a baseboard.
- **Crosspoint Switch**
  The 34 x 34 crosspoint switch is implemented using an Analog Devices* Asynchronous Digital crosspoint switch, AD8152. This chip allows flexible mapping of board Gigabit Ethernet MAC ports to AdvancedTCA fabric interface channels ports.

14.0 Return Material Authorization

This chapter provides information about returning a product for repair or replacement.

14.1 Returning a Defective Product (RMA)

Before returning any product, contact an Intel Customer Support Group to obtain either a Direct Return Authorization (DRA) or Return Material Authorization (RMA). Return Material Authorizations are only available for products purchased within 30 days. Return contact information by geography is given in the following subsections.

If the Customer Support Group verifies that the product is defective, they will have the Direct Return Authorization/Return Material Authorization Department issue you a DRA/RMA number to place on the outer package of the product. Intel cannot accept any product without a DRA/RMA number on the package.

14.1.1 For the Americas

Return Material Authorization (RMA) credit requests e-mail address: requests.rma@intel.com

Direct Return Authorization (DRA) repair requests e-mail address: uspss.repair@intel.com

DRA on-line form: http://support.intel.com/support/motherboards/draform.htm

Intel Business Link (IBL): http://www.intel.com/ibl

Telephone No.: 1-800-INTEL4U or 480-554-4904

Office Hours: Monday - Friday 0700-1700 MST Winter / PST Summer

14.1.2 For Europe, Middle East, and Africa (EMEA)

Return Material Authorization (RMA) e-mail address - emea.fs@intel.com

Direct Return Authorization (DRA) for repair requests e-mail address: emea.fs@intel.com

Intel Business Link (IBL): http://www.intel.com/ibl

Telephone No.: 00 44 1793 403063

Fax No.: 00 44 1793 403109

Office Hours: Monday - Friday 0900-1700 UK time

14.1.3 For Asia and Pacific (APAC)

RMA/DRA requests email address: apac.rma.front-end@intel.com

Telephone No.: 604-859-3111 or 604-859-3325

Fax No.: 604-859-3324

Office Hours: Monday - Friday 0800-1700 Malaysia time

Return Material Authorization (RMA) requests e-mail address:
rma.center.jpss@intel.com

Telephone No.: 81-298-47-0993 or 81-298-47-5417

Fax No.: 81-298-47-4264

Direct Return Authorization (DRA) for repair requests, contact the JPSS Repair center.

E-mail address: sugiyamakx@intel.co.jp

Telephone No.: 81-298-47-8920

Fax No.: 81-298-47-5468

Office Hours: Monday - Friday 0830-1730 Japan time

15.0 Customer Support

15.1 Customer Support

This chapter offers technical and sales assistance information for this product. Information on returning an Intel NetStructure® product for service is in the following chapter.

15.2 Technical Support and Return for Service Assistance

For all product returns and support issues, please contact your Intel product distributor or Intel Sales Representative for specific information.

15.3 Sales Assistance

If you have a sales question, please contact your local Sales Representative or the Regional Sales Office for your area. Address, telephone and fax numbers, and additional information is available at Intel's website located at:

http://www.intel.com/network/csp/sales/

15.4 Product Code Summary

Table 66 presents the product codes.

Table 66. Product code summary

Product Code	MM Number	Description
IXB28504XGBEFS	870497	AdvancedTCA line card: • Intel® IXP2850 network processor • Four 1000BASE-SX Gigabit Ethernet fiber interfaces accessible from the front panel • Board may contain restricted substances
IXB28504XGBEFSW	882941	AdvancedTCA line card: • Intel® IXP2850 network processor • Four 1000BASE-SX Gigabit Ethernet fiber interfaces accessible from the front panel • Lead-free to second level interconnect
IXB28504XGBEFSR	885586	AdvancedTCA line card: • Intel® IXP2855 network processor • Four 1000BASE-SX Gigabit Ethernet fiber interfaces accessible from the front panel • Lead-free to second level interconnect
IXB3GDEBUGCABLE	868313	Debug cable that allows connection of consoles to the Network Processor (NP) and the Intelligent Platform Management Controller (IPMC).

16.0 Certifications

Safety:

- IEC60950-1
- EN60950
- UL/CSA 60950-1
- EN60825
- IEC825

Electromagnetic Compatibility (EMC) emissions:

- CISPR22/EN55022 Class A
- EN300386
- FCC Rules CFR 47 Part 15B Class A
- ICES-003 Class A
- VCCI Class A

Electromagnetic Compatibility (EMC) immunity:

- CISPR24/EN55024
- EN300386

Network Equipment Building System (NEBS) compliance:

- Testing to Level 3. A report documenting the tests and results is available on request. Contact your Intel representative for more information.

17.0 Agency Information

17.1 North America (FCC Class A)

Federal Communications Commission (FCC) Part 15 Rules

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

17.2 Canada – Industry Canada (ICES-003 Class A)

Industry Canada ICES-003 Issue 3

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matérial brouilleur du Canada.

17.3 European Union

The products covered by this notice meet the following European Directives:

- 73/23/EEC Low Voltage Directive
- 89/336/EEC EMC Directive

To achieve CE compliance, be sure to select a host that already meets the EMC and Low Voltage Directives before the addition of any optional board. Remember that the use of option boards declared compliant with the Directives by their manufacturer only gives “presumption of compliance” for the whole system. It is the responsibility of the system supplier to verify that the requirements of the listed Directives are still met by the final system, as supplied to the end-user. System integrators should take notice of further conditions expressed in the sections below and the Safety Information sheet supplied with each board.

Compliance with the R&TTE Directive

The R&TTE Directive includes its own safety and EMC requirements. Although equipment declared compliant to the R&TTE Directive does not require explicit declaration of conformity to EMC and Low Voltage Directives, above conditions must also be met to satisfy the safety and EMC requirements of the R&TTE Directive.

Intel Declarations of Conformity for the products covered by this notice can be found under the “Network Building Blocks” heading at http://developer.intel.com/design/litcentr/ce_docs

Manufacturer's office in European Union:

Intel Corporation (UK) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ
UK
Tel: +44 (0)1793 403000
Fax: +44 (0)1793 641440

18.0 Safety Warnings

18.1 Safety Precautions

Review the following precautions to avoid personal injury and prevent damage to this product or products to which it is connected. To avoid potential hazards, use the product only as specified.

Read all safety information and understand the precautions associated with safety symbols, written warnings, and cautions before accessing parts or locations within the unit.

SYSTEM FOR RESTRICTED ACCESS USE ONLY!

Warning: To avoid the risk of electrical shock hazard, special measures and precautions must be taken when using these products:

- Access to this equipment must be restricted by locating this equipment where access can only be gained by SERVICE PERSONNEL who have been informed about the reasons for the restrictions applied to the location and about any precautions that shall be taken. Access is through the use of a TOOL, lock and key, or other means of security and is controlled by the authority responsible for the location.
- This product should only be used by SERVICE PERSONNEL who have the knowledge and training required to work with products of this type.
- To avoid shock, ensure that the chassis power cables are connected to a properly wired and grounded receptacle.
- The system containing these boards should not be operated with the faceplates, blank panels, or covers removed. Some voltages, that are on the board and inside the chassis, present an electrical shock and/or energy hazard to the user. Keep hands out of the chassis when power is applied or when performing hot swap of the boards.

Warning: Certain components such as heat sinks, power regulators, and processors may be hot; care should be taken to avoid these components.

Caution: Boards with fiber interfaces contain Class 1 laser devices. Class 1 lasers are not considered to be hazardous. All adjustments have been made at the factory prior to shipment of the devices. No maintenance or alteration to the device is required. Do not tamper with or modify the performance of the device.

19.0 Related Documentation

The following documents provide information that is related to the information provided in this document:

- *Dear Customer Letter*, a printed document supplied with each IXB2850 board, providing a pointer to the product web page
- *Safety Information Sheet*, a printed document supplied with each IXB2850 board
- Data Sheet
- PICMG 3.0 - www.picmg.org
- ECN 3.0-1.0 - www.picmg.org
- PICMG 3.1 - www.picmg.org
- IPMI 1.5 - www.intel.com/design/servers/ipmi
- FRU 1.0 - www.intel.com/design/servers/ipmi
- Wired for Management Baseline 2.0 - www.intel.com/design/servers/ipmi
- RedBoot* User's Guide - http://sources.redhat.com/ecos/docs-latest/redboot/redboot-guide.html

20.0 Glossary

AdvancedTCA	Advanced Telecom Computing Architecture
APB	Advanced Peripheral Bus
BMC	Board Management Controller
BSP	Board Support Package
CGL	Carrier Grade Linux
CLI	Command Line Interface
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRAMS	Compressed read-only file system for Linux* systems
CRC	Cyclic Redundancy Code
ECC	Error Correcting Code
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIC	Fabric Interface Card
Flash	Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
GMII	Gigabit Media Independent Interface
GPIO	General Purpose Input Output
GPL	General Public License
HA	High Availability
IEEE	Institute of Electrical and Electronics Engineers
IKE	Internet Exchange Key
IPMC	Intelligent Platform Management Controller
IPMI	Intelligent Platform Management Interface
JTAG	Joint Test Action Group
KCS	Keyboard Controller Style
LED	Light Emitting Diode
LSP	Linux Support Package
MAC	Media Access Controller
MIC	Media Interface Card
MMC	Media Mezzanine Card
MPHY	Multiple Physical Device

MTBF	Mean Time Between Failures
MTD	Memory Technology Device
NP	Network Processor
OC	Optical Carrier
OS	Operating System
PCI	Peripheral Component Interconnect
PHY	Physical device
PMC	PCI Mezzanine Card
POST	Power On Self Test
QDR	Quad Data Rate
RDRAM	Rambus* Dynamic Random Access Memory
ROM	Read Only Memory
RGMII	Reduced Gigabit Media Independent Interface
SDK	Software Development Kit
ShMC	Shelf Management Controller
Slow port	Address/Data I/O bus used for device configuration
SPHY	Single Physical Device
SPI	Optical Internetworking Forum specification for System Packet Interface
SRAM	Static Random Access Memory
SSU	Safe System Upgrade
SSUA	Safe System Upgrade Agent
SSUM	Safe System Upgrade Manager
TCAM	Ternary Content Addressable Memory
UART	Universal Asynchronous Receiver-Transmitter
VCXO	Voltage controlled crystal oscillator

Appendix A Boot Monitor Console Commands

A.1 Command Summary

The commands detailed in this section can be run from the RedBoot> prompt. The commands can also be incorporated into startup scripts that are stored in flash memory and automatically executed at startup.

Table 67. Boot Monitor console commands

Command	Description	Refer to
alias	Manage aliases kept in flash memory	Section A.2.1
alignment	Manage static alignment for the Media Switch Fabric (MSF) configuration	Section A.2.2
baudrate	Set or query the system console's baud rate	Section A.2.3
cache	Manage machine caches (data and instruction)	Section A.2.4
cfg mac	Assigns MAC address value to a given Ethernet port	Section A.2.5
cfg read	Read configuration parameter values	Section A.2.6
cfg save	Saves configuration data in EEPROM	Section A.2.7
cfg sernum	Assign a serial number to a component	Section A.2.8
cfg hw_version	Assign a hardware version to a component	Section A.2.9
cfg hw_revision	Assign a hardware revision to a component	Section A.2.10
cfg ipmi	Send IPMI command	Section A.2.11
cfg verify	Verify firmware components	Section A.2.12
cksum	Show POSIX checksum for a specified range of memory	Section A.2.13
comment	A comment indicator in RedBoot* scripts	Section A.2.14
conditional	Conditionally perform a command if a recent "load" or "fis load" command succeeds	Section A.2.15
cpld	Set of sub-commands to update/change CPLDs using an Xmodem connection and RedBoot support (a Lattice cable is not required).	Section A.2.16
deskew	Manage SRAM deskew settings	Section A.2.17
dump	Display memory contents	Section A.2.18
eeprom_cksum	Calculate and write EEPROM checksum	Section A.2.19
eth linkDown	Bring down an Ethernet link on the Base Interface	Section A.2.20
eth linkUp	Bring up an Ethernet link on the Base Interface	Section A.2.21
eth list	Display available Ethernet ports	Section A.2.22
eth select	Change default Ethernet port	Section A.2.23
exec	Execute an image with MMU off	Section A.2.24
memfill	Fill memory	Section A.2.25
fis create	Create an image in the Flash Image System (FIS) directory	Section A.2.26

Table 67. Boot Monitor console commands (Continued)

Command	Description	Refer to
fis delete	Delete an image in the FIS directory	Section A.2.27
fis erase	Erase a portion of flash memory	Section A.2.28
fis free	Display areas of flash memory that are currently not in use	Section A.2.29
fis init	Initialize the Flash Image System (FIS)	Section A.2.30
fis list	List images currently available in the FIS	Section A.2.31
fis load	Transfer an image from flash to RAM	Section A.2.32
fis lock	Write-protect (lock) a portion of flash memory	Section A.2.33
fis unlock	Unlock a portion of flash memory	Section A.2.34
fis write	Write data from memory to flash	Section A.2.35
fconfig	Manage the configuration kept in flash memory	Section A.2.36
fru addmrecord	Creates a new record in the specified FRU **Note**: Available on customized boards only.	Section A.2.37
fru delmrecord	Deletes the specified record from the specified FRU **Note**: Available on customized boards only.	Section A.2.38
fru mrecord	List FRU records, dump the content of a FRU, or modify a record in a FRU **Note**: Available on customized boards only.	Section A.2.39
go	Execute code at a specified location	Section A.2.40
help	Display help information	Section A.2.41
ipmi_event	Add an event to the system log	Section A.2.42
load	Download programs or data	Section A.2.43
net init	Initialize a debug slow port Ethernet controller	Section A.2.44
pci 8read	Read 8-bit wide data from the configuration space of the specified device	Section A.2.45
pci 8write	Write 8-bit wide data to the configuration space of the specified device	Section A.2.46
pci 16read	Read 16-bit wide data from the configuration space of the specified device	Section A.2.47
pci 16write	Write 16-bit wide data to the configuration space of the specified device	Section A.2.48
pci 32read	Read 32-bit wide data from the configuration space of the specified device	Section A.2.49
pci 32write	Write 32-bit wide data to the configuration space of the specified device	Section A.2.50
pci cfgshow	Read and show a configuration space of the specified device. Shows entire PCI configuration if no arguments are provided.	Section A.2.51
pci map	Show the result of the PCI bus scanning and base address (BAR) association process	Section A.2.52
pci reg show	Show internal PCI controller registers of the Intel® IXP2850 network processor	Section A.2.53
pci scan	Scan the PCI bus for devices	Section A.2.54
ping	Verify network connectivity	Section A.2.55
post all	Perform and show all results of the Power On Self Test (POST)	Section A.2.56
post bmc	Perform and show the results of the IPMI-Board Management Controller communication tests	Section A.2.57

Table 67. Boot Monitor console commands (Continued)

Command	Description	Refer to
post config	Set the POST verbose level and internal UART baud rate	Section A.2.58
post ethpci	Perform and show the results of the PCI Ethernet (Intel® 82546 Dual Port Gigabit Ethernet Controller) tests	Section A.2.59
post ethslow	Perform and show the results of the slow port Ethernet tests	Section A.2.60
post gpio	Perform and show the results of the General Purpose I/0 (GPIO) line verification tests	Section A.2.61
post i2c	Perform and show the results of the I^2C device tests	Section A.2.62
post interrupt	Perform and show the results of the interrupt controller verification tests	Section A.2.63
post led	Perform and show the results of the LED tests	Section A.2.64
post mem	Perform and show the results of the memory tests	Section A.2.65
post mezz	Perform and show the results of the Intel® IXF1104 and Gigabit Ethernet mezzanine verification tests	Section A.2.66
post msf	Perform and show the results of the Media Switch Fabric (MSF) access tests	Section A.2.67
post pci	Perform and show the results of the PCI tests	Section A.2.68
post slowport	Perform and show the results of the slow port verification tests	Section A.2.69
post status	Print current POST status table	Section A.2.70
post telecom_clock	Perform telecom clock tests	Section A.2.71
post uart	Perform and show the results of the UART tests	Section A.2.72
post ueng	Perform and show the results of the microengine register tests	Section A.2.73
post xscale	Perform and show the results of the Intel XScale® core verification tests	Section A.2.74
register 8read	Read 8-bit wide data from memory, SFR or slow port space of the IXP2850 network processor	Section A.2.75
register 8write	Write 8-bit wide data to memory, SFR or slow port space of IXP2850 network processor	Section A.2.76
register 16read	Read 16-bit wide data from memory, SFR or slow port space of the IXP2850 network processor	Section A.2.77
register 16write	Write 16-bit wide data to memory, SFR or slow port space of IXP2850 network processor	Section A.2.78
register 32read	Read 32-bit wide data from memory, SFR or slow port space of the IXP2850 network processor	Section A.2.79
register 32write	Write 32-bit wide data to memory, SFR or slow port space of IXP2850 network processor	Section A.2.80
reset	Reboot the board	Section A.2.81
sensor read	Read sensors	Section A.2.82
soft_reset boot	Soft reset	Section A.2.83
soft_reset check	Diagnostic command to check SRAM	Section A.2.84
soft_reset fill	Diagnostic command to fill SRAM with a suitable pattern	Section A.2.85
ssu	Safe upgrade of Network Processor (NP) flash partitions	Section A.2.86
ssu bootmode	Set the boot mode. Not intended for use with IXB2850 boards.	Section A.2.87
ssu diag	A password-protected service command used to diagnose problems with SSU functionality. Not intended for customer use.	Section A.2.88

Table 67. Boot Monitor console commands (Continued)

Command	Description	Refer to
ssu exec	Run the previously loaded Linux kernel image	Section A.2.89
ssu init	Initialize all the SSU structures required for SSU operation	Section A.2.90
ssu list	Display SSU-specific information about the SSU images stored on flash	Section A.2.91
ssu load	Load the active Linux kernel image from flash	Section A.2.92
ssu oneshot	Set/reset execution of the specified image type in “safe” mode for next boot verification	Section A.2.93
ssu upgrade	Download and write the specified SSU image using the Xmodem protocol or TFTP	Section A.2.94
ssu validate	Validate the image running in “safe” mode and set as the primary image	Section A.2.95
syslog	Manage the system log	Section A.2.96
trb	Switch between RedBoot images	Section A.2.97
update bmc	Update the Board Management Controller (BMC) firmware	Section A.2.98
update fru	Update a Field Replaceable Unit’s (FRU’s) information	Section A.2.99
update sdr	Store new SDR information for a given configuration in EEPROM	Section A.2.100
version	Display RedBoot version	Section A.2.101
watchdog	Turn the watchdog on or off	Section A.2.102

A.2 Command Descriptions

The console commands are described in the following subsections. Each subsection describes the function of the command, the command syntax, descriptions of each parameter that can be included in the command and an example showing how the command is used.

A.2.1 alias

Manage aliases kept in flash memory.

Syntax

```
alias name [value]
```

Parameters

- *name* - Name for the alias
- *value* - Value for the alias (if this parameter is not given, the current value of the alias is displayed)

Example

```
alias test “this is a test machine”
```

A.2.2 alignment

Manage static alignment for Media Switch Fabric (MSF) configuration.

Syntax

```
alignment read | write
```

Example

```
alignment read
```

A.2.3 baudrate

Set or query the system console baud rate.

Syntax

```
baudrate [-b <rate>]
```

Parameters

- *-b* - Set the given console baud rate. If the *-b* parameter is not specified, the command reads the existing baud rate. The following are valid values:
  - 600
  - 1200
  - 2400
  - 4800
  - 9600
  - 14400
  - 19200
  - 38400
  - 57600
  - 115200

Example

```
baudrate
```

A.2.4 cache

Manage machine caches (data and instruction).

Syntax

```
cache [on | off]
```

Parameters

- *on* - Enables the data and instruction caches
- *off* - Disables the data and instruction caches

Note: If no parameter is specified, the current cache settings are displayed.

Example

```
cache on

cache
```

A.2.5 cfg mac

Set the MAC address value for the specified Ethernet port.

Syntax

```
cfg mac -c <components> -i <interface_type> -p <port_number> -v <value>
```

Parameters

- *-c <component>* - The component to change the MAC address on:
  - *bb* - Baseboard
  - *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - *db2* - Media mezzanine card 2
  - *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - *mic2* - Media interface card 2
  - *fic* - Fabric interface card
  - *npmod* - Network processor module
  - *ap* - adjunct processor card
- *-i <interface_type>* - The interface type as defined in the FRU:
  - 0x0 – Debug interface
  - 0x3 – 82546 Dual Port Gigabit Ethernet Controller Fiber/Copper
  - 0x6 – IXF1104 Fiber/Copper
  - 0xb – NPU internal debug interface #1
  - 0xb – NPU internal debug interface #2
  - 0xb – NPU internal Gigabit Ethernet interface #1
  - 0xb – NPU internal Gigabit Ethernet interface #2
- *-p <port_number>* - The port number
- *-v <value>* - The MAC address to be assigned (in the format: AB:CD:EF:GH:IJ:KL)

Example

```
cfg mac -c bb -i 0 -p 0 -v 11:22:33:44:55:66
```

A.2.6 cfg read

Show each configuration parameter (in sequence) along with the parameter's current setting. Issuing this command without a parameter displays the configuration for all components.

Note: This command parses custom OEM records (that is, records with id=1 or id=2, both version 0.0). All unknown records are dumped in binary format.

Syntax

```
cfg read [-c <component>]
```

Parameters

- *-c <component>* component to read the configuration from:
  - *bb* - Baseboard

  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *ap* - Adjunct processor card
  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module

Example

```
cfg read -c db1
```

A.2.7 cfg save

Saves the current configuration in EEPROM.

Syntax

```
cfg save [-c <component>]
```

Parameters

- *-c <component>* - Component to save configuration of:
  - — *bb* - Baseboard
  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module
  - — *ap* - Adjunct processor card

Example

```
cfg save -c bb
```

A.2.8 cfg sernum

Assign a serial number to the specified component.

Syntax

```
cfg sernum -c<component> -v<sernum>
```

Parameters

- *-c <component>* - The component to change/assign a serial number to:
  - — *bb* - Baseboard
  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *ap* - Adjunct processor card

  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module
- *-v <sernum>* - The serial number (up to 32 alphanumeric characters)

Example

```
cfg sernum -c db1 -v 123456
```

A.2.9 cfg hw_version

Assign a hardware version to the specified component.

Syntax

```
cfg hw_version -c<component> -v<hw_version>
```

Parameters

- *-c <component>* - The component to change/assign the serial number to:
  - — *bb* - Baseboard
  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *ap* - Adjunct processor card
  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module
- *-v <hw_version>* - The hardware version written as nine alphanumerical characters in the format: AA-BB-CCC

Example

```
cfg hw_version -c db1 -v AA-BB-CCC
```

A.2.10 cfg hw_revision

Assigns a hardware revision to the specified component. The revision is the last three characters of the hardware version set by the cfg hw_version command.

Syntax

```
cfg hw_revision -c<component> -v<hw_revision>
```

Parameters

- *-c <component>* - The component to change/assign a serial number to:
  - — *bb* - Baseboard
  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *ap* - Adjunct processor card

  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module
- *-v <hw_revision>* - The serial number written as three alphanumerical characters (CCC)

Example

```
cfg hw_revision -c db1 -v CCC
```

A.2.11 cfg ipmi

Send IPMI command.

Syntax

```
cfg ipmi <net function> <command> <opt data>
```

Parameters

- *<net function>* - Net function (one byte)
- *<command>* - Command (one byte)
- *<opt data>* - Optional data

Example

```
cfg ipmi 2c 0f 00 01
```

A.2.12 cfg verify

Verification of firmware components.

Syntax

```
cfg verify
```

Example

```
cfg verify
```

A.2.13 cksum

Count and show a 32-bit POSIX checksum for the specified range of memory. This command is used to verify and compare large numbers of memory locations.

Syntax

```
cksum -b <location> -l <length>
```

Parameters

- *-b <location>* - The start of the memory range to compute a checksum. It must be a four byte aligned address.
- *-l <length>* - The length of the memory to compute. It must be at least four bytes and a number that is a multiple of four.

Example

```
cksum -b 0xa000000 -l 0x1000
```

A.2.14 comment

A comment indicator in a RedBoot script.

Syntax

```
comment <comment>
```

Example

```
comment This is a comment in a RedBoot script.
```

A.2.15 conditional

Conditionally perform a command only if the recent load or fis load command succeeded. Otherwise, skip the command.

Syntax

```
conditional <command>
```

Example

```
fis load -s "Linux kernel"
conditional exec
load -m tftp .....
```

A.2.16 cpld

A set of sub-commands to verify the CPLD version (a Lattice* cable is not required). For the required DIP switch settings, see Section 11.3.1, “Local Software Upgrade Procedure” on page 137.

The following sub-commands are available:

- *cpld version -c <chain_number>*
  Read the version of all CPLDs in the selected chain and verify the CPLD version against the RedBoot version.
- *cpld update -c <chain_number> -b <data_address>*

Parameters

- *-c <chain_number>* Valid values are:
  - 0 - Baseboard and NP module CPLD
  - 1 - Baseboard and BMC CPLD
  - 2 - Media mezzanine card 1 (not used on IXB2850 boards)
  - 3 - Media mezzanine card 2
  - 4 - Media interface card 1 (not used on IXB2850 boards)
  - 5 - Media interface card 2
- *-b <data_address>* - Base address

Example

```
cpld version -c 0

cpld update -c 0 -b 0x10000000
```

A.2.17 deskew

Manage SRAM deskew settings. The following three sub-commands are available:

- *deskew read* - Read and show the current dll/deskew values from EEPROM
- *deskew restore* - Restore the current dll/deskew values in EEPROM to factory dll/deskew values
- *deskew write* - Write the current dll/deskew values to EEPROM

Syntax

```
deskew read | restore | write
```

Example

```
deskew restore
```

A.2.18 dump

Display a range of memory.

Syntax

```
dump -b <location> [-l <length>]
```

Parameters

- *-d <location>* - Location in memory for start of data
- *-l <length>* - Length of data to be displayed (the default is 32)

Example

```
dump -b 0x100000
```

A.2.19 eeprom_cksum

Calculate and write EEPROM checksum.

Syntax

```
eeprom_cksum
```

Example

```
eeprom_cksum
```

A.2.20 eth linkDown

Bring down an Ethernet link on a port on the Base Interface.

Syntax

```
eth linkDown <interface_name>
```

Parameters

- <*interface_name*> - The name of the Ethernet port

Example

```
eth linkUp bpl0
```

A.2.21 eth linkUp

Bring up an Ethernet link on a port on the Base Interface.

Syntax

```
eth linkUp <interface_name>
```

Parameters

- <*interface_name*> - The name of the Ethernet port

Example

```
eth linkUp bpl0
```

A.2.22 eth list

List the available Ethernet ports.

Syntax

```
eth list
```

Example

```
eth list
```

A.2.23 eth select

Change the default Ethernet port.

Syntax

```
eth select <port name>
```

Parameters

- <*port name*> - The name of the Ethernet port

Example

```
eth select “slowport IXMB28x1”
```

A.2.24 exec

Execute an image with MMU off.

Syntax

```
exec [-w <timeout>] [-b <load addr> [-l <length>] [-r <ramdisk addr> [-s <ramdisk
length>] [-c <kernel command line>] [<entry_point>] [-p <kernel params address>]
```

Parameters

- *-w <timeout>* - Time to wait before execution (default value is 0)
- *-b <load_addr>* - Address in memory of the kernel Linux image
- *-l <length>* - Length of Linux kernel image
- *-r <ramdisk addr>* - Address in memory of “initrd” style RAMdisk passed to Linux kernel
- *-s <ramdisk length>* - Length of RAMdisk image passed to Linux kernel
- *-c <kernel command line>* - Command line to pass to Linux kernel
- *<entry_point>* - Starting address for Linux kernel execution
- *-p <kernel_params_address>* - Kernel parameters address

Example

```
exec -w 1 -c "console=ttyS0,115200 devfs=mount rw noinitrd root=/dev/nfs
mem=443M@0x0 pci=firmware ip=bootp" -p 0x100
```

A.2.25 memfill

Fill memory from <addr1> to <addr2> with the specified 32-bit <value>.

Syntax

```
memfill -b <addr1> -e <addr2> -v <value>
```

Parameters

- *-b <addr1>* - Starting address of the memory range to fill. It must be a four byte aligned address.
- *-e <addr2>* - Ending address of the memory range to fill. It must be a four byte aligned address.
- *-v <value>* - A 32-bit value used to fill the memory range.

Example

```
memfill -b 0xa0000000 -e 0xa100000 -v 0x12345678
```

A.2.26 fis create

Create an image in the Flash Image System (FIS) directory. The data for the image must exist in RAM memory before the copy. Use the load command to load the file into RAM.

Syntax

```
fis create [-b <data_address>] -l <length> [-f <flash_address>] [-e <entry>] [-r
<relocation_address>] [-s <data_length>] [-n] name
```

Parameters

- *-b <data_address>* - Address of data to be written to flash
- *-l <length>* - Length of flash area to be occupied (this value may be calculated based on the load command)
- *-f <flash_address>* - Address of flash area to be occupied (the default value is the first available block that is large enough)

- *-e <entry>* - Entry address for the executable image
- *-r <relocation_address>* - Address where the image should be relocated to by the load command
- *-s <data_length>* - Actual length of data written to the image
- *-n* - No image data written to flash; only the FIS directory is updated
- *name* - Name of the flash image

Example

```
fis create -b 0x1000000 -l 0x1e0000 -f 0x90020000 RedBoot
```

A.2.27 fis delete

Remove an image from the FIS. This image is erased and the name is removed from the FIS directory.

Syntax

```
fis delete name
```

Parameters

- *name* - Name of the flash image to be deleted

Example

```
fis delete diag
```

A.2.28 fis erase

Erase a portion of flash memory.

Syntax

```
fis erase -f <flash_address> -l <length>
```

Parameters

- *-f <flash_address>* - Starting address of the flash memory
- *-l <length>* - Length of the flash area to be erased

Example

```
fis erase -f 0x90500000  -l 0x100000
```

A.2.29 fis free

Displays areas of flash memory that are currently not in use. A block containing non-erased content is considered in use.

Syntax

```
fis free
```

Example

```
fis free
```

A.2.30 fis init

Initialize the Flash Image System (FIS).

Caution: Execution of this command causes loss of information previously stored in the FIS.

Syntax

```
fis init [-f] [-d]
```

Parameters

- *-f* - All blocks of flash memory (except for the boot blocks) are erased as part of the initialization procedure.
- *-d* - Create a dual bootmonitor (RedBoot) image.

Example

```
fis init
```

A.2.31 fis list

List the images currently available in the FIS.

Syntax

```
fis list [-c] [-d]
```

Parameters

- *-c* - Show the image checksum instead of memory address
- *-d* - Show the image data length instead of the amount of flash occupied by the image

Example

```
fis list -d
```

A.2.32 fis load

Transfer an image from flash to RAM. After the image has been loaded, it may be executed using the go command.

Syntax

```
fis load [-b <load_address>] [-c] [-d] name [-w] [-s]
```

Parameters

- *-b <load_address>* - Address to which the image should be loaded (the default value is the address associated with the image)
- *-c* - Compute and print the checksum of the image after it has been loaded
- *-d* - Decompress gzipped image while copying it to RAM
- *name* - The name of the file as shown in the FIS directory
- *-w* - Read the flash in 32-bit mode

- -s - Verify SSU header and image integrity. If the -s option is specified and the loaded image is not a valid SSU image (that is, it has an incorrect checksum, an incorrect SSU header or no SSU header), the load command fails.

Example

```
fis load -b 0x1000000 Diagnostics
```

A.2.33 fis lock

Write-protect (lock) a portion of flash memory to prevent accidental overwriting of images. In order to make modifications to the flash, the fis unlock command must be issued.

Syntax

```
fis lock -f <flash_address> -l <length>
```

Parameters

- *-f <flash_address>* - Starting address of the flash memory
- *-l <length>* - Length of the flash area to be locked

Example

```
fis lock -f 0x90020000  -l 0x1E0000
```

A.2.34 fis unlock

Unlock a portion of flash memory. This command is used to disable the write-protection enabled by the fis lock command.

Syntax

```
fis unlock -f <flash_address> -l <length>
```

Parameters

- *-f <flash_address>* - Starting address of the flash memory
- *-l <length>* - Length of flash area to be unlocked

Example

```
fis unlock -f 0x90020000  -l 0x1E0000
```

A.2.35 fis write

Write data from memory to flash.

Syntax

```
fis write -b <data_address> -l <length> -f <flash_address>
```

Parameters

- *-b <data_address>* - Address of the data to be written to flash
- *-l <length>* - Length of the flash area to be occupied (this value may be calculated based on the load command)

- *-f <flash_address>* - Address of the flash area to be occupied (the default is the first available block that is large enough to store the image)

Example

```
fis write -b 0x280000  -l 0x100000  -f 0x90200000
```

A.2.36 fconfig

Manage configuration kept in flash memory. When the command is invoked, a prompt appears for each value. The value must be typed in full. Pressing Enter keeps the current value.

Note: When executing the fconfig command without the *-d* option, you must backspace over the displayed value and enter a new value.

Syntax

```
fconfig [-i] [-l] [-n] [-f] | nickname [value]
```

Parameters

- *-i* - Reset the configuration database to the default state
- *-l* - List the current configuration without changing the values
- *-n* - Display the parameter “nicknames” rather than the full names
- *-f* - Display the full parameter names
- *nickname <value>* - Change/display only this parameter

Example

```
fconfig –l

fconfig

fconfig bootp
```

A.2.37 fru addmrecord

Creates a new record in a given FRU.

Syntax

```
fru addmrecord -c <component> -r <record type> -m <mfg id> -t <record id> -l
<length>
```

Parameters

- *-c <component>* - The FRU to operate on. Valid values are:
  - *bb* - Baseboard
  - *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - *db2* - Media mezzanine card 2
  - *ap* - Adjunct processor card
  - *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - *mic2* - Media interface card 2
  - *fic* - Fabric interface card

  - — *npmod* - Network processor module
  - — *rtm* - Rear transition module
  - — *ap* - Adjunct Processor (AP)
- *-r <record type>* - New record type (for example, 0xC0 for OEM records)
- *-m <mfg id>* - OEM manufacturer's code; three bytes wide (for example, 0x12ab56 gives the manufacturer's code byte string "56 ab 12")
- *-t <record id>* - OEM record ID
- *-l <length>* - Total record length, including record header. The total length must be >= 9 (which is the combined length of the record header and the OEM record header)

Note: The cfg save command must be issued to save any changes made to non-volatile memory.

Example

```
fru addrecord -c bb -r 0xC0 -m 0x23 -t 0x01 48
```

A.2.38 fru delmrecord

Deletes the specified record from the specified FRU.

Syntax

```
fru delmrecord -c <component> -n <mrec_index>
```

Parameters

- *-c <component>* - The FRU to operate on. Valid values are:
  - — *bb* - Baseboard
  - — *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - — *db2* - Media mezzanine card 2
  - — *ap* - Adjunct processor card
  - — *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - — *mic2* - Media interface card 2
  - — *fic* - Fabric interface card
  - — *npmod* - Network processor module
  - — *rtm* - Rear transition module
  - — *ap* - Adjunct Processor (AP)
- *-n <mrec_index>* - Index of the multirecord to be updated; counted from 1

Note: The cfg save command must be issued to save any changes made to non-volatile memory. The command is rejected if the specified record is the last remaining record in the FRU.

Example

```
fru delmrecord -c bb -n 5
```

A.2.39 fru mrecord

Enables the following operations on FRU records:

- List the records in the specified FRU
- Dump the binary content of the specified record of a FRU
- Modify a field in the specified record of a FRU

Syntax

```
fru mrecord -c <component>

fru mrecord -c <component> -n <mrec_index>

fru mrecord -c <componnet> -n <mrec_index> -o <offset> [-s <size>] [-a] data
```

Note: The first command syntax is used to list all the records in a given FRU. The second command syntax is used to dump the binary content of a given record. The last command syntax is used when modifying record content.

Parameters

- *-c <component>* - The FRU to operate on. Valid values are:
  - *bb* - Baseboard
  - *db1* - Media mezzanine card 1 (not used on IXB2850 boards)
  - *db2* - Media mezzanine card 2
  - *ap* - Adjunct processor card
  - *mic1* - Media interface card 1 (not used on IXB2850 boards)
  - *mic2* - Media interface card 2
  - *fic* - Fabric interface card
  - *npmod* - Network processor module
  - *rtm* - Rear transition module
  - *ap* - Adjunct Processor (AP)
- *-n <mrec_index>* - Index of the multirecord to be updated; counted from 1
- *-o <offset>* - Offset from the beginning of the record data area where new data should be placed. The offset is counted from the beginning of the record (which contains the record header). Offset values less than five (<5) are not allowed to prevent changes to the record header.
- *-s <size>* - The size of an existing field that is being modified. If this option is not specified, the size of the existing field is equal to the size of the new data, otherwise the data is filled with spaces (in ASCII mode) or zeros (in HEX mode).
- *-a* - Specifies that the data should be written as an ASCII string, otherwise, HEX mode is assumed.
- *-e* - Expert mode, allows modification of all record types, not just 0xC0 OEM type records.
- *-v data* - New data to be placed in the FRU record:
  - In ASCII mode, it is possible to write the data directly. If spaces are required, the text must be enclosed in parentheses.
  - In HEX mode, the format is: 00:11:22:33: ... :EE:FF

Note: When all FRU record modifications are completed (using one or more command executions), the `cfg save` command must be issued to store the changes in non-volatile memory.

Example

```
fru mrecord -c bb -n 6 -o 19 -a -v "  01"
```

A.2.40 go

Execute the code at a specified location.

Syntax

```
go [-w <timeout>] [entry]
```

Parameters

- *-w* <*timeout*> - Executes code after a given period of time
- *entry* - The memory address of the code to be executed

Example

```
go 0x1000000
```

A.2.41 help

Display help on available commands.

Syntax

```
help <topic>
```

Parameters

- <*topic*> - The command to display help for

Example

```
help go
```

A.2.42 ipmi_event

Sends a booting-related IPMI message to the BMC.

Syntax

```
ipmi_event boot -t {npu_boot|post_result|img_error|boot_complete} [-i <info>]

ipmi_event post -t <post_type> [-i <info>]

ipmi_event diag -t <diag_type> [-i <info>]
```

Parameters

- *boot*, *post*, *diag* - Indicates the event class. See Section 9.4.7, “IPMI Event Messaging” on page 115.
- *-t* - Event data. One of the following:
  - {npu_boot|post_result|img_error|boot_complete}. See “data 2” in Table 52, “IPMI event data for boot class” on page 116.

  - — <post_type>. See “data 2” in Table 53, “IPMI event data for the POST class” on page 116.
  - — <diag_type>. See “data 2” in Table 54, “IPMI event data for the diagnostics class” on page 117.
- *-i <info>* - Additional event data (data 3). See “data 3” in Table 52, Table 53 and Table 54 depending on the event class.

A.2.43 load

Download programs or data to the RedBoot platform.

Syntax

```
load [-v] [-d] [-r] [-m <protocol>] [-h <server_IP_address>] [-b <location>]
[file_name] [-s]
```

Parameters

- *-v* - Display a small spinner, while the download is in progress
- *-d* - Decompress the data stream
- *-r* - Specify raw (or binary) data
- *-m* - Specify the protocol to be used for transfer:
  - — tftp
  - — xmodem
- *-h <server_IP_address>* - The IP address of TFTP server
- *-b <location>* - The address in memory to load data to
- *<file_name>* - The name of the file to be transferred
- -s - Verify SSU header and image integrity. If the -s option is specified and the loaded image is not a valid SSU image (that is, it has an incorrect checksum, an incorrect SSU header or no SSU header), the load command fails. The intention is to prevent the execution of a damaged diagnostics image.

Example

```
load -r -m xmodem -b 0x1000000
```

```
load -s ...
conditional go
```

A.2.44 net init

Reinitialize a debug slow port Ethernet controller after changes in the static IP address or after changing BOOTP activity (enable/disable). After this command has been issued, the debug Ethernet is ready to download code.

Syntax

```
net init
```

Example

```
net init
```

A.2.45 pci 8read

Read 8-bit wide data from the configuration space of the device located on <*bus*>:<*device*>:<*function*> with the given <*offset*> from the start of the configuration space. This command remembers the last valid configuration for <*bus*>, <*device*> and <*function*>. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the <*offset*> only.

Syntax

```
pci 8read -b <bus> -d <device> -f <function> -o <offset>
```

Parameters

- *-b* <*bus*> - The PCI bus number where a device is installed
- *-d* <*device*> - The PCI bus device location where a device is installed
- *-f* <*function*> - The PCI bus function location where a device is located
- *-o* <*offset*> - The offset from the start of the configuration space of the device

Example

```
pci 8read -b 1 -d 3 -f 2 -o 0x14
```

A.2.46 pci 8write

Write 8-bit wide data to the configuration space of the device located on <*bus*>:<*device*>:<*function*> with the given <*offset*> from the start of the configuration space. This command remembers the last valid configuration for <*bus*>, <*device*> and <*function*>. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the <*offset*> and <*value*> only.

Syntax

```
pci 8write -b <bus> -d <device> -f <function> -o <offset> -v <value>
```

Parameters

- *-b* <*bus*> - The PCI bus number where a device is installed
- *-d* <*device*> - The PCI bus device location where a device is installed
- *-f* <*function*> - The PCI bus function location where a device is located
- *-o* <*offset*> - The offset from the start of the configuration space of the device
- *-v* <*value*> - The value that should be written to the configuration space

Example

```
pci 8write -b 1 -d 3 -f 2 -o 0x14 -v 0x1234
```

A.2.47 pci 16read

Read 16-bit wide data from the configuration space of the device located on <*bus*>:<*device*>:<*function*> with the given <*offset*> from the start of the configuration space. This command remembers the last valid configuration for <*bus*>, <*device*> and <*function*>. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the <*offset*> parameter only.

Syntax

```
pci 16read -b <bus> -d <device> -f <function> -o <offset>
```

Parameters

- *-b <bus>* - The PCI bus number where a device is installed
- *-d <device>* - The PCI bus device location where a device is installed
- *-f <function>* - The PCI bus function location where a device is located
- *-o <offset>* - The offset from the start of the configuration space of the device

Example

```
pci 16read -b 1 -d 3 -f 2 -o 0x14
```

A.2.48 pci 16write

Write 16-bit wide data to the configuration space of the device located on *<bus>:<device>:<function>* with the given *<offset>* from start of the configuration space. This command remembers the last valid configuration for *<bus>*, *<device>* and *<function>*. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the *<offset>* and *<value>* parameters only.

Syntax

```
pci 16write -b <bus> -d <device> -f <function> -o <offset> -v <value>
```

Parameters

- *-b <bus>* - The PCI bus number where a device is installed
- *-d <device>* - The PCI bus device location where a device is installed
- *-f <function>* - The PCI bus function location where a device is located
- *-o <offset>* - The offset from the start of the configuration space of the device
- *-v <value>* - The value that should be written to the configuration space

Example

```
pci 16write -b 1 -d 3 -f 2 -o ox14 -v 0x1234
```

A.2.49 pci 32read

Read 32-bit wide data from the configuration space of the device located on *<bus>:<device>:<function>* with the given *<offset>* from the start of the configuration space. This command remembers the last valid configuration for *<bus>*, *<device>* and *<function>*. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the *<offset>* only.

Syntax

```
pci 32read -b <bus> -d <device> -f <function> -o <offset>
```

Parameters

- *-b <bus>* - The PCI bus number where a device is installed
- *-d <device>* - The PCI bus device location where a device is installed

- *-f <function>* - The PCI bus function location where a device is located
- *-o <offset>* - The offset from the start of the configuration space of the device

Example

```
pci 32read -b 1 -d 3 -f 2 -o 0x14
```

A.2.50 pci 32write

Write 32-bit wide data to the configuration space of the device located on *<bus>:<device>:<function>* with the given *<offset>* from start of the configuration space. This command remembers the last valid configuration for *<bus>*, *<device>* and *<function>*. Therefore, it is only necessary to specify all the parameters when accessing the configuration space of the given PCI device for the first time. Subsequent access to the same device requires the *<offset>* and *<value>* parameters only.

Syntax

```
pci 32write -b <bus> -d <device> -f <function> -o <offset> -v <value>
```

Parameters

- *-b <bus>* - The PCI bus number where a device is installed
- *-d <device>* - The PCI bus device location where a device is installed
- *-f <function>* - The PCI bus function location where a device is located
- *-o <offset>* - The offset from the start of the configuration space of the device
- *-v <value>* - The value that should be written to the configuration space

Example

```
pci 32write -b 1 -d 3 -f 2 -o 0x14 -v 0x12345678
```

A.2.51 pci cfgshow

Read and show the configuration space of the device located on *<bus>:<device>:<function>* or show the entire PCI configuration (if no parameters are provided).

Note: The configuration for the network processor is not displayed. Use the pci reg show command to display network processor configuration information.

Syntax

```
pci cfgshow -b <bus> -d <device> -f <function>
```

Parameters

- *-b <bus>* - The PCI bus number where a device is installed
- *-d <device>* - The PCI bus device location where a device is installed
- *-f <function>* - The PCI bus function location where a device is located

Example

```
pci cfgshow -b 1 -d 3 -f 2
```

A.2.52 pci map

Show the result of the PCI bus scanning and base address (BAR) association process.

Syntax

```
pci map
```

Example

```
pci map
```

A.2.53 pci reg show

Show internal PCI controller registers of the IXP2850 network processor.

Syntax

```
pci reg show
```

Example

```
pci reg show
```

A.2.54 pci scan

Scan the PCI bus for any devices and register their parameters. This data can then be viewed using the pci map command.

Syntax

```
pci scan
```

Example

```
pci scan
```

A.2.55 ping

Verify network connectivity.

Syntax

```
ping [-v] [-i <local_IP_address>] [-l <length>] [-n <count>] [-t <timeout>] [-r
<rate>] -h <server_IP_address>
```

Parameters

- *-v* - Verbose mode, display information on every packet
- *-i <local_IP_address>* - The IP address RedBoot should use. The default value is set by the fconfig command or obtained using the BOOTP protocol.
- *-l <length>* - The length of the ICMP data payload
- *-n <count>* - The number of packets to be sent
- *-t <timeout>* - The amount of time to wait for the round-trip to complete (in milliseconds)
- *-r <rate>* - The time between successive sends (in milliseconds)
- *-h <server_IP_address>* - The address of the host to contact

Example

```
ping -h 192.168.1.100 -n 100
```

A.2.56 post all

Perform and show the results of the Power On Self Test (POST) process. A hard-coded sequence of POST tests is always performed when the board is booted up. This cannot be change. This command allows the repetition of the initial POST for the board. Tests performed by this command are:

- UART ports
- RDRAM/SDRAM
- QDR SRAM
- BBSRAM
- PCI bus
- BMC - ID EEPROMS using BMC communication paths
- Debug Ethernet port
- IXP28x0 microengines
- Media and backplane device
- Slow port bus
- Interrupt mechanisms
- XScale core
- Media Switch Fabric
- GPIO
- I^2C bus
- LEDs
- IXF1104
- Telecom clock
- MB GBE

Syntax

```
post all
```

A.2.57 post bmc

Perform IPMI Board Management Controller (BMC) communication tests. These tests check for the presence of a BMC and reads the EEPROM device ID.

Syntax

```
post bmc -h | -a | {-f | -i | -s}
```

Parameters

- *-f* - Check if the BMC is present on the board
- *-i* - Read the ID EEPROM from the board
- *-s* - BMC show ID EEPROM
- *-a* - Run all BMC tests

- *-h* - Display help screen

Example

```
post bmc -a
```

A.2.58 post config

Set the verbose output for POST tests and internal UART baud rate.

Syntax

```
post config -h | -v <level> -b <baud rate> -d
```

Parameters

- *-h* - Display help screen
- *-v <level>* - Specify verbose level; possible values are:
  - 0 - No console output
  - 1 - Minimal console output. This is the default value.
  - 2 - Individual errors from each test are output to the console
  - 3 - Individual warnings from each test are output to the console
  - 4 - Prints debug information
  - 5 - Prints all debug information
- *-b <baud>* - Changes the internal UART board rate
- *-d* - Return the above values to their default settings

Example

```
post config -v 3
```

A.2.59 post ethpci

Perform PCI Ethernet tests. These tests check for the presence of an Ethernet chip, registers access and sends/receives test packets.

Syntax

```
post ethpci -h | -a | -f -r -i -m -e -n <port>
```

Parameters

- *-f* - Checks if an Ethernet device is present on board
- *-i* - Run Ethernet interrupt test
- *-r* - Run register access test
- *-m* - Run MAC (internal) loopback test
- *-e* - Run external system loopback test
- *-a* - Test all (except for external loopback test)
- *-h* - Display help screen
- *-n <port>* - Specify Ethernet PCI port number; one of the following:
  - 0 - Port 0
  - 1 - Port 1

— 2 - All ports

Example

```
post ethpci -a
```

A.2.60 post ethslow

Perform and show results of the slow port Ethernet tests. These tests check for the presence of an Ethernet chip, registers access and sends/receives test packets.

Syntax

```
post ethslow -h | -a | -f -r -i -e -m
```

Parameters

- *-f* - Check if an Ethernet device is present on the board
- *-i* - Run Ethernet interrupt test
- *-r* - Run register access test
- *-a* - Run all pre-defined slow port Ethernet tests
- *-m* - Run MAC (internal) system loopback test
- *-e* - Run external system loopback test
- *-h* - Display help screen

Example

```
post ethslow -a
```

A.2.61 post gpio

Perform GPIO line verification tests.

Syntax

```
post gpio -h | -a | -d -r -e -l
```

Parameters

- *-h* - Display help screen
- *-a* - Run all GPIO tests
- *-d* - Run GPIO detection test
- *-r* - Run GPIO registers test
- *-e* - Run GPIO edge detection test
- *-l* - Run GPIO level detection test

Example

```
post gpio -a
```

A.2.62 post i2c

Perform and show results of I^2C device test.

Syntax

```
post i2c -h | -a | -w -p -c -r -s -d <device> -b <address> -v <data/count>
```

Parameters

- *-h* - Display help screen
- *-a* - Run all I^2C read tests
- *-w* - Run I^2C write test
- *-p* - Run I^2C page write test
- *-c* - Run I^2C current read test
- *-r* - Run I^2C random read test
- *-s* - Run I^2C sequential test
- *-d <device>* - Specify an I^2C device
  - 0...7 - The device address on I^2C bus
- *-b <address>* - Specify an I^2C test address
  - 0...FFF - The address in the I^2C device (except the -s test)
- -v <data/count> - Specify test value
  - 0...FF - Data to write into I^2C device (for *-w*, *-c* and *-r* tests only)
  - 1...1000 - Count/size used in the test (for *-p* and *-s* tests only)

Example

```
post i2c -w -d 2 -b 0 -v 5A
```

A.2.63 post interrupt

Perform interrupt controller verification tests. The 125 micorsecond reference clock interrupt and simulation interrupts are checked.

Syntax

```
post interrupt -h | -a | -i -s
```

Parameters

- *-h* - Display help screen
- *-a* - Run all predefined interrupt tests
- *-i* - Test the125 micorsecond interrupt from the CPLD
- *-s* - Test interrupt simulation

Example

```
post interrupt -a
```

A.2.64 post led

Perform and show the results of the LED test.

Syntax

```
post led -h | -a | -o -r -y -g
```

Parameters

- *-h* - Display help screen
- *-a* - Run all LED tests
- *-o* - Run the orange LED test
- *- r* - Run the red LED test
- *-y* - Run the yellow LED test
- *-g* - Run the green LED test

Example

```
post led -a
```

A.2.65 post mem

Perform and show the results of the memory tests. Only one option (indicating the type of the test) can be used each time the command is executed.

Syntax

```
post mem -h | -a | -w <0 | 1> -c <5 | A> -i -d -b <start address> -l <length> -n
<loop number> -t <memory type>
```

Parameters

- *-w* - Run walking ones or walking zeros test:
  - 1 - Walking ones test
  - 0 - Walking zeroes test
- *-c* - Run known pattern tests:
  - 5 - 0x5a5a5a5a pattern test
  - A- 0xa5a5a5a5 pattern test
- *-a* - Run all pre-defined memory tests
- *-i* - Run incremental test
- *-d* - Run address bus test
- *-h* - Display help screen
- *-b <start address>* - An offset from memory to start address for test
- *-l <length>* - A memory length for testing
- *-n <loop number>* - The number of test to repeat
- *-t <memory type>* - Indicate the memory type to be tested:
  - SDRAM
  - QDRAM0
  - QDRAM1
  - QDRAM2
  - QDRAM3

Example

```
post mem -a -b 0x1000000 -l <length> -t SDRAM
```

A.2.66 post mezz

Perform IXF1104, Gigabit Ethernet mezzanine and Fabric Interface Card (FIC) test verification. The internal loopback tests are performed and a fixed number of frames are sent and received.

Syntax

```
post mezz -h | -a | -t <type> -p <ports> -d <duplex> -l <loopback>
```

Parameters

- *-h* - Display help screen
- *-a* - Run all mezzanine functionality tests
- *-t <type>* - Test type; valid values are:
  - *valbb* - IXF1104 on baseboard tests
  - *gbe* - Gigabit Ethernet mezzanine card tests
  - *fic* - Fabric Interface Card (FIC) tests
- *-p <port>* port number
  - *0* - all ports (1 to 4 for IXF1104 single port test types)
  - *0* - all ports (1 to 4 for media mezzanine card 1 Gigabit Ethernet test types (not used on IXB2850 boards, 5 to 8 for media mezzanine card 2 Gigabit Ethernet test types)
- *-d <duplex>* duplex mode (valid in IXF1104 tests only)
  - *h* - half-duplex mode
  - *f* - full-duplex mode
- *-l <loopback>* loopback type
  - *m* - MAC loopback
  - *p* - PHY system loopback for IXF1104 tests

Example

```
post mezz -t valbb -p 0 -d f -l p
```

A.2.67 post msf

Perform Media Switch Fabric (MSF) access tests.

Syntax

```
post msf -h -a -r -d
```

Parameters

- *-h* - Display help screen
- *-a* - Run all MSF tests
- *-r* - Run MSF registers test
- *-d* - Run MSF default tests

Example

```
post msf -r
```

A.2.68 post pci

Perform PCI tests that check for the presence of PCI devices such as an AP, 82546 Dual Port Gigabit Ethernet Controller and PCI bridges.

Syntax

```
post pci -h | -a | -f -i -b
```

Parameters

- *-f* - Check if all PCI devices are present on the board
- *-i* - Run PCI interrupt test
- *-b* - Number of 21154 PCI bridges present on board; 1 (default) or 2
- *-a* - Run all pre-defined PCI tests
- *-h* - Display help screen

Example

```
post pci -a
```

A.2.69 post slowport

Perform slow port verification tests. This test expects that the read value from <address> masked with <mask> is equal to the <value> specified in the test description.

Syntax

```
post slowport -h | -a | -b <address> -v <value> -m <mask>
```

Parameters

- *-h* - Display help screen
- *-b <address>* - Address of slow port to test
- *-v <value>* - Value to write to <address>
- *-m <mask>* - Pattern to mask <value> after reading
- *-a* - Run all pre-defined slow port tests

Example

```
post slowport -a 0xa100000 -v 0x6e32 -m 0xff
```

A.2.70 post status

Print the current POST status table.

Syntax

```
post status
```

Example

```
post status
```

A.2.71 post telecom_clock

Perform telecom clock tests.

Syntax

```
post telecom_clock -h | -a
```

Parameters

- *-h* - Displays help screen
- *-a* - Run all telecom clock tests

Example

```
post telecom_clock -a
```

A.2.72 post uart

Perform and show results of the UART tests.

Syntax

```
post uart -h | -a | -d -r -q -j -p -i -l -n <UART ID>
```

Parameters

- *-d* - Dump all UART readable registers
- *-r* - Run UART register test
- *-q* - Run UART Non-FIFO polling test
- *-j* - Run UART Non-FIFO interrupt test
- *-p* - Run UART FIFO polling test
- *-i* - Run UART FIFO interrupt test
- *-a* - Run all UART register tests
- *-h* - Display help screen
- *-l* - Run loopback test
- *-n <UART ID>* - ID of the UART to be tested:
  - 1 - Internal UART
  - 2 - Second IXP2850 network processor UART
  - 3 - Communication to BMC

Example

```
post uart -a -n 1
```

A.2.73 post ueng

Perform microengine register test. This test checks control, context, and timer registers accessible by the microengines.

Syntax

```
post ueng -h | -a | -c -r -t -s -m -d <microengine number>
```

Parameters

- *-c* - Run microengine context test
- *-m* - Run microengine msf register test
- *-r* - Run microengine register test
- *-t* - Run timers and counters test
- *-s* - Run microengine control store test
- *-d <microengine number>* - The microengine ID to be tested. The range is:
  - 0 to 3
  - 16 to 19
  - 99 (for all microengine tests)
- *-a* - Run all pre-defined microengine tests
- *-h* - Display help screen

Example

```
post ueng -d 1 -a
```

A.2.74 post xscale

Perform XScale core verification tests. The scratchpad memory and IRQ interrupt are tested.

Syntax

```
post xscale -h | -a | -s -p -i
```

Parameters

- *-h* - Display help screen
- *-a* - Run all pre-defined XScale core tests
- *-s* - Run XScale core, scratch ring test
- *-p* - Run XScale core, scratch pattern test
- *-i* - Run XScale core interrupt test

Example

```
post xscale -s
```

A.2.75 register 8read

Read 8-bit wide data from memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 8read -b <address>
```

Parameters

- *- b <address>* - Memory, SFR or slow port address.

Example

```
register 8read -b 0xa1000000
```

A.2.76 register 8write

Write 8-bit wide data to memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 8write -b <address> -v <value>
```

Parameters

- *- b <address>* - Memory, SFR or slow port address
- *- v <value>* - Data to be written

Example

```
register 8write -b 0xa1000000 -v 0x12
```

A.2.77 register 16read

Read 16-bit wide data from memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 16read -b <address>
```

Parameters

- *- b <address>* - Memory, SFR or slow port address

Example

```
register 16read -b 0xa1000000
```

A.2.78 register 16write

Write 16-bit wide data to memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 16write -b <address> -v <value>
```

Parameters

- *- b <address>* - Memory, SFR or slow port address
- *- v <value>* - Data to be written

Example:

```
register 16write -b 0xa1000000 -v 0x1234
```

A.2.79 register 32read

Read 32-bit wide data from memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 32read -b <address>
```

Parameters

- - *b* <*address*> - Memory, SFR or slow port address

Example

```
register 32read -b 0xa1000000
```

A.2.80 register 32write

Write 32-bit wide data to memory, SFR or the slow port space of the IXP2850 network processor.

Syntax

```
register 32write -b <address> -v <value>
```

Parameters

- - *b* <*address*> - Memory, SFR or slow port address
- - *v* <*value*> - Data to be written

Example

```
register 32write -b 0xa1000000 -v 0x12345678
```

A.2.81 reset

Reboot the Network Processor (NP) or Board Management Controller (BMC).

Syntax

```
reset [ np | bmc ]
```

Parameters

- *np* - Reset the NP
- *bmc* - Reset the BMC
- no parameter - Reset the NP (same as *np*)

Example:

```
reset

reset bmc

reset up
```

A.2.82 sensor read

Read sensors.

Syntax

```
sensor read –n <sensor_num>
```

Parameters

- *-n <sensor_num>* - Sensor number

Example

```
sensor read –n 0x2
```

A.2.83 soft_reset boot

Reboot Redboot with soft reset.

Syntax

```
soft_reset boot
```

Example

```
soft_reset boot
```

A.2.84 soft_reset check

Check if SRAM is filled with a predefined pattern.

Syntax

```
soft_reset check [-f]
```

Parameters

- *-f* - Display full information about each mismatched memory cell.

Example

```
soft_reset check
```

A.2.85 soft_reset fill

Fill the entire SRAM with a predefined pattern.

Syntax

```
soft_reset fill
```

Example

```
soft_reset fill
```

A.2.86 ssu

Safe System Upgrade (SSU) offers a safe method of upgrading the selected partitions in the Network Processor (NP) flash.

If the upgrade fails, SSU undoes all modifications and restores the previous board image (image rollback). SSU verifies the CRC of the image SSU header before storing the image in flash. This SSU header is linked to the end of actual image and stored on flash memory. The SSU commands upgrade the following partitions:

- On the NP flash:
  - RedBoot
  - Linux Kernel
  - Linux root filesystem
  - Target application image
- On the Board Management Controller (BMC) flash:
  - BMC image

A downloaded image is stored in the backup partition and will not be used unless it is verified. For verification, a backup image must be executed in “safe” mode. This image will not be used for the subsequent execution unless it is validated. Once it is validated, it becomes the primary image to be used for the subsequent booting, while the former primary image becomes the backup one.

A.2.87 ssu bootmode

Used to set the booting mode.

Note: This command is **not** intended for use with IXB2850 boards.

A.2.88 ssu diag

A service command that is password protected and used to diagnose problems with SSU functionality.

Note: This command is **not** intended for customer use.

A.2.89 ssu exec

Run the previously loaded Linux kernel image, and provide information to the kernel (via the kernel command line) about the active Linux root file system and the target application image.

Syntax

```
ssu exec [-h | -d | -n | -f | -c <cmdline> | -w <waittime> | -p <pbase>] [<entry_point>]
```

Parameters

- *-h* - Display help screen
- *-d* - Print debug information before proceeding
- *-n* - Specifies the root file system on NFS
- *-f* - Specifies the root file system on flash
- *-c <cmdline>* - Specifies the command line for Linux (use quotation marks for the text)
- *-w <waittime>* - Specifies a delay for kernel execution
- *-p <pbase>* - Specifies the address of embedded kernel command line parameters

- <entry_point> - Specifies the address to jump to (the default is the address set by the `ssu load` command)

Example

```
ssu exec -f
```

A.2.90 ssu init

Initialize all the SSU structures required for SSU operation. This command should be run after a non-SSU flash partition update.

Syntax

```
ssu init [-l <kernel size>] [-r <rootfs size>] [-t <target_app size>]
```

Parameters

- *-l <kernel size>* - Kernel partition initialization only
- *-r <rootfs size>* - Root file system partition initialization only
- *-t <target_app size>* - Target application partition initialization only

A.2.91 ssu list

Display SSU-specific information about the SSU images stored on flash.

Syntax

```
ssu list [-h] | [-a]
```

Parameters

- *-h* - Display help screen
- *-a* - Display active partitions before inactive ones; otherwise, the partitions are displayed by type (active, then inactive).

Example

```
ssu list -a
```

A.2.92 ssu load

Load the active Linux kernel image from flash.

Syntax

```
ssu load [-h | -d | -b <address>]
```

Parameters

- *-h* - Display help screen
- *-d* - Print debug information before proceeding
- *-b <address>* - Load the image at the specified address instead of the default address (0x0D008000)

Example

```
ssu load
```

A.2.93 ssu oneshot

Set/reset execution of the specified image type in “safe” mode for next boot verification.

Syntax

```
ssu oneshot [-h] | -i <image type>
```

Parameters

- *-h* - Display help screen
- *-i <image type>* - The image type; one of the following:
  - *bootmonitor* - RedBoot image
  - *kernel* - Linux kernel image
  - *rootfs* - Linux root file system image
  - *targetapp* - Target application image
  - *bmc* - BMC image

Example

```
ssu oneshot -i kernel
```

A.2.94 ssu upgrade

Download and write the specified SSU image using the Xmodem protocol or TFTP.

Syntax

```
ssu upgrade [-h] | { [-i <image_type>] [-b <address> -l <length>] }
```

Parameters

- *-h* - Display help screen
- *-i <image type>* - The image type; one of the following:
  - *bootmonitor* - RedBoot image
  - *kernel* - Linux kernel image
  - *rootfs* - Linux root file system image
  - *targetapp* - Target application image
  - *bmc* - BMC image
- *-b <address>* - The location in RAM where the uploaded image should be stored
- *-l <length>* - The length of the uploaded image

Example

```
ssu upgrade -b 0x03000000 -l 0x300000
```

A.2.95 ssu validate

Validate the image running in “safe” mode and set as the primary image.

Syntax

```
ssu validate [-h] | -i <image type>
```

Parameters

- *-h* - Display help screen
- *-i <image type>* - The image type; one of the following:
  - *bootmonitor* - RedBoot image
  - *kernel* - Linux kernel image
  - *rootfs* - Linux root file system image
  - *targetapp* - Target application image
  - *bmc* - BMC image

Example

```
ssu validate -i bmc
```

A.2.96 syslog

Manage the system log.

Syntax

```
syslog {clear | show [-n <number>]}
```

Parameters

- clear - Clears the entire system event log.
- show -n <*number*> - Shows the number of system log entries specified by the *-n* parameter.
  If the *-n* parameter is omitted, shows the entire system log.

Example

```
syslog show -n 3
```

A.2.97 trb

Switch between RedBoot images (“RedBoot” and “_RedBoot”). Use this command only if it is certain that the second image is operational. To test an image that has just been burned, instead use the *ssu oneshot* command (see the ssu command for details).

Syntax

```
trb
```

Example

```
trb
```

A.2.98 update bmc

Store updated Board Management Controller (BMC) firmware in the BMC flash memory.

Note: Before running this command, you must use the load command to load the new BMC firmware into RAM.

Syntax

```
update bmc -b<base_address> -l<length>
```

Parameters

- *-b <base_address>* - The RAM address where the new BMC firmware has been loaded.
- *-l <length>* - The length of the new BMC firmware in RAM.

Example

```
update bmc -b 0x1000000 -l <length>
```

A.2.99 update fru

Stores new field replaceable unit (FRU) information in a given component's EEPROM. The FRU update fails if there is a hardware protection (write protect) on the appropriate board ID EEPROM.

Note: Before running this command, you must use the load command to load the new FRU information into RAM.

Warning: Running this command erases all information in the given FRU, including MAC addresses. Before performing this command, you should use the cfg read command and note the MAC address settings and after resetting the board, write and save them using the cfg mac and cfg save commands.

Syntax

```
update fru -c <component> -b <base_address> -l <length> -o
```

Parameters

- *-c <component>* - The component to change/assign FRU information to:
  - *bb* - baseboard
  - *db1* - media mezzanine card 1 (not used on IXB2850 boards)
  - *db2* - media mezzanine card 2
  - *mic1* - media interface card 1 (not used on IXB2850 boards)
  - *mic2* - media interface card 2
  - *fic* - fabric interface card
  - *npmod* - network processor module
  - *ap* - adjunct processor card
- *-b <base_address>* - The RAM address where the new FRU information is to be loaded
- *-l <length>* - The length of the new FRU information in RAM.
- *-o* - Forces MAC addresses, serial numbers, custom OEM records (id=1 and id=2, version 0.0) and all other unknown type records to be overwritten with information from the FRU file.

Example

```
update fru -c bb -b 0x1000000 -l <length>
```

A.2.100 update sdr

Stores new Sensor Device Record (SDR) information for a given configuration in EEPROM.

Note: Before running this command, the load command must be executed to load new SDR information to RAM.

Syntax

```
update sdr -c <component> -b <start_address> -l <length>
```

Parameters

- *-c <component>* - The component to change/assign FRU information to:
  - *bb* - baseboard
  - *db1* - media mezzanine card 1 (not used on IXB2850 boards)
  - *db2* - media mezzanine card 2
  - *mic1* - media interface card 1 (not used on IXB2850 boards)
  - *mic2* - media interface card 2
  - *fic* - fabric interface card
  - *npmod* - network processor module
  - *ap* - adjunct processor card
- *-b <start_address>* - The RAM address where the new SDR information has been loaded.
- *-l <length>* - The length of the new SDR information in RAM.

Example

```
update sdr -c bb -b 0x3000000 -l 0x76f
```

A.2.101 version

Display RedBoot version information.

Syntax

```
version
```

Example

```
version
```

A.2.102 watchdog

Turn watchdog on or off.

Syntax

```
watchdog on -t <sec>

watchdog off
```

Parameters

- *-t <sec>* - Turn the watchdog timeout on for *<sec>* seconds.

Example

```
watchdog on -t 60
```

Appendix B Power On Self Test

The Power-On Self Test (POST) performs initialization and basic tests of the Intel XScale® core and main baseboard components that the XScale core depends on. These tests are designed so that external cable connections to the board are not required.

B.1 POST Categories

The categories of POST tests are:

- UART POST
- Memory POST
- PCI POST
- BMC POST
- Slow Port Ethernet POST
- PCI Ethernet POST
- Microengines POST
- Slow Port POST
- Interrupt POST
- XScale Core POST
- MSF POST
- General Purpose I/O (GPIO) POST
- I2C POST
- LED POST
- Media Access POST
- Media Traffic POST
- Telecom Clock POST

B.1.1 UART POST

The UART POST covers the initialization and test of the serial ports. Each serial line register should be written and then read for verification. An internal loopback test should also be performed. After tests are completed, the console serial port can be used to report the progress of POST and the boot process in general. The tests cases presented in the following table are executed for all UART devices:

- UART#0 - internal UART
- UART#1 - UART to IPMI controller

- UART#2 - external UART

Test ID	Description
00h	Register access
01h	Non-FIFO polling
02h	Non-FIFO interrupt test
03h	FIFO polling
04h	FIFO interrupt test
05h	Loopback

B.1.2 Memory POST

The following memory types are installed and should be tested:

- **DRAM** - The remaining part of DRAM (not tested by ROM code) is tested in the second phase of POST. This is a 'quick' version of the test, where only one location of each 4K block of memory is tested (pattern test). Note that a long DRAM test (pattern test of each memory location) is available from Boot Monitor's console.
- **SRAM** - This is a "quick" version of the test, where only one location of each 4K block of memory is tested (pattern test). A long SRAM test (pattern test of each memory location) is available from Boot Monitor's console. Note that this applies to all four SRAM controllers supporting SRAM installed on baseboard.

The tests cases presented in the following table are executed.

Test ID	Description
00h	Walking ones
01h	Walking zeros
02h	Known pattern tests: • DRAM and SRAM: - 0x5a5a5a5a pattern - 0xa5a5a5a5 pattern
03h	Address bus test
04h	Incremental test
05h	Unique test

B.1.3 PCI POST

The NPU is the PCI host, so it performs the main PCI configuration. These tests check for the presence of the PCI devices, such as PCI bridges and verify PCI interrupt generation. The tests cases presented in the following table are executed.

Test ID	Description
00h	Device scan on PCI bus – all PCI devices board presence checking
01h	PCI interrupt test

B.1.4 BMC POST

The BMC POST performs IPMI communication tests. The test checks for presence of the Board Management Controller (BMC) and establish a connection with the BMC through the UART port. The contents of the ID EEPROMs are read from the BMC to detect the hardware configuration (there is one ID EEPROM on the baseboard and one on each

extension card). Information about the hardware configuration (contents of ID EEPROMs) is stored in the BMC for use by operational software. Additionally, information about the detected hardware can be displayed on the console. The tests cases presented in the following table are executed.

Test ID	Description
00h	BMC board presence checking
01h	Baseboard ID EEPROM reading from BMC

B.1.5 Slow Port Ethernet POST

The slow port Ethernet POST test checks for the presence of an Ethernet device, registers access and sends/receives test packets. The tests cases presented in the following table are executed.

Test ID	Description
00h	Slow port Ethernet device board presence checking
01h	Ethernet interrupt test
02h	Register access test
03h	Send/receive packet test (internal loopback)

B.1.6 PCI Ethernet POST

The PCI Ethernet POST test checks for the presence of the PCI Ethernet device (82546 Dual Port Gigabit Ethernet Controller), registers access and sends/receives test packets. The tests cases presented in the following table are executed for both PCI Ethernet device ports.

Test ID	Description
00h	PCI Ethernet device board presence checking
01h	Ethernet interrupt test
02h	Register access test
03h	Send/receive packet test (internal loopback)

B.1.7 Microengines POST

The Microengines POST tests check the control, context, and timer registers accessible by microengines. A microengine is loaded with very simple code that modifies a selected location in DRAM (for example, multiplication by 2). The selected memory location is initiated before the microengine executes the loaded code and verified after execution. These tests are repeated for every microengine. The tests cases presented in the following table are executed.

Test ID	Description
00h	Control Store test
01h	Register test
02h	MSF CSR test
03h	Timers and Counters test
04h	Context test

B.1.8 Slow Port POST

The slow port POST tests perform slow port registers verification. The tests cases presented in the following table are executed.

Test ID	Description
00h	Registers default values test
01h	Registers write/read test

B.1.9 Interrupt POST

The Interrupt controller verification test is available. Simulated interrupts are checked. The tests cases presented in the following table are executed.

Test ID	Description
00h	Simulated interrupt test

B.1.10 XScale Core POST

XScale core verification tests are available, including tests of scratchpad memory and interrupts. The tests cases presented in the following table are executed.

Test ID	Description
00h	Interrupts
01h	Scratchpad pattern test
02h	Scratchpad ring test
03h	Generic timer

B.1.11 MSF POST

The MSF tests perform Media Switch Fabric registers verification. The tests cases presented in the following table are executed.

Test ID	Description
00h	Registers default values test
01h	Registers write/read test

B.1.12 General Purpose I/O (GPIO) POST

The GPIO POST tests verify the GPIO lines. The tests cases presented in the following table are executed.

Test ID	Description
00h	Detection test
01h	Register test
02h	Edge detection test
03h	Level detection test

B.1.13 I²C POST

The I²C bus is a software-emulated bus that uses two GPIO lines. The bus is used to access EEPROM that stores SPI-4 static alignment and SRAM deskew settings. Because EEPROM content is used to store information about memory size, only addresses from 0x80 are available for testing. The tests cases presented in the following table are executed.

Test ID	Description
00h	Write-check test
01h	Page write
02h	Current position read
03h	Random read
04h	Sequential read

B.1.14 LED POST

The LED POST tests are automatic tests checking proper work of CPLD devices responsible for controlling LEDs. The operator is responsible for visual verification of LED operation. The tests cases presented in the following table are executed.

Test ID	Description
00h	Walking 0/1 test for baseboard/NPM LEDs
01h	Default state of MMC#1 LEDs; not applicable to IXB2850 boards
02h	Walking 0/1 test for MMC#1 LEDs; not applicable to IXB2850 boards
03h	Default state of MMC#2 LEDs
04h	Walking 0/1 test for MMC#2 LEDs

B.1.15 Media Access POST

The test performs MAC/Framer, PHY/Serdes, and register verification. The test cases presented in the following table are performed for:

- MMC#1devices; not applicable to IXB2850 boards
- MMC#2 devices
- Baseboard devices
- FIC devices

Test ID	Description
00h	Registers default values test for MAC/Framer device
01h	Registers write/read test for MAC/Framer device
02h	Registers default values test for PHY/Serdes device
03h	Registers write/read test for PHY/Serdes device
04h	Registers default values test for MAC/Framer ports
05h	Registers write/read test for MAC/Framer ports
06h	Registers default values test for PHY/Serdes ports

Test ID	Description
07h	Registers write/read test for PHY/Serdes ports
08h	Registers default values test for MIC devices
09h	Registers write/read test for MIC devices

B.1.16 Media Traffic POST

The test performs MAC/Framer and PHY/Serdes initialization and starts the microcode responsible for receiving and transmitting frames. During system loopback test, the microengine sends a frame with a checksum, then receives a frame on the same interface. The following devices are tested during the Media POST:

- SPI-3/4 Bridge and Fork FPGA
- Baseboard MAC and PHY
- FIC MAC
- Quad Gigabit Ethernet Mezzanine Card MAC and PHY

The test cases presented in the following table are performed for:

- MMC#1devices (not applicable to IXB2850 boards)
- MMC#2 devices (Quad Gigabit Ethernet Mezzanine Card)
- Baseboard devices
- FIC devices

Test ID	Description
00h	MAC/Framer device configuration
01h	MAC/Framer port configuration
02h	Traffic test on system loopback on MAC/Framer ports
03h	PHY/Serdes port configuration
04h	Traffic test on system loopback on PHY/Serdes ports

B.1.17 Telecom Clock POST

The Telecom Clock POST performs Telecom Clocks (Zarlink*) functionality test. This test generates known frequencies and verifies if the Zarlink synchronies with the reference clock. The tests cases presented in the following table are executed.

Test ID	Description
00h	Telecom Clock device board presence checking
01h	2.048 MHz frequency test
02h	19.44 MHz frequency test
03h	8 kHz frequency test
04h	1.544 MHz frequency test

Appendix C Diagnostics

The Diagnostics Test Module consists of various test routines for the different hardware components on board such as, the Intel XScale® core, memory, UART, debug Ethernet, PCI, microengines, etc. This appendix describes:

- Diagnostics Tests
- Diagnostics Commands

C.1 Diagnostics Tests

The diagnostics tests include:

- LED Tests
- BMC Tests
- XScale Core Tests
- Memory Tests
- Serial/UART Tests
- Serial/UART Tests
- PCI Tests
- GPIO Tests
- I2C Tests
- Slow Port Tests
- Slow Port Ethernet Tests
- PCI Ethernet Tests
- Interrupt Tests
- MSF Tests
- Microengine Tests
- Media Access Tests
- Media Traffic Tests
- Telecom Clock Tests

C.1.1 LED Tests

The LED tests are automatic tests checking the proper operation of the CPLD devices responsible for controlling the LEDs. The operator is responsible for visual verification of LED operation. The tests cases presented in the following table are executed.

Test ID	Description
00h	Walking 0/1 test for baseboard/NPM LEDs
01h	Default state of MMC#1 LEDs; not applicable to IXB2850 boards
02h	Walking 0/1 test for MMC#1 LEDs; not applicable to IXB2850 boards
03h	Default state of MMC#2 LEDs
04h	Walking 0/1 test for MMC#2 LEDs

The user specifies how many times the test is performed (the default is 1).

C.1.2 BMC Tests

The BMC POST tests perform IPMI BMC communication tests. The tests check for presence of the Board Management Controller and establish a connection with the BMC through a UART port. The content of the ID EEPROMs are read from the BMC. The tests cases presented in the following table are executed.

Test ID	Description
00h	BMC board presence checking
01h	Baseboard ID EEPROM reading from BMC

The user specifies how many times a test is performed (the default is 1).

C.1.3 XScale Core Tests

XScale core verification tests are available, including tests of scratchpad memory and interrupts. The tests cases presented in the following table are executed.

Test ID	Description
00h	Interrupts
01h	Scratchpad pattern test
02h	Scratchpad ring test
03h	Generic timer
04h	Watchdog timer

The user specifies how many times a test is performed (the default is 1).

C.1.4 Memory Tests

The following memory types are installed and should be tested:

- DRAM
- SRAM

The tests cases presented in the following table are executed.

Test ID	Description
00h	Walking ones
01h	Walking zeros
02h	Known pattern tests • DRAM and SRAM: - 0x5a5a5a5a pattern - 0xa5a5a5a5 pattern
03h	Address bus test
04h	Incremental test
05h	Unique test
06h	User defined pattern
07h	ECC test – DRAM only

The user can specify:

- memory test start address
- size of tested memory block
- data pattern in user defined pattern test
- how many times test is performed (default is 1)

C.1.5 Serial/UART Tests

UART tests verify each line registers as well as interrupt generation and internal loopback. The tests cases presented in the following table are executed for all UART devices:

- UART#0 – NPU embedded UART
- UART#1 – first external UART
- UART#2 – second external UART

Test ID	Description
00h	Register access
01h	Non-FIFO polling
02h	Non-FIFO interrupt test
03h	FIFO polling
04h	FIFO interrupt test
05h	Loopback

The user can specify which UART should be tested.

C.1.6 PCI Tests

The NPU is the PCI host, so it performs the main PCI configuration. The PCI tests check for the presence of PCI devices, such as PCI bridges and verify PCI interrupt generation. The tests cases presented in the following table are executed.

Test ID	Description
00h	Device scan on PCI bus – all PCI devices board presence checking
01h	PCI interrupt test

The user can specify how many times test is performed (the default is 1).

C.1.7 GPIO Tests

The GPIO tests verify the NPU GPIO lines. The tests cases presented in the following table are executed.

Test ID	Description
00h	Detection test
01h	Register test
02h	Edge detection test
03h	Level detection test

The user can specify how many times test is performed (the default is 1).

C.1.8 I^2C Tests

The I^2C bus is a software-emulated bus that uses two GPIO lines. The bus is used to access EEPROM that stores SPI-4 static alignment and SRAM deskew settings. Because EEPROM content is used to store information about memory size, only addresses from 0x80 are available for testing. The tests cases presented in the following table are executed.

Test ID	Description
00h	Write-check test
01h	Page write
02h	Current position read
03h	Random read
04h	Sequential read

The user can specify:

- Device address on the bus (in the range 0 to 7)
- Address in I^2C device (in the range 0 to FFF)
- Page size (in the range 1 to 20)
- Data written into I^2C device (in the range 0 to FF)
- How many times test is performed (the default is 1)

C.1.9 Slow Port Tests

The slow port tests perform slow port registers verification. The tests cases presented in the following table are executed.

Test ID	Description
00h	Registers default values test
01h	Registers write/read test

The user can specify how many times test is performed (the default is 1).

C.1.10 Slow Port Ethernet Tests

The slow port Ethernet tests check for the presence of an Ethernet device, register access and send/receive test packets. The tests cases presented in the following table are executed.

Test ID	Description
00h	Slow port Ethernet device board presence checking
01h	Ethernet interrupt test
02h	Register access test
03h	Send/receive packet test (internal loopback)
04h	Send/receive packet (external loopback – special cable required)

The user can specify how many times test is performed (the default is 1).

C.1.11 PCI Ethernet Tests

The test checks for the presence of a PCI Ethernet chip (Intel® 82546 Dual Port Gigabit Ethernet Controller), registers access and sends/receives test packets. The tests cases presented in the following table are executed for both PCI Ethernet device ports.

Test ID	Description
00h	PCI Ethernet device board presence checking
01h	Ethernet interrupt test
02h	Register access test
03h	Send/receive packet test (internal loopback)
04h	Send/receive packet (external loopback – special cable required)

The user can specify how many times the test is performed (the default is 1).

C.1.12 Interrupt Tests

Interrupt controller verification tests are available. The test checks the IRQ line from the interrupt controller to the XScale core. The tests cases presented in the following table are executed.

Test ID	Description
00h	Simulated interrupt test

The user can specify how many times the test is performed (the default is 1).

C.1.13 MSF Tests

These tests perform Media Switch Fabric (MSF) registers verification. The tests cases presented in the following table are executed.

Test ID	Description
00h	Registers default values test
01h	Registers write/read test

The user can specify how many times the test is performed (the default is 1).

C.1.14 Microengine Tests

The tests check the control, context, and timer registers accessible by microengines. A microengine is loaded with very simple code that modifies a selected location in DRAM (for example, multiplication by 2). The selected memory location is initiated before the microengine executes the loaded code and verified after execution. The tests cases presented in the following table are executed.

Test ID	Description
00h	Control Store test
01h	Register test
02h	MSF CSR test
03h	Timers and Counters test
04h	Context test

The user can specify:

- The microengine number
- How many times test is performed (the default is 1)

Microengines will be completely tested during media tests (see Section C.1.15), where packet transmission and reception is checked.

C.1.15 Media Access Tests

The test performs MAC/Framer, PHY/Serdes and MIC verification. The test cases presented in the following table are performed for:

- MMC#1devices; not applicable to IXB2850 boards
- MMC#2 devices
- Baseboard devices
- FIC devices

Test ID	Description
00h	Registers default values test for MAC/Framer device
01h	Registers write/read test for MAC/Framer device
02h	Registers default values test for PHY/Serdes device
03h	Registers write/read test for PHY/Serdes device
04h	Registers default values test for MAC/Framer ports
05h	Registers write/read test for MAC/Framer ports

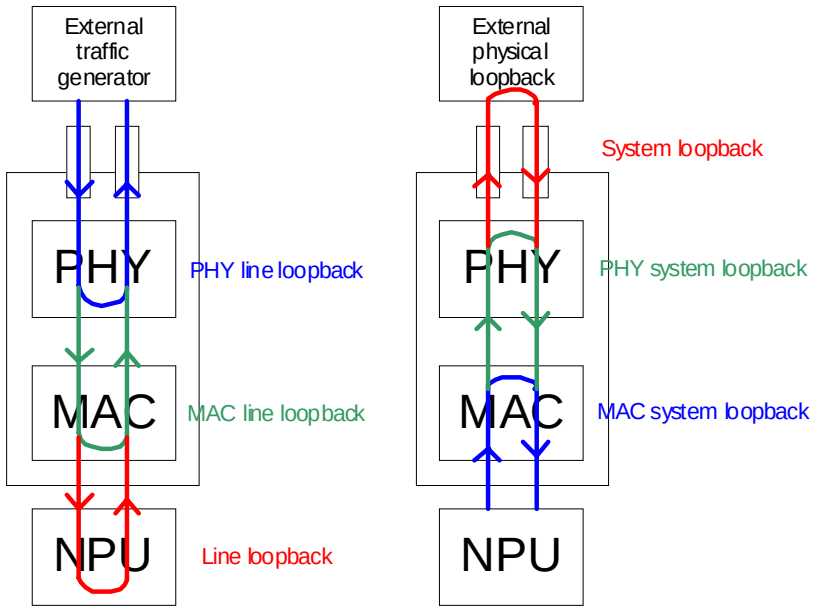
Test ID	Description
06h	Registers default values test for PHY/Serdes ports
07h	Registers write/read test for PHY/Serdes ports
08h	Registers default values test for MIC devices
09h	Registers write/read test for MIC devices

The user can specify how many times test is performed (default is 1).

C.1.16 Media Traffic Tests

The test performs initialization of MAC/PHY for Gigabit Ethernet and starts the microcode responsible for receiving and transmitting frames. During these tests, the media interfaces are verified by two types of loopback; system and line loopback (see Figure 51). During a system loopback test, the microengine sends frames with a checksum; after reception, the checksum is verified. This loopback may require an external loopback connection. In the case of a line loopback, an external packet generator is needed.

Figure 51. Media loopback test types



Line loopback types System loopback types

The following devices are tested during the Media POST:

- SPI-3/4 Bridge and Fork FPGA
- Baseboard MAC and PHY
- FIC MAC
- Quad Gigabit Ethernet Mezzanine Card MAC and PHY

The test cases presented in the following table are performed on all ports of tested device.

Test ID	Description
00h	MAC/Framer device configuration
01h	MAC/Framer port configuration
02h	Traffic test on system loopback on MAC/Framer ports
03h	PHY/Serdes port configuration
04h	Traffic test on system loopback on PHY/Serdes ports
03h	External port configuration
04h	Traffic test on external loopback

The user can specify:

- Traffic type – single packet, packet burst or continuous
- Tested port
  - Quad Gigabit Ethernet Mezzanine ports
  - FIC Gigabit Ethernet port
  - Baseboard Gigabit Ethernet port
- Interface speed
  - 10Mbps, 100Mbps and 1Gbps for Ethernet (on backplane is only, 1Gbps)
- Interface mode
  - Full duplex, half duplex for Ethernet

C.1.17 Telecom Clock Tests

The Telecom Clock tests perform the Telecom Clocks (Zarlink*) functionality verification. This test generates known frequencies and verifies if the Zarlink synchronizes with the reference clock. The tests cases presented in the following table are executed.

Test ID	Description
00h	Telecom Clock device board presence checking
01h	2.048 MHz frequency test
02h	19.44 MHz frequency test
03h	8 kHz frequency test
04h	1.544 MHz frequency test

The user can specify how many times the test is performed (the default is 1).

C.2 Diagnostics Commands

The supported diagnostics commands are described in the following subsections. For information on using the diagnostics commands, see Section 6.9.1, “Running the Diagnostics” on page 79.

C.2.1 banner

Print a banner containing basic hardware and software information.

Syntax

```
banner
```

Example

```
banner
```

C.2.2 blockcopy

Copy a block of memory between address *low* and *high* to address *dest*.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
blockcopy low high dest
```

Parameters

- *low* address in memory where the block begins
- *high* address in memory where the block ends
- *dest* address in memory where the block should be copied to

Example

```
blockcopy 800000 800100 400000
```

C.2.3 c

Interactively change the contents of a specified memory address. This command allows you to enter a 32-bit value for the <*addr*> argument (1000000 in the example shown below). After you press **Enter**, you are prompted to then enter a value for the address. To exit out of this interface and return to the DIAG> prompt, press <ESC>.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
c addr
```

Parameters

- *addr* memory address that will change

Example

```
c 1000000
```

C.2.4 cfg

Read (r) or write (w) a byte (b), word (w) or dword (d) from/to a PCI device.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
cfg[rw] [bwd]
cfgrb bus slot addr
cfgwb bus slot addr data
```

```
cfgrw bus slot addr
cfgww bus slot addr data
cfgrd bus slot addr
cfgwd bus slot addr data
```

Parameters

- *bus*- PCI bus number of the device
- *slot*- slot number of the device
- *addr*- memory address that will be written to/read from
- *data*- data to write to the memory address

Example

```
cfgrb 1 0 0
```

C.2.5 doff

Disable data cache.

Syntax

```
doff
```

Example

```
doff
```

C.2.6 don

Create page table, enable data cache and enable wb.

Syntax

```
don
```

Example

```
don
```

C.2.7 exit

Exit the diagnostics prompt.

Syntax

```
exit
```

Example

```
exit
```

C.2.8 fill

Fill a block of memory with the specified value.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
fill low high val
```

Parameters

- *low* address in memory where the block begins
- *high* address in memory where the block ends
- *val* value that will be copied to every memory cell in the specified block

Example

```
fill 800000 800100 41
```

C.2.9 help

Print the help screen for a given diagnostic command.

Syntax

```
h command
```

Parameters

- *command* specific command to display help screen for

Note: A specific command is optional. You can enter *h* to get help on all diagnostic commands.

Example

```
h fill
```

(Gets help on the fill command.)

C.2.10 i2cread

Read from an I^2C device and display the results.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
i2cread slave_addr offset_addr count
```

Parameters

- *slave_addr* address of the I^2C device on the bus
- *offset_addr* memory address inside the I^2C device
- *count* number of bytes read (decimal format)

Example

```
i2cread 1 0 10
```

C.2.11 i2cwrite

Write string data into the memory of an I^2C device.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
i2cwrite slave_addr offset_addr data
```

Parameters

- *slave_addr* address of the I^2C device on the bus
- *offset_addr* memory address inside the I^2C device
- *data* hexadecimal string to write to I^2C device memory

Example

```
i2write 4 30 123456789abcdef123456
```

C.2.12 io

Read (r) or write (w) a byte (b), word (w) or dword (d) from/to a PCI IO.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
io[rw] [bwd]
iorb addr rept silent
iowb addr data rept
iorw addr rept silent
ioww addr data rept
iord addr rept silent
iowd addr data rept
```

Parameters

- *addr*- memory address that will be written to/read from
- *rept* repeat this on a loop. Can be set as follows:
  - — 0- perform a read or write one time only
  - — 1- perform a read or write on a loop until a key is pressed
- *data*- data to write to the memory address
- *silent*- do not trace any information on the console
  - — 0- display information on the console
  - — 1- run in silent mode

Example

```
iorb 1 0 0
```

C.2.13 ioff

Disable instruction cache.

Syntax

```
ioff
```

Example

```
ioff
```

C.2.14 ion

Enable instruction cache.

Syntax

```
ion
```

Example

```
ion
```

C.2.15 mem

Read (r) or write (w) a byte (b), word (w) or dword (d) from/to memory.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
mem[rw] [bwd]
memrb addr rept silent
memwb addr data rept silent
memrw addr rept silent
memww addr data rept silent
memrd addr rept silent
memwd addr data rept silent
```

Parameters

- *addr*- memory address that will be written to/read from
- *rept* repeat this on a loop. Can be set as follows:
  - — 0- perform a read or write one time only
  - — 1- perform a read or write on a loop until a key is pressed
- *data*- data to write to the memory address
- *silent*- do not trace any information on the console
  - — 0- display information on the console
  - — 1- run in silent mode

Example

```
mem 1 0 0
```

C.2.16 p

Display the contents of memory that starts at address *addr* and is *size* bytes long.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
p addr size
```

Parameters

- *addr* starting address of a memory block
- *size* length of a memory block

Example

```
p 800000 80
```

C.2.17 sethoe

Set the halt on error value.

Syntax

```
sethoe val
```

Parameters

- *val* halt on error value

Example

```
sethoe 3
```

C.2.18 setverbose

Set the verbose output level for all diagnostic commands.

Syntax

```
setverbose level
```

Parameters

- *level* verbose level. Can be set to any of the following:
  - 0- DISABLE
  - 1- TYPE_ERROR
  - 2- TYPE_WARNING
  - 3- TYPE_DBGMSG1
  - 4- TYPE_DBGMSG2
  - 5- TYPE_DBGMSG3

Example

```
setverbose 3
```

C.2.19 test

Display a list of test commands within the diagnostics image.

Syntax

```
test command
```

Parameters

- *command* specific test command to display description of.

Note: A specific test command is optional. You can enter *test* to get help on all test commands.

Example

```
test
```

C.2.20 tf

Enable test flags. Causes given device to be included or excluded from tests in the *test all* command.

Syntax

```
tf test_type_ID status
```

Parameters

- *test_type_id* test identifier. Can be any of the following:
  - 0- LED
  - 1- GPIO
  - 2- MSF
  - 3- SLOWPORT
  - 4- UENG
  - 5- XSCALE
  - 6- INTERRUPT
  - 7- BMC
  - 8- I^2C
  - 9- UART
  - A- MEDIA
  - B- MEM
  - C- TIMERS
  - D- ETHPCI
  - E- PCI
  - F-ETHSLOW
- *status* test flag. Can be any of the following:
  - 0- do not perform test
  - 1- perform test

Example

```
tf 4 1
```

C.2.21 t bmc

Perform Board Management Controller (BMC) tests. The tests read base card and mezzanine card EEPROMs.

Syntax

```
t bmc option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *f*- BMC presence test
  - *i*- BMC read ID EEPROM
  - *s*- BMC show ID EEPROM
  - *a*- all BMC tests
  - *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t bmc f
```

C.2.22 t ethslow

Perform slow port Ethernet tests.

Syntax

```
t ethslow option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *f*- Ethernet device presence test
  - *r*- register access test
  - *e*- external system loopback test
  - *m*- MAC (internal) system loopback test
  - *i*- Ethernet interrupt test
  - *a*- all Ethernet tests
  - *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t ethslow r
```

C.2.23 t generate

This command starts the traffic generator for all ports configured by the t media command.

Syntax

```
t generate mode test sizeMin sizeMax pattern countHi countLo
```

Description

Starts the traffic generator based on configuration set by t media command. The test starts receive and transmit microblocks. The microblocks sends and receives frames with wire speed. The microblocks send frames with checksum that makes possible to receive frame on any interface or even by another board running the same test.

Parameters

mode - a mode of generation

- E - Enable frame test
- D - Disable frame test
- S - Enable frame test
- H - Help
- HM - Help (mode only)
- HT - Help (test only)
- HP - Help (frame parameters only)

test - a test type to perform

- S - selected frame size test, default pattern
- I - incremental frame size test, default pattern
- D - decremental frame size test, default pattern
- R - random frame size test, default pattern
- *S - * frame size test, selected pattern
- *I - * frame size test, incremental byte pattern
- *IB - * frame size test, incremental byte pattern
- *IW - * frame size test, incremental word pattern
- *ID - * frame size test, incremental dword pattern
- *D - * frame size test, decremental byte pattern
- *DB - * frame size test, decremental byte pattern
- *DW - * frame size test, decremental word pattern
- *DB - * frame size test, decremental dword pattern
- *R - * frame size test, random pattern
- C - short microengines counter show
- CF - Full microengines counter show
- CT - transmit microengines counter show
- CR - receive microengines counter show
- CO - other microengines counter show

sizeMin - a minimal length of packet to be send (including header etc.)

sizeMax - a maximal length of packet to be send (including header etc.)

pattern - a pattern for packet payload, or starting seed

countHi - higher 32 bits of 64-bit transmit frame counter

countLo - lower 32 bits of 64-bit transmit frame counter

C.2.24 t gpio

Perform GPIO tests that check presence, registers, edge detection and level detection.

Syntax

```
t gpio option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *d*- GPIO detection test
  - *r*- GPIO register test
  - *e*- GPIO edge detection test
  - *l*- GPIO level detection test
  - *a*- all GPIO tests
  - *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t gpio l
```

C.2.25 t i2c

Perform I^2C bus tests. The I^2C bus is a software-emulated bus that uses two GPIO lines. The bus is used to read DDRAM EEPROM content.

Syntax

```
t i2c w dev addr data
t i2c t dev addr data
t i2c p dev addr size
t i2c c dev addr data
t i2c r dev addr count
t i2c s dev addr count
t i2c a dev
t i2c u mem_size
t i2c v mem_size
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *w*- I^2C write-only command
  - *t*- write check test
  - *p*- I^2C page write test
  - *c*- I^2C current read test
  - *r*- I^2C random read-only test
  - *s*- I^2C sequential read test
  - *a*- all I^2C read-only tests
  - *h*- help
  - *u*- update DDRAM EEPROM content with specified mem_size
  - *v*- check DDRAM EEPROM content with specified *mem_size*
- *dev*- I^2C device address on the bus (0-7)
- *addr*- address in I^2C device (0-FFF)
- *size*- page size used in test (1-20)

- *data*- data to write into I^2C device (0-FF)
- *count*- count size used in test (1-1000)
- *mem_size*- memory size that should be programmed into EEPROM. Can be set as follows:
  - *256*- 256MB
  - *512*- 512MB

Example

```
t i2c p 1 0 10
```

C.2.26 t int

Perform interrupt controller tests. The test checks the IRQ line from the interrupt controller to the XScale core. CPLD interrupt simulation and 125us clock interrupt is tested.

Syntax

```
t int option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *s*- interrupt simulation test
  - *i*- 125us interrupt test
  - *a*- all interrupt tests
  - *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t int i
```

C.2.27 t led

Perform LED tests. The tests cover all LED tests but it is assumed that the LED is working correctly.

Syntax

```
t led option state loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *o*- orange LED test
  - *g*- green LED test
  - *r*- red LED test
  - *rn*- NP module LED red test
  - *gn*- NP module LED green test
  - *yn*- NP module LED yellow test
  - *an*- NP module all LED test

- *gb*- BB module LED green test
- *rb*- BB module LED red test
- *yb*-BB module LED yellow test
- *ab*- BB module all LED tests
- *y*- yellow LED test
- *a*- all LED tests
- *h* - help

- *state*- indicates the expected state of the LED. Can be set as follows:
  - *e*- enable LED
  - *d*- disable LED
  - *s*- inverse LED state
  - *t*- test LED (default)
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t led y
```

C.2.28 t media

This command prepares the media for traffic generation.

Syntax

```
t media mode test ports speed type
```

Parameters

mode - a test mode

- B - Base Board mode
- FICM - Fabric Interface Card type M mode
- FICC48 - Fabric Interface Card type C48 mode
- G - Quad Gigabit Ethernet Mezzanine Card mode
- R - reset all previously set media configurations and disable tests for all modes
- C - configuration trace - show ports configuration and link state
- H - help
- HM - help - mode only
- HT - help - test only
- HP - help - ports only
- HS - help - speed and type only

test - a test type to perform

- L - NPU line loopback - receive frame and transmit
- LM - media line loopback - MAC receive frame and transmit
- LP - PHY line loopback - PHY receive frame and transmit
- S - NPU system loopback - NPU generate, receive and check frames, use external loopback

- SM - media system loopback - NPU generate, receive and check frames, loopback on media. In the case of FIC C48 it configures so called shallow loopback.
- SP - PHY system loopback - NPU generate, receive and check frames, loopback on PHY. In the case of FIC C48 it configures so called deep loopback.
- R - all MAC and PHY registers test
- RM - all MAC registers test
- RMD - MAC default registers check
- RMU - MAC update registers test
- RP - all PHY registers test
- RPD - MAC default registers test
- RPU - PHY update registers test
- RF - all Fiber MIC registers test
- RFD - default Fiber MIC registers test
- RFU - update Fiber MIC registers test
- RC - all Copper MIC registers test (not applicable to IXB2850)
- RCD - default Copper MIC registers test (not applicable to IXB2850)
- RCU - update Copper MIC registers test (not applicable to IXB2850)
- D - disable test for selected mode
- C - short counter show
- CMF - Full media hardware counter show (also for C48)
- CMT -transmit media hardware counter show (also for C48)
- CMR - receive media hardware counter show (also for C48)
- CMO - other media hardware counter show (also for C48)
- CPF - full PHY hardware counter show (also for C48)
- CPT - transmit PHY hardware counter show (also for C48)
- CPR - receive PHY hardware counter show (also for C48)
- CPO - other PHY hardware counter show (also for C48)
- H - help

ports -- list (demarcated by commas) of ports to be tested

- DB1/1 - interface number '1' on daughter card number '1'
- DB1/2 - interface number '2' on daughter card number '1'

For FIC C48 DB1/1 and DB1/2 should be specified to point the interface when temporary solution is used (FIC inserted in DB slot - then the DB1 flow is tested)

- DB1/3 - interface number '3' on daughter card number '1'
- DB1/4 - interface number '4' on daughter card number '1'
- DB1 - all interfaces on daughter card number '1'
- DB2/1 - interface number '1' on daughter card number '2'
- DB2/2 - interface number '2' on daughter card number '2'
- DB2/3 - interface number '3' on daughter card number '2'
- DB2/4 - interface number '4' on daughter card number '2'
- DB2 - all interfaces on daughter card number '2'

- DB - all interfaces on both daughter cards
- DF1/1 - fiber interface number '1' on daughter card number '1'
- DF1/2 - fiber interface number '2' on daughter card number '1'
- DF1/3 - fiber interface number '3' on daughter card number '1'
- DF1/4 - fiber interface number '4' on daughter card number '1'
- DF1 - all fiber interfaces on daughter card number '1'
- DF2/1 - fiber interface number '1' on daughter card number '2'
- DF2/2 - fiber interface number '2' on daughter card number '2'
- DF2/3 - fiber interface number '3' on daughter card number '2'
- DF2/4 - fiber interface number '4' on daughter card number '2'
- DF2 - all fiber interfaces on daughter card number '2'
- DF - all fiber interfaces on both daughter cards
- DC1/1 - copper interface number '1' on daughter card number '1'
- DC1/2 - copper interface number '2' on daughter card number '1'
- DC1/3 - copper interface number '3' on daughter card number '1'
- DC1/4 - copper interface number '4' on daughter card number '1'
- DC1 - all copper interfaces on daughter card number '1'
- DC2/1 - copper interface number '1' on daughter card number '2'
- DC2/2 - copper interface number '2' on daughter card number '2'
- DC2/3 - copper interface number '3' on daughter card number '2'
- DC2/4 - copper interface number '4' on daughter card number '2'
- DC2 - all copper interfaces on daughter card number '2'
- DC - all copper interfaces on both daughter cards
- FB1 - fiber interface number '1'
- FB2 - fiber interface number '2'
- FB - both fiber interfaces
- CP1 - copper interface number '1'
- CP2- copper interface number '2'
- CP - both copper interfaces
- BASE1 - base interface number '1'
- BASE2 - base interface number '2'
- BASE - both base interfaces
- FIC1/1 - fabric interface number '1'
- FIC1/2 - fabric interface number '2'
- FIC1/3 - fabric interface number '3'
- FIC1/16 - fabric interface number '16'

For FIC C48 FIC1/1 and FIC1/2 should be specified to point the interface when target solution is used (FIC inserted in FIC slot - then the FIC flow is tested)

- FIC1 - all fabric interfaces

speed - interface speed

- 0 - autonegotiation of speed for 'B' and 'F' mode
- 10 - 10MHz for 'B' and 'F' mode
- 100 - 100MHz for 'B' and 'F' mode
- 1000 - 1GHz for 'B' and 'F' mode
- 3 - OC-3 for 'P' and 'A' mode
- 12 - OC-12 for 'P' and 'A' mode

type - Intel® IXF1104 and Marvell* Alaska PHY duplex settings

- F - full duplex for 'B' and 'F' mode
- H - half duplex for 'B' and 'F' mode
- A - autonegotiation of duplex for 'B' and 'F' mode
- O - OC-mode for 'P' and 'A' mode
- S - STM-mode for 'P' and 'A' mode

Example

```
t media b rad
```

Sample Output

```
RESULT=PASS
```

C.2.29 t mem

Perform memory tests. The tests cover all types of memory and various test methods such as walking ones or walking zeros.

Syntax

```
t mem option test_option start_addr size data loops verbose
```

Parameters

- *option*- indicates the memory type to test. Can be set as follows:
  - *s*- SRAM memory test channel 0 and 1 (default)
  - *s0*- SRAM memory test channel 0 only
  - *s1*- SRAM memory test channel 1 only
  - s2- SRAM memory test channel 2 only
  - s3- SRAM memory test channel 3 only
  - *d*- DRAM memory test
  - *a*- all memory types
  - *h*- help
- *test_option*- sets the test method to use. Can be set as follows:
  - *w1*- walking one test
  - *w0*- walking zero test
  - *w*- both walking tests
  - *c5*- checker 0x5A5A5A5A test
  - *ca*- checker 0xA5A5A5A5 test

  - *c*- both checker test
  - *u*- unique test
  - *b*- address bus test
  - *p*- pattern test (a value for *data* is required)
  - *i*- incremental test
  - *e*- ECC test (only applicable to DRAM)
  - *a*- all tests
- *start_addr*- memory address where the test should begin
- *size*- number of bytes to test starting from *start_addr*
- *data*- data used for patter test (mandatory for the pattern (*p*) test)
- *loops*- specifies the number of times a test is performed (default value is 1)
- *verbose*- sets the verbose level for the test. This setting is only valid during the initial execution of a looped command.

Example

```
t mem a a
```

C.2.30 t msf

Perform MSF tests.

Syntax

```
t msf option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *d*- MSF register default test
  - *r*- MSF register write-read test
  - *a*- all MSF tests
  - *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t msf d
```

C.2.31 t pci

Perform PCI tests. The tests check if all requested PCI devices are present and PCI interrupts can be generated.

Syntax

```
t pci option bridges AP_presence
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - *p*- PCI presence test

  - — *i*- PCI interrupt test
  - — *a*- all PCI tests
  - — *h*- help
- *bridges*- specifies the number of PCI bridges. Can be set as follows:
  - — *b1*- search for one PCI bridge (default)
  - — *b2*- search for two PCI bridges
- AP_presence- indicates if an Adjunct Processor (AP) mezzanine card is present at the PMC site. Can be set as follows:
  - — *0* - AP is not present (default)
  - — *1* - AP is present

Example

```
t pci i
```

C.2.32 t slowport

Perform slow port tests.

Syntax

```
t slowport option loop
```

Parameters

- *option*- indicates the test to perform. Can be set as follows:
  - — *d*- Slow port register default test
  - — *r*- Slow port register write-read test
  - — *a*- all slow port tests
  - — *h*- help
- *loop*- specifies the number of times the test is performed (default is 1)

Example

```
t slowport d
```

C.2.33 t uart

Perform UART tests. The test cover all types of UART in the system. User intervention is required to finish the test.

Syntax

```
t uart command uart_id
```

Parameters

- *command*- indicates a test method to perform. Can be set as follows:
  - — *d*- dump all UART readable registers
  - — *r*- UART register test
  - — *l*- UART loopback test (if *uart_id* is not equal to 1)
  - — *q*- UART non-FIFO polling test (if *uart_id* is not equal to 2)

  - *j*- UART non-FIFO interrupt test (if *uart_id* is not equal to 2)
  - *p*- UART FIFO polling test (if *uart_id* is not equal to 2)
  - *i*- UART FIFO interrupt test (if *uart_id* is not equal to 2)
  - *a*- all UART tests
  - *h*- help
- *uart_id*- UART ID for test. Can be set as follows:
  - *1*- internal UART
  - *2*- UART to BMC
  - *3*- network processor external UART

Example

```
t uart r 1
```

C.2.34 t ueng

Perform a microengine test.

Syntax

```
t ueng option microengine loop
```

Parameters

- *option*- indicates a test method to perform. Can be set as follows:
  - *c*- crypto test (for Intel® IXP2850 network processor only)
  - *m*- MSF register test
  - *r*- register test
  - *s*- control store test
  - *t*- timers and counters test
  - q- QDR access stress test (channel number 0...3)
  - *a*- all microengine tests
  - *h*- help
- *microengine*- specifies the microengine ID to test. 0-3 and 16-19. Entering 99 will test all microengines.
- *loop*- specifies how many times the test is performed (default is 1)

Example

```
t ueng c 1
```

C.2.35 t xscale

Perform scratchpad and core component tests on the XScale core. The tests cover IRQs and internal scratch memory tests.

Syntax

```
t xscale option loop
```

Parameters

- *option*- indicates a test option to perform. Can be set as follows:
  - *i*- interrupt test
  - *r*- scratchpad ring test
  - *s*- scratchpad pattern test
  - *g*- generic timer test
  - *w*- watchdog timer test
  - *a*- all tests
  - *h*- help
- *loop*- specifies how many times the test is performed (default is 1)

Example

```
t xscale s 1
```

C.2.36 wrv

Write, read and verify to a specified memory address.

Note: All memory addresses are entered in hexadecimal format without the *0x* prefix.

Syntax

```
wrv location size
```

Parameters

- *location* address of a memory block
- *size* length of a memory block (decimal format)

Example

```
wrv 800000 80
```

Appendix D OEM IPMI Commands

This appendix contains two sets of OEM IPMI Commands:

- Intel OEM IPMI Commands
- Board-Specific OEM IPMI Commands

D.1 Intel OEM IPMI Commands

The Intel OEM IPMI commands supported by IXB2850 boards include:

- Get Slot Number
- Bused Resource Control Request
- Get Bused Resource Control Status
- Set Bused Resource Control
- Get Shelf Address
- Set Processor Watchdog

D.1.1 Get Slot Number

Get Slot Number **NetFn**: Intel Platform Specific (32h/33h) **CMD**: 10h		
Description: Command sent to the BMC to get the board slot number and the BMC firmware version.		
Data	**Byte**	**Data Field**
Request		
Response	1	**Completion Code** (see below)
	2	**Slot Number** (slots are numbered starting with 0)
	3	MSB Firmware version
	4	LSB Firmware version

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully

D.1.2 Bused Resource Control Request

Bused Resource Control Request **NetFn:** Intel Platform Specific (32h/33h) **CMD:** 12h		
Description: Command sent from the NPU to the BMC to perform an operation associated with a particular bused resource.		
Data	**Byte**	**Data Field**
Request	1	**Resource ID**: • 00h – Metallic Test Bus pair #1 • 01h – Metallic Test Bus pair #2 • 02h – Synch clock group 1 (CLK1A and CLK1B pairs) • 03h – Synch clock group 2 (CLK2A and CLK2B pairs) • 04h – Synch clock group 3 (CLK3A and CLK3B pairs)
	2	**Command**: • 00h – Request • 01h – Relinquish • 02h – Notify • 03h – Lock • 04h – Unlock • 05h – Reserve
Response	1	**Completion Code** (see below)
	2	**Resource ID** (copied from Request Data)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_OUTOFRANGE	C9h	Invalid command (out of supported range)
COMPCODE_INVALIDFIELD	CCh	Invalid Resource ID

D.1.3 Get Bused Resource Control Status

Get Bused Resource Control Status **NetFn:** Intel Platform Specific (32h/33h) **CMD:** 13h		
Description: Command sent from the NPU to the BMC to get the status of a particular bused resource.		
Data	**Byte**	**Data Field**
Request	1	**Resource ID** (see Section D.1.2)
Response	1	**Completion Code** (see below)
	2	**Resource ID** (copied from Request Data)
	3	**State**: • 00h – In control • 01h – No control • 02h – Waiting for control • 03h – Locked for control • 04h – Ready to get control
	4	**Last command**. Last command sent to the ShMC in Bused Resource Control Request command
	5	**Last status**. Status from response to the last Bused Resource Control Request command

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_INVALIDFIELD	CCh	Invalid Resource ID

D.1.4 Set Bused Resource Control

Set Bused Resource Control **NetFn**: Intel Platform Specific (32h/33h) **CMD**: 14h		
Description: Command sent from the BMC to the NPU to disable/enable the controlling of a particular bused resource. This command is called when the BMC receives a standard PICMG **Bused Resource Control** command from the ShMC.		
Data	**Byte**	**Data field**
Request	1	**Command**: • 00h – Enable • 01h – Disable
	2	**Resource ID** (see Section D.1.2, "Bused Resource Control Request" on page 257)
Response	1	**Completion Code** (see below)
	2	**Resource ID** (copied from Request Data)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_OUTOFRANGE	C9h	Invalid Command (out of supported range)
COMPCODE_INVALIDFIELD	CCh	Invalid Resource ID

D.1.5 Get Shelf Address

Get Shelf Address **NetFn:** Intel Platform Specific (32h/33h) **CMD:** 15h		
Description: Command sent to the BMC to get address info from the ShMC. This command should return information obtained from the ShMC in response to the standard PICMG **Get Shelf Address Info** command.		
Data	**Byte**	**Data Field**
Request	1	Counter
Response	1	**Completion Code** (see below)
	2	Counter
	3	Data length
	4 ... N	Data

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully

D.1.6 Set Processor Watchdog

Set Processor Watchdog **NetFn**: Intel Platform Specific (32h/33h) **CMD**: 16h		
Description: Command sent to the BMC to start the watchdog for a particular processor. The timeout is given in seconds. A timeout set to zero deactivates the watchdog for given processor.		
Data	**Byte**	**Data Field**
Request	1	**Processor ID**: • 00h – NPU • 01h – AP
	2	LSB Timeout
	3	MSB Timeout
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_INVALIDFIELD	CCh	Invalid Processor ID

D.2 Board-Specific OEM IPMI Commands

Table 68 lists OEM IPMI commands that are specific to the IXB2850 board. These commands use NetFn = 08h/09h (firmware commands) for this purpose.

Table 68. Board-specific OEM IPMI commands

Command Name	CMD	Comments
Firmware General Commands		
Enter Xfer Mode	00h	See Section D.2.1 for details.
Download	01h	See Section D.2.2 for details.
Exit Xfer Mode	04h	See Section D.2.3 for details.
Set Segment	05h	See Section D.2.4 for details.
Set Status Word	08h	See Section D.2.5 for details.
Get Status Word	09h	See Section D.2.6 for details.
Read Image Header Info	0Ah	See Section D.2.7 for details.
Read Image Header Data	0Bh	See Section D.2.8 for details.
Burn Image	0Ch	See Section D.2.9 for details.
Last Operation Status	0Eh	See Section D.2.10 for details.
Perform Operation	0Fh	See Section D.2.11 for details.
Set Software Reset Timeout	16h	See Section D.2.12 for details.
Firmware SSU commands		
SSU Get Image Information	25h	See Section 11.4.1, “Safe System Upgrade IPMI Commands” on page 152 for details.
SSU Image Upgrade	26h	
SSU Change Image State	27h	
SSU Validate All Images	28h	
SSU Upgrade Operation Status	29h	

D.2.1 Enter Xfer Mode

Enter Xfer Mode **NetFn**: Firmware (08h/09h) **CMD**: 00h		
Description: Command used to enter into BMC firmware upgrade mode		
Data	**Byte**	**Data Field**
Request	1	'I'
	2	'N'
	3	'T'
	4	'E'
	5	'L'
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_INVALIDFIELD	CCh	Invalid Request data

D.2.2 Download

Download **NetFn**: Firmware (08h/09h) **CMD**: 01h		
Description: Command used to download data to the BMC RAM for a specified segment		
Data	**Byte**	**Data Field**
Request	1	LSB Offset
	2	MSB Offset
	3	Data Length
	4 to N	Data
Response	1	**Completion Code** (see below)
	2	**LSB Offset** (copied from Request Data)
	3	**MSB Offset** (copied from Request Data)
	4	**Data Length** (copied from Request Data)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_ COMMANDILLEGAL	CDh	Illegal command (the BMC is not in upgrade mode)

D.2.3 Exit Xfer Mode

Exit Xfer Mode **NetFn**: Firmware (08h/09h) **CMD**: 04h		
Description: Command used to exit from BMC upgrade mode		
Data	**Byte**	**Data field**
Request		
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_ COMMANDILLEGAL	CDh	Illegal command (the BMC is not in upgrade mode)

D.2.4 Set Segment

Set Segment **NetFn**: Firmware (08h/09h) **CMD**: 05h		
Description: Command used to specify the segment of firmware code (virtual segments – in flash/EEPROMs (see Section 8.6, "Non-Volatile Storage Programming" on page 104 for details)		
Data	**Byte**	**Data Field**
Request	1	LSB Segment
	2	MSB Segment
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_INVALIDFIELD	CCh	Invalid Segment
COMPCODE_ COMMANDILLEGAL	CDh	Illegal command (the BMC is not in upgrade mode)

D.2.5 Set Status Word

Set Status Word **NetFn**: Firmware (08h/09h) **CMD**: 08h		
Description: Command used to set the Status Word for BMC image		
Data	**Byte**	**Data Field**
Request	1	Status Word
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed

D.2.6 Get Status Word

Get Status Word **NetFn**: Firmware (08h/09h) **CMD**: 09h		
Description: Command used to get the Status Word for BMC image		
Data	**Byte**	**Data Field**
Request		
Response	1	**Completion Code** (see below)
	2	Status Word

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed

D.2.7 Read Image Header Info

Read Image Header Info **NetFn**: Firmware (08h/09h) **CMD**: 0Ah		
Description: Command used to read information about the BMC Image Header.		
Data	**Byte**	**Data Field**
Request	1	**Image ID**: • 00h – BMC firmware image 1 • 01h – BMC firmware image 2 • 10h – Baseboard CPLD chain • 11h – NPM CPLD chain • 12h – DB1 CPLD chain (not applicable to IXB2850 boards) • 13h – DB2 CPLD chain • 14h – MIC1 CPLD chain (not applicable to IXB2850 boards) • 15h – MIC2 CPLD chain • 17h – FIC CPLD chain
Response	1	**Completion Code** (see below)
	2	LSB Size
	3	MSB Size
	4	7:1 – Reserved 0:0 – Access: • 0b – Image is accessed by bytes • 1b – Image is accessed by words

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed
COMPCODE_INVALIDFIELD	CCh	Invalid Image ID

D.2.8 Read Image Header Data

Read Image Header Data **NetFn**: Firmware (08h/09h) **CMD**: 0Bh		
Description: Command used to read the BMC Image Header. This command should be used after a **Read Image Header Info** command to get the Image Header.		
Data	**Byte**	**Data Field**
Request	1	**Image ID** – see Section D.2.7, "Read Image Header Info" on page 264
	2	LSB Offset
	3	MSB Offset
	4	Counter
Response	1	**Completion Code** (see below)
	2	Counter
	3 to N	Data

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed (header absent)
COMPCODE_REQDATATOOLONG	C8h	Requested data exceed header size
COMPCODE_INVALIDFIELD	CCh	Invalid Image ID

D.2.9 Burn Image

Burn Image **NetFn**: Firmware (08h/09h) **CMD**: 0Ch		
Description: Command used to burn-in image downloaded to BMC RAM to non-volatile storage (for given segment set by **Set Segment** command before download)		
Data	**Byte**	**Data Field**
Request	1	LSB Segment
	2	MSB Segment
	3	LSB Size
	4	MSB Size
Response	1	**Completion Code** (see below). The requester should not wait for the completion code since a response message can arrive after a long time (when burn process finishes). The Last Operation Status command should be used to track the burning process (see Section D.2.10, "Last Operation Status" on page 266 for details).

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed
COMPCODE_OUTOFRANGE	C9h	Requested image size exceed size of given segment
COMPCODE_INVALIDFIELD	CCh	Invalid segment

D.2.10 Last Operation Status

Last Operation Status **NetFn**: Firmware (08h/09h) **CMD**: 0Eh		
Description: Command used to read status of the last operation performed by BMC		
Data	**Byte**	**Data Field**
Request	1	**Operation Type**: • 00h – Download/Burn operations • 02h – CPLD chain operations
Response	1	**Completion Code** (see below)
	2	**Operation Status**: • 00h – Idle • 01h – Passed • 02h – Failed • 03h – In progress
	3	**Operation Status Details** – Not used

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_INVALIDFIELD	CCh	Invalid Operation Type

D.2.11 Perform Operation

Perform Operation **NetFn**: Firmware (08h/09h) **CMD**: 0Fh		
Description: General purpose command		
Data	**Byte**	**Data Field**
Request	1	**Operation Type**: • 03h – read CPLD chains parameters
	2	Request Data Size
	3 ... N	Request Data
Response	1	**Completion Code** (see below)
	2	Operation Type
	3	Response Data Size
	4 ... N	Response Data

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_FAILED	20h	Operation failed
COMPCODE_INVALIDFIELD	CCh	Invalid Operation Type
COMPCODE_ COMMANDILLEGAL	CDh	Illegal command (BMC not in upgrade mode)

D.2.12 Set Software Reset Timeout

Set Software Reset Timeout **NetFn**: Firmware (08h/09h) **CMD**: 16h		
Description: Command used for soft reset purposes. Allows a user to set the duration of the soft reset and the start of the soft reset procedure.		
Data	**Byte**	**Data Field**
Request	1	**Request Data**: Time to soft reset in seconds (0 to 31) If set to 0, a hard reset occurs immediately.
	2	**Request Data**: Total time to soft reset in seconds (0 to 255) If set to 0, a hard reset occurs immediately. If 255 seconds limit is reached, the soft reset is activated immediately, even if the command has been sent to override the soft reset timeout.
Response	1	**Completion Code** (see below)

Completion Codes

Code	Value	Description
COMPCODE_NORMAL	00h	Operation completed successfully
COMPCODE_REQDATABADLEN	C7h	Wrong request length
COMPCODE_OFFSETOUTOFRANGE	CCh	The first value is greater than 31 seconds

Appendix E Driver API Reference

This appendix describes the following driver APIs:

- Baseboard Driver External API
- Gigabit Ethernet Media Driver API

E.1 Baseboard Driver External API

The baseboard driver API is described under the following topics:

- Baseboard Driver API Functions
- Baseboard Driver API Structures

E.1.1 Baseboard Driver API Functions

Table 69 lists the baseboard driver functions.

Table 69. Baseboard driver API function summary

Function	Description
Open and Close Functions	
Bb_Open( )	Sets driver mode to “UP”
Bb_Close( )	Sets driver mode to “DOWN”
Device Information Functions	
Bb_GetDevType( )	Detects device type
Bb_GetSlotNumber( )	Retrieves slot number for the baseboard
Bb_GetFruParam( )	Retrieves type, size, and value of FRU parameters
Bb_GetMacAddr( )	Retrieves MAC address based on interface type and port number
SPI Functions	
Bb_SetMode( )	Sets operation mode of a device
LED Control Functions	
Bb_LedControl( )	Set LED status on the baseboard devices
Bb_MediaLedControl( )	Set LED status on the mezzanine devices
MIC Control Functions	
Bb_MicRtmCommand( )	Perform an action on a MIC device
Reset Functions	
Bb_ResetControl( )	Global reset of baseboard devices
PLL Control Functions	
Bb_DrvGet_PLL_Number( )	Check installed PLLs

Table 69. Baseboard driver API function summary (Continued)

Function	Description
Bb_DrvSet_Primary_Source( )	Select clock source for PLL
Bb_DrvSet_Secondary_Source( )	Select clock source for PLL
Bb_DrvSet_Sync_Source( )	Select synchronization source for PLL
Bb_DrvSet_Sync_SelMode( )	Select synchronization mode for PLL
Bb_DrvGet_Sync_Status( )	Read status information
Bb_DrvSet_Mezz_Ref( )	Set reference signal for mezzanine
Bb_DrvSet_Backpl_Ref( )	Set backplane reference signal
Bb_DrvGet_CDC_Config( )	Read CDC configuration
Bb_DrvGet_CDC_Init( )	Initialize the CDC module
Crosspoint Switch Configuration Functions	
Bb_ConnectPort( )	Connect GE MAC device interface to the backplane interface port
Bb_DisconnectPort( )	Disconnect GE MAC device interface from the backplane interface port

E.1.1.1 Bb_Open( )

Name

Bb_Open( )

Synopsis

```
int
Bb_Open
(
    void
)
```

Description

This function sets operational mode UP for the driver.

Parameters

Type	Name	Description
Void	n/a	n/a

Returns

Return code	Description
handle	Driver unique handle

E.1.1.2 Bb_Close( )

Name

Bb_Close( )

Synopsis

```
int
```

```
Bb_Close
(
    int handle
)
```

Description

This routine sets operational mode DOWN for the driver.

Parameters

Type	Name	Description
IN	handle	Driver unique handle

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed

E.1.1.3 Bb_GetDevType( )

Name

Bb_GetDevType( )

Synopsis

```
int
Bb_GetDevType
(
    int handle,
    int dev_id
)
```

Description

This function detects the device type on the baseboard.

A single baseboard supports two MIC cards inserted either on mezzanine cards or on the baseboard directly.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	dev_id	ID for the device to be examined one of the following: • BASE_BOARD_ID • MEDIA_CARD1_ID (not applicable to IXB2850 boards) • MEDIA_CARD2_ID • MIC1_ID (not applicable to IXB2850 boards) • MIC2_ID

Returns

Return Code	Description
BB_FAILURE	Operation failed
dev_type	Return code, if not returning BB_FAILURE, is the numeric value of the device type found. It can be one of the following: • BASEBOARD_ID • GB_ETH_ID • MIC_F4xGB_MEZZ_ID

E.1.1.4 Bb_GetSlotNumber( )

Name

Bb_GetSlotNumber( )

Synopsis

```
int
Bb_GetSlotNumber
(
    int handle
)
```

Description

This function retrieves the physical slot number where the baseboard is inserted.

Parameters

Type	Name	Description
IN	Handle	Driver unique handle

Returns

Return Code	Description
BB_FAILURE	Operation initialization failed
Number	Number of the slot occupied by the baseboard

E.1.1.5 Bb_GetFruParam( )

Name

Bb_GetFruParam( )

Synopsis

```
int
Bb_GetFruParam
(
    int handle,
    int fru_id,
    int param_id,
    int *pType
    size_t *pSize,
    void *pValue
)
```

Description

This routine retrieves the type, size and value of the specified FRU parameter.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	fru_id	ID of FRU to be processed, can be one of the following: • BASE_BOARD_ID (FRU on baseboard) • NP_MODULE_ID (FRU on NPM) • MEDIA_CARD1_ID (FRU on media mezzanine card in slot 1, not applicable to IXB2850 boards) • MEDIA_CARD2_ID (FRU on media mezzanine card in slot 2) • MIC1_ID (FRU on device inserted in MIC slot 1, not applicable to IXB2850 boards) • MIC2_ID (FRU on device inserted in MIC slot 2) • FIC_IC (FRU on the Fabric Interface Card) • PMC_ID (FRU on PMC card)
IN	param_id	ID of FRU parameter to be retrieved, can be one of the following: • BB_MANUFACTURER (board manufacturer name) • BB_PRODUCT_NAME (board product name) • BB_SERIAL_NUMBER (board serial number) • BB_PART_NUMBER (8-digit board part number) • BB_MAC_INFO (MAC address, interface type, and port number) • BB_PICMG_INFO (manufacturer ID, PICMG ID)
OUT	pType	Type of returned FRU parameter: • BB_STRING (string data) • BB_NUMBER (number) • BB_MAC_ARRAY (table of structures containing MAC information) • BB_PICMG_ARRAY (a structure containing PICMG information)
OUT	pSize	Size of output parameter. Possible values are: • MAX_FRU_STR_LEN (64 bytes) • MAX_MAC_NUMBER (15 x MAC_INFO size of 14 bytes)
OUT	pValue	Pointer to output parameter

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed
BB_INVALID_SIZE	*pValue buffer is too small (size value is too small)
BB_TRY_AGAIN	Variable value cannot be retrieved at this time (driver has not been initialized yet)
BB_UNKNOWN_VAR	Incorrect variable identifier

E.1.1.6 Bb_GetMacAddr( )

Name

Bb_GetMacAddr( )

Synopsis

```
int
Bb_GetMacAddr
(
```

```
    int handle,
    int fru_id,
    unsigned int iface,
    unsigned int port,
    unsigned int *pSize,
    void *pValue
)
```

Description

This function gets the MAC address based on the specified interface type and port number.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	fru_id	ID of FRU to be processed, can be one of the following: • BASE_BOARD_ID (FRU on baseboard) • NP_MODULE_ID (FRU on NPM) • MEDIA_CARD1_ID (FRU on media mezzanine card in slot 1, not applicable to IXB2850 boards) • MEDIA_CARD2_ID (FRU on media mezzanine card in slot 2) • MIC1_ID (FRU on device inserted in MIC slot 1, not applicable to IXB2850 boards) • MIC2_ID (FRU on device inserted in MIC slot 2) • FIC_IC (FRU on the Fabric Interface Card) • PMC_ID (FRU on PMC card)
IN	iface	Interface type of the interface associated with MAC to be retrieved. The interface types are: BB_DEBUG_IFACE (10BASE-T debug port) BB_IXF_1104_IFACE (Intel® IXF1104 Gigabit Ethernet MAC) BB_82546_IFACE (82546 Dual Port Gigabit Ethernet Controller)
IN	port	Port number of the interface associated with MAC to be retrieved
OUT	pSize	Size of output parameter (6 bytes for MAC address)
OUT	pValue	Pointer to output parameter

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed

E.1.1.7 Bb_SetMode( )

Name

Bb_SetMode( )

Synopsis

```
int
Bb_SetMode
(
    int handle,
    int dev_id,
    int mode,
```

```
    int parity
)
```

Description

This function sets the operation mode for a device.

Note: This function changes device parameters that were set during initialization.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	dev_id	Device ID, can be one of the following: • BB_DEV0 • BB_DEV1 • BB_DEV2
IN	mode	Operating mode to be set on the device, can be one of the following: • BB_IXDP2801_SPI3_MODE
IN	parity	Parity to set on the device The following parity settings can be used: • BB_PARITY_NONE • BB_PARITY_EVEN • BB_PARITY_ODD

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed

E.1.1.8 Bb_LedControl( )

Name

Bb_LedControl ( )

Synopsis

```
int
Bb_LedControl
(
    int handle,
    bb_dev_id_t dev_id,
    bb_led_cmd_t cmd,
    int led_id
)
```

Description

This function sets the LED status on all devices present on the baseboard based on their unique ID.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	dev_id	Device ID, can be one of the following: BASE_BOARD_ID MEDIA_CARD1_ID, not applicable to IXB2850 boards MEDIA_CARD2_ID MIC1_ID, not applicable to IXB2850 boards MIC2_ID FIC_ID NP_MODULE_ID PMC_ID
IN	cmd	Command to execute: BB_LED_ON BB_LED_OFF
IN	led_id	LED identifier For all devices: 0 - BB_LED_ALL (all LEDs on given device are chosen) If dev_id is BASE_BOARD_ID or NP_MODULE_ID the following assignment is valid: • 1 – Yellow LED • 2 – Red LED • 3 – Green LED • 4 – Orange LED For MIC and mezzanine cards the following mapping is valid: • 1 – Green LED on port 1 • 2 – Green LED on port 2 • 3 – Green LED on port 3 • 4 – Green LED on port 4 • 5 – Amber LED on port 1 • 6 – Amber LED on port 2 • 7 – Amber LED on port 3 • 8 – Amber LED on port 4 ***Note:*** When Red and Green are ON at the same time, the combined color is Orange.

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed
BB_HW_IN_RESET	The LED cannot be controlled as the device is in reset state.
BB_HW_NOT_FOUND	The device is not found in a system.
BB_BAD_PARAMS	The specified parameters are wrong.

E.1.1.9 Bb_MediaLedControl( )

Name

Bb_MediaLedControl( )

Synopsis

```
int
Bb_MediaLedControl
(
```

```
    int handle,
    bb_dev_id_t dev_id,
    bb_media_led_cmd_t cmd,
    int port
)
```

Description

This function controls the LED status on Mezzanine and MIC cards.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	dev_id	Device ID, can be one of the following: • MEDIA_CARD1_ID, not applicable to IXB2850 boards • MEDIA_CARD2_ID • MIC1_ID, not applicable to IXB2850 boards • MIC2_ID • FIC_ID
IN	cmd	Command to be executed: • BB_MEDIA_OFF (link down) • BB_MEDIA_ACTIVE (link up) • BB_MEDIA_ERROR (error occurred) • BB_MEDIA_ENABLE (laser on) See the table below for mapping to devices.
IN	port	Port number: starting from 1. One might also specify BB_PORT_ALL (chooses all port on given device)

	MIC	Mezzanine Card
BB_MEDIA_OFF	None	None
BB_MEDIA_ACTIVE	Green	Green
BB_MEDIA_ERROR	None	None
BB_MEDIA_ENABLE	Amber	Amber

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed
BB_HW_IN_RESET	The LED cannot be controlled as the device is in reset state.
BB_HW_NOT_FOUND	The device is not found in a system.
BB_BAD_PARAMS	The specified parameters are wrong.

E.1.1.10 Bb_MicRtmCommand( )

Name

Bb_MicRtmCommand( )

Synopsis

```
int
Bb_MicRtmCommand
(
    int handle,
    bb_dev_id_t dev_id,
    int port,
    bb_mic_cmd_t cmd,
    int *param
)
```

Description

This function controls MIC (performs an action on the MIC device).

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	dev_id	Device ID, can be one of the following: • MEDIA_CARD1_ID, not used • MEDIA_CARD2_ID
IN	port	Port number on selected MIC (1 – 8) or BB_PORT_ALL (choose all ports on selected device)
IN	cmd	Command to be executed: General commands (`port` variable is ignored): • BB_INIT • BB_RESET_MIC • BB_ENABLE_NORM_MIC • BB_ENABLE_DBG_MIC • BB _READ_VERSION Commands for Fiber MIC devices: • BB _READ_SDET (can not be used with BB_PORT_ALL) • BB _ENABLE_TDIS • BB _DISABLE_TDIS • BB_ENABLE_GREEN_LED • BB_DISABLE_GREEN_LED • BB_ENABLE_RED_LED • BB_DISABLE_RED_LED • BB_ENABLE_FIBER_PORT • BB_ENABLE_COPPER_PORT
IN/OUT	param	Additional parameter to retrieve command results

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed
BB_HW_IN_RESET	The LED cannot be controlled as the device is in reset state
BB_HW_NOT_FOUND	The device is not found in a system
BB_BAD_PARAMS	The specified parameters are wrong
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed

E.1.1.11 Bb_ResetControl( )

Name

Bb_ResetControl( )

Synopsis

```
int
Bb_ResetControl
(
    int handle,
    bb_reset_cmd_t cmd,
    bb_reset_id_t dev_id,
    int *param
)
```

Description

This function controls the global resets (external) of baseboard devices.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	Cmd	Command to be executed: • BB_RESET_SET • BB_RESET_CLEAR • BB_RESET_GET (gets reset state)
IN	dev_id	Device ID, can be one of the following: • BB_SOFTWARE_RESET • BB_PERIPHERALS_RESET • BB_ADD_UART_RESET • BB_SAUS_UART_RESET • BB_DEBUG_CRYSTAL_RESET • BB_MEDIA_CARD1_RESET • BB_MEDIA_CARD2_RESET • BB_FIC_RESET • BB_VALLEJO_RESET • BB_FPGA_TX_RESET • BB_FPGA_RX_RESET • BB_ALASKA_RESET • BB_ALL_RESET • BB_SW_UPDATE_RESET • BB_IPMI_TO_INT_RESET
OUT	param	Device reset status (after BB_RESET_GET operation): • BB_RESET_SET • BB_RESET_CLEAR

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation initialization failed

E.1.1.12 Bb_DrvGet_PLL_Number( )

Name

Bb_DrvGet_PLL_Number( )

Synopsis

```
int
Bb_DrvGet_PLL_Number
(
    bb_drv_t *drv,
    int *pll_number
)
```

Description

This function checks for installed PLLs and returns their number and position.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
OUT	pll_number	Number of installed PLLs: 0 – no PLL detected 1 – only PLL_1 detected 2 – only PLL_2 detected 3 – PLL_1 and PLL_2 detected

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

Function can only be called once driver has been initialized.

E.1.1.13 Bb_DrvSet_Primary_Source( )

Name

Bb_DrvSet_Primary_Source( )

Synopsis

```
int
Bb_DrvSet_Primary_Source
(
    bb_drv_t *drv,
    int pll_number,
    int clksrc_id
)
```

Description

This function selects a clock source as a Primary Reference Source for the selected PLL.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	pll_number	PLL number: • BB_PLL_1 • BB_PLL_2
IN	clksrc_id	Clock source ID: • BB_CLK_1A • BB_CLK_2A • BB_CLK_3A • BB_LM_1 • BB_LM_2

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

The function can only be called once the driver has been initialized.

E.1.1.14 Bb_DrvSet_Secondary_Source( )

Name

Bb_DrvSet_Secondary_Source( )

Synopsis

```
int
Bb_DrvSet_Secondary_Source
(
    bb_drv_t *drv,
    int pll_number,
    int clksrc_id
)
```

Description

This function selects a clock source as a Secondary Reference Source for the selected PLL.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	pll_number	PLL number: • BB_PLL_1 • BB_PLL_2
IN	clksrc_id	Clock source ID: • BB_CLK_1B • BB_CLK_2B • BB_CLK_3B • BB_LM_1 • BB_LM_2

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

E.1.1.15 Bb_DrvSet_Sync_Source( )

Name

Bb_DrvSet_Sync_Source( )

Synopsis

```
int
Bb_DrvSet_Sync_Source
(
    bb_drv_t *drv,
    int pll_number,
    int syncsrc_id
)
```

Description

This function selects a synchronization source for the selected PLL.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	pll_number	PLL number: • BB_PLL_1 • BB_PLL_2
IN	syncsrc_id	Synchronization source ID: • BB_PRIMARY_SYNC_SRC • BB_SECONDARY_SYNC_SRC

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

E.1.1.16 Bb_DrvSet_Sync_SelMode( )

Name

Bb_DrvSet_Sync_SelMode( )

Synopsis

```
int
Bb_DrvSet_Sync_Source
(
    bb_drv_t *drv,
    int pll_number,
    int sync_mode
)
```

Description

This function selects a synchronization mode for the selected PLL.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	pll_number	PLL number: • BB_PLL_1 • BB_PLL_2
IN	Sync_mode	Synchronization mode: • BB_MANUAL_SYNC • BB_AUTO_REVERT_SYNC • BB_AUTO_NOREVERT_SYNC

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

The function can only be called once the driver has been initialized.

E.1.1.17 Bb_DrvGet_Sync_Status( )

Name

Bb_DrvGet_Sync_Status( )

Synopsis

```
int
Bb_DrvSet_Sync_Source
(
    bb_drv_t *drv,
    int pll_id,
    bb_PLL_status_t *pll_status
)
```

Description

This function reads the status information for the selected PLL.

Parameters

Type	Name	Description
IN	Drv	Driver unique handle
IN	pll_id	PLL number: • BB_PLL_1 • BB_PLL_2
OUT	pll_status	Status information for the selected PLL

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

E.1.1.18 Bb_DrvSet_Mezz_Ref( )

Name

Bb_DrvSet_Mezz_Ref( )

Synopsis

```
int
Bb_DrvSet_Mezz_Ref
(
    bb_drv_t *drv,
    int mezz_id,
    int clksrc_id
)
```

Description

This function sets the reference signal for the selected Mezzanine Card.

Parameters

Type	Name	Description
IN	Drv	Driver unique handle
IN	mezz_id	Mezzanine ID: • MEDIA_CARD1_ID, not applicable to IXB2850 boards • MEDIA_CARD2_ID
IN	clksrc_id	Clock source ID: • BB_CLK_1A • BB_CLK_1B • BB_CLK_2A • BB_CLK_2B • BB_CLK_3A • BB_CLK_3B • BB_PLL_1_2M • BB_PLL_1_19M • BB_PLL_1_8K • BB_PLL_1_1544K • BB_PLL_2_2M • BB_PLL_2_19M • BB_PLL_2_8K • BB_PLL_2_1544K

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

E.1.1.19 Bb_DrvSet_Backpl_Ref( )

Name

Bb_DrvSet_Backpl_Ref( )

Synopsis

```
int
Bb_DrvSet_Backpl_Ref
(
    bb_drv_t *drv,
    int bclk_id,
    int clksrc_id
)
```

Description

This function sets the backplane reference signal source. It sets only the signal driver. Receivers are selected by the functions Bb_DrvSet_Primary_Source( ) and Bb_DrvSet_Secondary_Source( ).

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	bclk_id	Backplane clock line number: • BB_CLK_1A • BB_CLK_1B • BB_CLK_2A • BB_CLK_2B • BB_CLK_3A • BB_CLK_3B
IN	clksrc_id	Clock source ID: • BB_PLL_1_2M • BB_PLL_1_19M • BB_PLL_1_8K • BB_PLL_1_1544K • BB_PLL_2_2M • BB_PLL_2_19M • BB_PLL_2_8K • BB_PLL_2_1544K

Returns

Return code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

See Also

- Bb_DrvSet_Primary_Source( )
- Bb_DrvSet_Secondary_Source( )

E.1.1.20 Bb_DrvGet_CDC_Config( )

Name

Bb_DrvGet_CDC_Config( )

Synopsis

```
int
Bb_DrvGet_CDC_Config
(
    bb_drv_t *drv,
    bb_CDC_Config_t *bb_CDC_Conf
)
```

Description

This function reads the CDC configuration. The structure is updated by the CDC monitor.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
OUT	bb_CDC_Conf	CDC configuration

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

E.1.1.21 Bb_DrvGet_CDC_Init( )

Name

Bb_DrvGet_CDC_Init( )

Synopsis

```
int
Bb_DrvGet_CDC_Init
(
    bb_drv_t *drv,
    bb_CDC_Config_t bb_CDC_Conf
)
```

Description

This function initializes the CDC configuration with the data passed in the configuration structure.

Parameters

Type	Name	Description
IN	drv	Driver unique handle
IN	bb_CDC_Conf	CDC configuration

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_ERROR	Operation failed

Note

This function can only be called once the driver has been initialized.

E.1.1.22 Bb_ConnectPort( )

Name

Bb_ConnectPort( )

Synopsis

```
int
Bb_ConnectPort
(
    int handle,
    int channelId,
    int portNo
)
```

Description

This function connects the Gigabit Ethernet MAC interface on the baseboard IXF1104 or FIC IXF1104 to the specified backplane interface port. The only possible connections are:

- Onboard 82546 Dual Port Gigabit Ethernet Controller ports to the BB_ATCA_BASE_CHANNEL_1 or BB_ATCA_BASE_CHANNEL_2.
- Onboard IXF1104 ports to the BB_ATCA_FABRIC_CHANNEL_1
- IXF1104 ports on FIC to the BB_ATCA_FABRIC_CHANNEL_2

Traffic from the base channels is processed directly by the Intel XScale core and traffic from the fabric channels is passed to the microengines.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	channelId	Channel ID. Backplane interface channel can be one of the following: • BB_ATCA_BASE_CHANNEL_1 • BB_ATCA_BASE_CHANNEL_2 • BB_ATCA_FABRIC_CHANNEL_1 • BB_ATCA_FABRIC_CHANNEL_2
IN	portNo	Port Number within the confines of particular backplane interface channel: • Fabric interface channel – ports 0-3 are available • Base interface channel – port 0 is available

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation failed

E.1.1.23 Bb_DisconnectPort( )

Name

Bb_ConnectPort( )

Synopsis

```
int
Bb_DisconnectPort
(
    int handle,
    int channelId,
    int portNo
)
```

Description

This function disconnects the Gigabit Ethernet MAC interface on the baseboard or FIC IXF1104 from the specified backplane interface port.

Parameters

Type	Name	Description
IN	handle	Driver unique handle
IN	channelId	Channel ID. Backplane interface channel can be one of the following: • BB_ATCA_BASE_CHANNEL_1 • BB_ATCA_BASE_CHANNEL_2 • BB_ATCA_FABRIC_CHANNEL_1 • BB_ATCA_FABRIC_CHANNEL_2
IN	portNo	Port Number within the confines of particular backplane interface channel: • Fabric interface channel – ports 0-3 are available • Base interface channel – port 0 is available

Returns

Return Code	Description
BB_SUCCESS	Operation completed successfully
BB_FAILURE	Operation failed

E.1.2 Baseboard Driver API Structures

The baseboard driver API uses the following data structures:

- PLL Status Structure
- PLL Status Structure
- CDC Configuration Structure

E.1.2.1 PLL Status Structure

The PLL status structure is defined as follows:

```
typedef struct bb_PLL_status
{
    int PLLstate;/* Operation status: LOCK'ed, HOLDOVER, FREERUN */
    int PLLRefSel;/* Primary input, Secondary Input */
    int PLLRefSelRevert;/* Reference selector for AutoRevert Sync Mode */
    int PLL_FCS;/* FCS = 1.1Hz, FCS = 0.1Hz */
    PLL_SyncMode;/* BB_MANUAL_SYNC, BB_AUTO_REVERT_SYNC, BB_AUTO_NONREVERT_SYNC */
    int PLL_PrimAqStatus;
    int PLL_SecAqStatus;
    int PLL_HoldoverCounter;
} bb_PLL_status_t;
```

E.1.2.2 CDC Configuration Structure

The CDC configuration structure is defined as follows:

```
typedef struct bb_CDC_Config
{
    int PLL_number;/* 0, 1, 2 */
    int PLL_1_reference[2];/* primary_reference, secondary_reference */
    int PLL_2_reference[2];/* primary_reference, secondary_reference */
    int LM_1_input;
    int LM_2_input;
    int clk_1A_source;
    int clk_1B_source;
    int clk_2A_source;
    int clk_2B_source;
    int clk_3A_source;
    int clk_3B_source;
    int PLL_converg_mode;
    int gpio_freq;
    int gpio_input;
} bb_CDC_Config_t;
```

E.2 Gigabit Ethernet Media Driver API

The baseboard driver API is described under the following topics:

- Gigabit Ethernet Media Driver API Functions
- Gigabit Ethernet Media Driver API Structures

E.2.1 Gigabit Ethernet Media Driver API Functions

E.2.1.1 GbEMAC_DeviceStart( )

Name

GbEMAC_DeviceStart( )

Synopsis

```
void
GbEMAC_DeviceStart
(
    unsigned int arg_PortMask,
    unsigned int *arg_Handle,
    unsigned int arg_mode
)
```

Description

This function is called by the application to open the device. It checks whether the requested port is on the baseboard or on one of the daughter boards, and it opens it for use or initializes if it is being called for the first time.

Parameters

Type	Name	Description
IN	arg_PortMask	The Port Mask, indication the ports to be opened: • 0:3 – Four ports on the first media mezzanine card (DB#1); not used on IXB2850 boards • 4:7 – Four ports on the second media mezzanine card (DB#2) • 8:11 – Four ports on baseboard IXF1104 (Quad Gigabit Ethernet Controller) connected to fabric interface channel 1 • 12:15 – Four ports on FIC IXF1104 connected to fabric interface channel 2
IN/OUT	arg_Handle	Handle pointer – filled in after successful return
IN	arg_mode	Indicates fiber/Cu mode, duplex and speed of operation, see Table 70

Returns

Return Code	Description
SUCCESS (=0)	Success
GBEPHY_STATUS_OK (=0)	Success
ERROR_INVALID_HANDLE	Invalid port mask in arg_Handle parameter
ERROR_TASK_LOCK_FAILED	Task lock could not be done
SEMAPHORE_COULD_NOT_CREATED	Semaphore could not be created

Return Code	Description
GBEPHY_WRITE_REG_FAIL	Error occurred while trying to write to hardware register
GBEPHY_INVALID_MODE	Invalid arg_mode parameter
-1	Other error

Notes

The GbEMAC_DeviceStart( ) function can open multiple Gigabit Ethernet devices and multiple ports simultaneously. This is accomplished using the bit port mask, **arg_PortMask**, as the first argument of the function call. Any combination of valid ports can be used.

The GbEMAC_DeviceStart( ) function configures ports to the operational mode specified in its third argument, **arg_mode**. Operational mode determines the following:

- Media type: fiber or copper
- Speed: 10/100/1000 bps
- Duplex: HD or FD
- Auto-negotiation for the Gigabit Ethernet port
- Bus parity used on the SPI port: even or odd

Table 70. Interpretation of the arg_mode value

Bit position	Associated to	Value interpretation
4	SPI-3 Block Mode	**0** - Single Physical Device (SPHY) 4x8 Mode **1** - Multiple Physical Device (MPHY) Mode
3	SPI-3 Parity	**0** - Odd Parity **1** - Even Parity
2-0	Channel, duplex and speed selection mode	**000** - Fiber Mode **001** - Copper 1000 Half Duplex **010** - Copper 1000 Full Duplex **011** - Copper 100 Half Duplex **100** - Copper 100 Full Duplex **101** - Copper 10 Half Duplex **110** - Copper 10 Full Duplex **111** - Copper Auto-sense

Note: Speed auto-negotiation is not supported on IXB2850 board base interface ports. These ports operate only in 1 GB full-duplex copper mode.

Note: There is a bit in the mode parameter to specify either 4x8 or 1x32 SPI bus mode. However, this bit is ignored by the driver for IXB2850 boards.

Using a separate GbEMAC_DeviceStart( ) for each port is recommended. Applications may want to apply different configurations for individual ports and all ports specified by **arg_PortMask** are opened in the same operational mode. To change the operational mode of a port once it is set, the application needs to close the port and then open it again, specifying the new mode.

Ports can be opened by multiple applications for simultaneous use.

The application receives a handle that it must use for any future port reference. The application provides a variable for the driver to store that handle.

Caution: It is important to initialize the variable to 0 before calling the GbEMAC_DeviceStart( ) function. Otherwise, the driver may interpret a non-zero handle as a reference to

previously opened ports. The application should sanity check the returned handle for a non-zero value. For details about the handle structure, see the ixf1104ce_driver_api.h header file.

The device driver is designed to track the number of applications using the Gigabit Ethernet ports/devices and will not close the device until all ports are closed.

E.2.1.2 GbEMAC_DeviceStop( )

Name

GbEMAC_DeviceStop( )

Synopsis

```
void
GbEMAC_DeviceStop
(
    unsigned int arg_PortMask,
    unsigned int *arg_Handle
)
```

Description

This routine is called by the application to close the device. If it is the last application to use the driver it performs deinitialization.

Parameters

Type	Name	Description
IN	arg_PortMask	The Port Mask, indication the ports to be opened: • 0:3 – Four ports on the first media mezzanine card (DB#1); not applicable to IXB2850 boards • 4:7 – Four ports on the second media mezzanine card (DB#2) • 8:11 – Four ports on the baseboard IXF1104 connected to fabric interface channel 1 • 12:15 – Four ports on FIC IXF1104 connected to the fabric interface channel 2
IN	arg_Handle	Pointer to handle obtained during device open call

Returns

Return Code	Description
SUCCESS (=0)	Success
ERROR_INVALID_HANDLE	Invalid arg_Handle parameter
ERROR_MAC_NOT_INITIALIZED	Device was not initialized
ERROR_TASK_LOCK_FAILED	Task lock could not be done

Note

If the value in **arg_PortMask** is not the same as the one specified during the open to obtain the handle, one or more ports specified during the open will not be closed. The handle remains valid and non-zero until all ports previously opened by the application are closed.

E.2.1.3 GbEMAC_Ioctl( )

Name

GbEMAC_Ioctl( )

Synopsis

```
void  GbEMAC_Ioctl
(
    unsigned int *arg_Handle,
    unsigned int arg_IoctlCommand,
    void *arg_pIoctlStruct
)
```

Description

This function performs IOCTL operation on the driver.

Parameters

Type	Name	Description
IN	arg_Handle	Pointer to handle obtained during device open call
IN	arg_IoctlCommand	IOCTL command to be executed
IN/OUT	arg_pIoctlStruct	IOCTL structure that holds values that are read or should be written from/to the driver

Returns

Return Code	Description
SUCCESS (=0)	Success
ERROR_INVALID_HANDLE	Invalid arg_Handle parameter
IOCTL_BUFFER_POINTER_NULL	Parameter arg_pIoctlStruc is NULL
IOCLT_ERROR_INPUT_PORT_NUMBER_INVALID	Port number in structure arg_pIoctlStruc is invalid
ERROR_MAC_NOT_INITIALIZED	Device was not initialized
IOCTL_PORT_NOT_OPEN	Given port was not opened
IOCTL_ERROR_UNKNOWN_IOCTL_CODE	Parameter arg_IoctlCommand is invalid
MODE_ALREADY_SET_IN_SPECIFIED_DUPLEX_MODE	Specified duplex mode already set
ERROR_INVALID_DUPLEX_MODE	Invalid duplex mode

Note

The handle is used by the driver to check if the caller has the rights to operate the device, (that is, whether it previously opened the ports).

The parameter **arg_IoctlCommand** specifies which operation to perform. The **arg_pIoctlStruct** parameter structure specifies the port to be operated on in addition to a pointer to a storage variable. This variable contains either the value used by a change operation or the returned value at the completion of the operation.

The driver performs validation of the parameters and performs all set up actions required for the operation (for example, resets, related registers check, etc.). The **ioctl** commands that pertain to the global device registers ignore port numbers specified within the **arg_pIoctlStruct** parameter structure.

An example of **ioctl** use is setting the MAC address of the port (when required by the application) using the **SET_STN_ADDR** command.

Note: The **arg_pIoctlStruct** for the Station (MAC) address set(/get) must contain a 64-bit value.

Another example is reading port statistics with commands ranging from **GET_RX_OCTETS_OK** to **GET_FC_COLLISION_SEND**.

The following tables shows how the GbEMAC_Ioctl( ) SET/GET commands map to the IXF1104 MAC registers.

Table 71. GbEMAC_Ioctl( ) SET commands and IXF1104 register map

Ioctl	MAC Register
MAC Control Registers	
SET_DUPLEX_MODE	DesiredDuplex
SET_FDFC_TYPE	FDFCType
SET_COLLISION_DIST	CollisionDistance
SET_COLLISION_THLD	CollisionThreshold
SET_FCTX_TIMER	FCTxTimerValue
SET_FDFC_ADDR	FDFCAddressLow, FDFCAddressHigh
SET_IPG_RECEIVE_TIME2	PGReceiveTime2
SET_IPG_TRANSMIT_TIME	PGTransmitTime
SET_PAUSE_THRESHOLD	PauseThreshold
SET_MAX_FRAME_SIZE	MaxFrameSize
SET_MAC_IF_MODE	MacIFMode
SET_FLUSH_TX	FlushTX
SET_FC_MODE	FCEnable
SET_FC_BACK_PRESSURE_LEN	FCBackPressureLength
SET_SHORT_RUNT_TH	ShortRuntsThreshold
SET_UNKNOWN_FRAME_STT	DiscardUnknownControlFrame
SET_TX_CONFIG_WORD	TxConfigWord
SET_DIV_CONFIG_WORD	DiverseConfigWrite
SET_PKT_FILTER_CTL	PacketFilterControl
SET_MUL_PORT_ADD	PortMulticastAddressLow, PortMulticastAddressHigh
Global Status & Configuration Registers	
SET_PORT_STATUS	Port Enable
SET_INTERFACE_MODE	Interface Mode
SET_MAC_SOFT_RESET	MAC Soft Reset
SET_MDIO_RESET	MDIO Soft Reset
SET_UI_ENDIAN_MODE	Microprocessor Interface
SET_LED_MODE	LED Control
SET_LED_FLASH_RATE	LED Flash Rate
SET_LED_FAULT_ACTION	LED Fault Disable
RX FIFO Registers	

Table 71. GbEMAC_Ioctl( ) SET commands and IXF1104 register map (Continued)

Ioctl	MAC Register
SET_RFIFO_HIGH_WATERMARK	RX FIFO High Watermark Port 0, RX FIFO High Watermark Port 1, RX FIFO High Watermark Port 2, RX FIFO High Watermark Port 3
SET_RFIFO_LOW_WATERMARK	RX FIFO Low Watermark Port 0, RX FIFO Low Watermark Port 1, RX FIFO Low Watermark Port 2, RX FIFO Low Watermark Port 3
SET_RX_FIFO_PORT_RESET	RX FIFO Port Reset
SET_RX_FIFO_FRAME_DROP	RX FIFO Errored Frame Drop Enable
SET_CAPTURE_ENABLE_RX_FIFO	RX FIFO Loopback Enable
SET_PRE_PENDING_CRC_ENABLE	RX FIFO Padding and CRC Strip Enable
SET_JUMBO_PKT_SIZE	RX FIFO Jumbo Packet Size Port 0, RX FIFO Jumbo Packet Size Port 1, RX FIFO Jumbo Packet Size Port 2, RX FIFO Jumbo Packet Size Port 3
TX FIFO Registers	
SET_TX_FIFO_HIGH_WATERMARK	TX FIFO High Watermark Port 0, TX FIFO High Watermark Port 1, TX FIFO High Watermark Port 2, TX FIFO High Watermark Port 3
SET_TX_FIFO_LOW_WATERMARK	TX FIFO Low Watermark Port 0, TX FIFO Low Watermark Port 1, TX FIFO Low Watermark Port 2, TX FIFO Low Watermark Port 3
SET_MAC_THRESHOLD	TX FIFO MAC Threshold Port 0, TX FIFO MAC Threshold Port 1, TX FIFO MAC Threshold Port 2, TX FIFO MAC Threshold Port 3
SET_LOOP_RX_TX	Loop RX Data to TX FIFO
SET_TX_FIFO_PORT_RESET	TX FIFO Port Reset
SET_TX_MINI_FRAME_SIZE	TX FIFO Mini Frame Size for MAC and Padding Enable Port 0 to 3
MDIO Registers	
SET_MDIO_CMD_ADDR	MDI Single Command
SET_MDIO_SINGLE_RW_DATA	MDI Single Read and Write Data
SET_AS_PHY_ADDR	Autoscan PHY Address Enable
SET_MDIO_CTL	MDI Control
SPI-3 Registers	
SET_SPI3_TX_CONFIG	SPI-3 Transmit and Global Configuration
SET_SPI3_RX_CONFIG	SPI-3 Receive Configuration
SerDes Registers	
SET_ACDC_COUPLING	Tx and Rx ACDC Coupling Selection
SET_RX_DATA_SYNC	Clock and IF Mode Change Enable Ports 0-3
GBIC Registers	
SET_GBIC_CTL	GBIC Control Ports 0-3
SET_I2C_CTL_DATA	I2C Control Ports 0-3

Table 72. GbEMAC_Ioctl( ) GET commands and IXF1104 register map

Ioctl	MAC Register
MAC Control Registers	
GET_DUPLEX_MODE	DesiredDuplex
GET_FDFC_TYPE	FDFCType
GET_COLLISION_DIST	CollisionDistance
GET_COLLISION_THLD	CollisionThreshold
GET_FCTX_TIMER	FCTxTimerValue
GET_FDFC_ADDR	FDFCAddressLow, FDFCAddressHigh
GET_IPG_RECEIVE_TIME1	IPGReceiveTime1
GET_IPG_RECEIVE_TIME2	IPGReceiveTime2
GET_IPG_TRANSMIT_TIME	IPGTransmitTime
GET_PAUSE_THRESHOLD	PauseThreshold
GET_MAX_FRAME_SIZE	MaxFrameSize
GET_MAC_IF_MODE	MacIFMode
GET_FLUSH_TX	FlushTX
GET_FC_MODE	FCEnable
GET_FC_BACK_PRESSURE_LEN	FCBackPressureLength
GET_SHORT_RUNT_TH	ShortRuntsThreshold
GET_UNKNOWN_FRAME_STT	DiscardUnknownControlFrame
GET_RX_CONFIG_WORD	RxConfigWord
GET_TX_CONFIG_WORD	TxConfigWord
GET_DIV_CONFIG_WORD	DiverseConfigWrite
GET_PKT_FILTER_CTL	PacketFilterControl
GET_MUL_PORT_ADD	PortMulticastAddressLow, PortMulticastAddressHigh
MAX RX Statistics Registers	
GET_RX_OCTETS_OK	RxOctetsTotalOK
GET_RX_OCTETS_BAD	RxOctetsBAD
GET_RX_UC_PKTS	RxUCPckts
GET_RX_MC_PKTS	RxMCPkts
GET_RX_BC_PKTS	RxBCPkts
GET_RX_PKTS_64	RxPkts64Octets
GET_RX_PKTS_65_127	RxPkts65to127Octets
GET_RX_PKTS_128_255	RxPkts128to255Octets
GET_RX_PKTS_256_511	RxPkts256to511Octets
GET_RX_PKTS_512_1023	RxPkts512to1023Octets
GET_RX_PKTS_1024_1518	RxPkts1024to1518Octets
GET_RX_PKTS_1519_MAX	RxPkts1519toMaxOctets
GET_RX_FCS_ERR	FCSErrors
GET_VLAN_TAG	Tagged
GET_RX_DATA_ERR	RxDataErro

Table 72. GbEMAC_Ioctl( ) GET commands and IXF1104 register map (Continued)

Ioctl	MAC Register
GET_RX_ALLIGN_ERR	Align Errors
GET_RX_LONG_ERR	LongErrors
GET_RX_JABBER_ERR	JabberErrors
GET_RX_PAUSE_MAC_CTL	PauseMacControlReceivedCounter
GET_RX_UNKNOWN_CTL_FRAME	UnknownMacControlFrameCounter
GET_VLONG_ERR	VeryLongErrors
GET_RUNT_ERR	RuntErrors
GET_SHORT_ERR	ShortErrors
GET_SEQ_ERR	SequenceErrors
GET_SYMBOL_ERR	SymbolErrors
MAX TX Statistics Registers	
GET_TX_OCTETS_OK	RxOctetsTotalOK
GET_TX_OCTETS_BAD	RxOctetsBAD
GET_TX_UC_PKTS	RxUCPckts
GET_TX_MC_PKTS	RxMCPkts
GET_TX_BC_PKTS	RxBCPkts
GET_TX_PKTS_65_127	RxPkts65to127Octets
GET_TX_PKTS_64	RxPkts64Octets
GET_TX_PKTS_128_255	RxPkts128to255Octets
GET_TX_PKTS_256_511	RxPkts256to511Octets
GET_TX_PKTS_512_1023	RxPkts512to1023Octets
GET_TX_PKTS_1024_1518	RxPkts1024to1518Octets
GET_TX_PKTS_1519_MAX	RxPkts1519toMaxOctets
GET_TX_DEFERRED_ERR	DeferredTx
GET_TX_TOTAL_COLLISION	TxTotalCollisions
GET_TX_SINGLE_COLLISION	TxSingleCollisions
GET_TX_MUL_COLLISION	TxMultipleCollisions
GET_LATE_COLLISION	TxLateCollisions
GET_TX_EXCV_COLLISION	ExcessiveCollisionErrors
GET_TX_EXCV_DEFERRED_ERR	ExcessiveDeferralErrors
GET_TX_EXCV_LEN_DROP	TxExcessiveLengthDrop
GET_TX_UNDERRUN	TxUnderrun
GET_TX_VLAN_TAG	Tagged
GET_TX_CRC_ERR	CRCError
GET_TX_PAUSE_FRAME	TxPauseFrames
GET_FC_COLLISION_SEND	FlowControlCollisionsSend
Global Status and Configuration Registers	
GET_PORT_STATUS	Port Enable
GET_INTERFACE_MODE	Interface Mode
GET_MAC_SOFT_RESET	MAC Soft Reser

Table 72. GbEMAC_Ioctl( ) GET commands and IXF1104 register map (Continued)

Ioctl	MAC Register
GET_MDIO_RESET	MDIO Reset
GET_UI_ENDIAN_MODE	Microprocessor Interface
GET_LED_MODE	LED Control
GET_LED_FLASH_RATE	LED Flash Rate
GET_LED_FAULT_ACTION	LED Fault Disable
GET_JTAG_ID	JTAG ID (Device Revision)
RX FIFO Registers	
GET_RFIFO_HIGH_WATERMARK	RX FIFO High Watermark Port 0, RX FIFO High Watermark Port 1, RX FIFO High Watermark Port 2, RX FIFO High Watermark Port 3
GET_RFIFO_LOW_WATERMARK	RX FIFO Low Watermark Port 0, RX FIFO Low Watermark Port 1, RX FIFO Low Watermark Port 2, RX FIFO Low Watermark Port 3
GET_RX_FIFO_PORT_RESET	RX FIFO Port Reset
GET_RX_FIFO_FRAME_DROP	RX FIFO Errored Frame Drop Enable
GET_RX_FIFO_OVERFLOW_STT	RX FIFO Overflow Event
GET_DROPPED_PKTS	RX FIFO Number of Error Packets Dropped Port 0, RX FIFO Number of Error Packets Dropped Port 1, RX FIFO Number of Error Packets Dropped Port 2, RX FIFO Number of Error Packets Dropped Port 3
GET_CAPTURE_ENABLE_RX_FIFO	RX FIFO Loopback Enable
GET_PRE_PENDING_CRC_ENABLE	RX FIFO Padding and CRC Strip Enable
GET_MATCHING_PATTERN	
GET_JUMBO_PKT_SIZE	RX FIFO Jumbo Packet Size Port 0, RX FIFO Jumbo Packet Size Port 1, RX FIFO Jumbo Packet Size Port 2, RX FIFO Jumbo Packet Size Port 3
TX FIFO Registers	
GET_TX_FIFO_HIGH_WATERMARK	TX FIFO High Watermark Port 0, TX FIFO High Watermark Port 1, TX FIFO High Watermark Port 2, TX FIFO High Watermark Port 3
GET_TX_FIFO_LOW_WATERMARK	TX FIFO Low Watermark Port 0, TX FIFO Low Watermark Port 1, TX FIFO Low Watermark Port 2, TX FIFO Low Watermark Port 3
GET_MAC_THRESHOLD	Threshold Port 1, TX FIFO MAC Threshold Port 2, TX
GET_TX_FIFO_OVERFLOW_STT	TX FIFO Overflow/Underflow Event/Out of Sequence
GET_LOOP_RX_TX_STT	Loop RX Data to TX FIFO
GET_TX_FIFO_PORT_RESET	TX FIFO Port Reset
GET_TX_DROP_FRAME	TX FIFO Number of Frames Removed Port 0, TX FIFO Number of Frames Removed Port 1, TX FIFO Number of Frames Removed Port 2, TX FIFO Number of Frames Removed Port 3
GET_TX_DROP_PKTS	TX FIFO Number of Dropped Packets Port 0, TX FIFO Number of Dropped Packets Port 1, TX FIFO Number of Dropped Packets Port 2, TX FIFO Number of Dropped Packets Port 3
GET_TX_OCCUPANCY	TX FIFO Occupancy Counter for Port 0, TX FIFO Occupancy Counter for Port 1, TX FIFO Occupancy Counter for Port 2, TX FIFO Occupancy Counter for Port 3
GET_TX_MINI_FRAME_SIZE	TX FIFO Mini Frame Size for MAC and Padding Enable Port 0 to 3

Table 72. GbEMAC_Ioctl( ) GET commands and IXF1104 register map (Continued)

Ioctl	MAC Register
MDIO Registers	
GET_MDIO_CMD_ADDR	MDI Single Command
GET_MDIO_SINGLE_RW_DATA	MDI Single Read and Write Data
GET_AS_PHY_ADDR	Autoscan PHY Address Enable
GET_MDIO_CTL	MDI Control
SPI-3 Registers	
GET_SPI3_TX_CONFIG	SPI-3 Transmit and Global Configuration
GET_SPI3_RX_CONFIG	SPI-3 Receive Configuration
SerDes Registers	
GET_ACDC_COUPLING	Tx and Rx ACDC Coupling Selection
GET_RX_DATA_SYNC	Clock and IF Mode Change Enable Ports 0-3
GBIC Registers	
GET_GBIC_CTL	GBIC Control Ports 0-3
GET_I2C_CTL_DATA	I2C Control Ports 0-3

E.2.1.4 GbEMAC_Callback( )

Name

GbEMAC_Callback( )

Synopsis

```
GbEMAC_Callback
(
    uint32 *argHandle,
    void * arg_pCallback,
    void * arg_pUserContext
)
```

Description

This function registers the user callback functions to an array, and when the interrupt occurs, all the respective functions are called.

Parameters

Type	Name	Description
IN	argHandle	Pointer to handle obtained during device open call
IN	arg_pCallback	The user application function, whose pointer is passed to this API to register with the driver as a callback function. Whenever an interrupt has been generated, this function is called by the driver library.
OUT	arg_pUserContext	Identifies the application. This is useful when the same callback function is registered by multiple applications otherwise has no significance.

Returns

Return Code	Description
SUCCESS (=0)	Success
ERROR_INVALID_HANDLE	Invalid argHandle parameter
NO_MORE_CALLBACK_FOR_GBIC_INTERRUPT	Callback array is full – no more applications can be registered to use the driver

Note

The application can register to receive link state change notifications by calling GbEMAC_Callback( ). When GbEMAC_Callback( ) is called, it registers the user callback, **arg_pCallback**, called by the driver each time the link state changes on ports specified by **argHandle**. The handle must be a valid handle (one received by the application at port opening).

The third argument of the registration function, **arg_pUserContext**, is a variable that is returned by the driver after the GbEMAC_Callback( ) function is called to identify the application. This is useful when the same callback function is registered by multiple applications.

E.2.2 Gigabit Ethernet Media Driver API Structures

The Gigabit Ethernet Media driver API uses following structures to present some of its internal attributes to the user:

```
/* These structures are used in GbEMAC_Ioctl() function to carry value */
/* that user whishes to write or read to/from internal driver structures */
/* This structure is used for 32-bit values */

typedef struct gbe_mac_s_ioctl_ptr
{
    uint32 portIndex;/* Port Number */
    uint32  value;/* Buffer which hold the value to be set/get */
} gbe_mac_ioctl_ptr;


/* This structure is used for 64-bit values (addresses and counters) */

typedef struct gbe_mac_s_ioctl_ptr_64 {
    uint32 portIndex;/* Port Number */
    uint32 valueHigh;/* High 47:32 bits of the address */
    uint32 valueLow;/* Low 31:0 bits of the address */
} gbe_mac_ioctl_ptr_64;
```