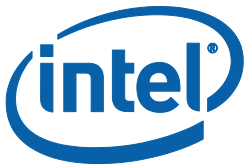

Intel® Xeon Phi™ Coprocessor

Intel® Manycore Platform Software Stack (Intel® MPSS)

Boot Configuration Guide

Document Number: 328344-001US

Revision: 0.6

World Wide Web: http://www.intel.com

Disclaimer and Legal Information

Revision History

Document Number	Revision Number	Description	Revision Date
xxxx	0.1	Initial draft	9/2012
328344-001	0.2	Initial editorial pass; incorporated excerpts from Intel® MPSS Readme	11/2012
328344-001	0.3	Incorporated technical review edits	1/2013
328344-001	0.4	Added chapters 7 & 8; added figures; minor edits	1/2013
328344-001	0.5	Changed sections 4.4.4.2 and 5.3.1.1.	4/2013
328344-001	0.6	Minor edits in Sections 4.4.4.3, 4.4.4.4, 5.3.1.2, 5.3.1.3, 5.3.1.4, and 8.1.1.	6/2013

Contents

List of Figures

1 Introduction

The Intel® Xeon Phi™ coprocessors are PCIe based add-in cards that run a version of Linux* tailored for these coprocessors. The Linux* OS for Intel® Xeon Phi™ coprocessors, as well as a range of drivers and utilities, are included in the Intel® Manycore Platform Software Stack (Intel® MPSS). The responsibilities of these drivers and utilities include:

- Placing the Linux* boot image and root file system into coprocessor memory.
- Controlling coprocessor booting, shutdown and reset.
- Providing a virtual console.
- Providing an IP (over PCIe) networking connection to each coprocessor.
- Directing power management of each coprocessor.
- Supporting high speed data transfer to and from the coprocessor.

The PCIe bus is the only communication channel available to the Intel ® Xeon Phi™ coprocessors. Therefore configuration and provisioning of the OS to be executed on each Intel® Xeon Phi™ coprocessor is performed by the host system in which the coprocessor is installed.

The Linux* kernel and file system image for the Intel ® Xeon Phi™ coprocessors are installed into the host file system as part of Intel® MPSS installation. The coprocessor file system image can be configured through the use of the **micctrl** utility described below and/or directly by the host root.

The **mic.ko** driver is the component of Intel® MPSS that provides PCIe bus access and implements the coprocessor boot process. To boot a coprocessor, mic.ko injects the Linux* kernel image and a kernel command line into coprocessor memory and signals it to begin execution. A virtual console driver and a virtual network driver are also built into mic.ko. Finally, mic.ko directs power management of the installed coprocessors and provides a high speed data transfers over PCIe through its Intel® Symmetric Communications Interface (SCIF) driver.

The **mpssd** daemon directs the initialization and booting of the Intel® Xeon Phi™ coprocessor cards based on a set of configuration files. mpssd is started and stopped as a Linux* service and instructs the cards to boot or shutdown. It supplies the final file system image to the cards when requested. mpssd will also log debug information from each coprocessor.

micctrl is a utility through which the user can control (boot, shutdown, reset) each of the installed Intel® Xeon Phi™ coprocessors. micctrl also offers numerous options to simplify the process of configuring each coprocessor. Section 5 of this document, “The micctrl Utility”, describes the **micctrl** utility in detail.

2 Post Installation Quick Configuration

After the installation of the RPMs (consult the readme.txt file for installation instructions), the system administrator must complete the Intel® Xeon Phi™ coprocessor configuration before starting the Intel® MPSS service.

2.1 Step 1: Ensure Root Access

User access to the Intel® Xeon Phi™ coprocessor node is provided through the secure shell utilities. Ensure the **root** user has ssh keys by looking in the */root/.ssh* directory for either the **id_rsa.pub** or **id_dsa.pub** key files. If no SSH keys exist, use the **ssh-keygen** command to generate a set:

```
user_prompt> ssh-keygen
```

2.2 Step 2: Generate the Default Configuration

Each Intel® Xeon Phi™ coprocessor card has a unique configuration file in the */etc/sysconfig/mic* directory. Initialize the default configuration for the Intel® Xeon Phi™ coprocessors installed on the system:

```
user_prompt> sudo micctrl --initdefaults
```

The **micctrl --initdefaults** command creates and populates default configuration values into Intel® MPSS specific configuration files. These configuration files, default.conf and micN.conf, are located at */etc/sysconfig/mic/*, where N is an integer number (0, 1, 2, 3, etc.) that identifies each coprocessor installed in the system. default.conf is common to all installed Intel® Xeon Phi™ coprocessors.

2.3 Step 3: Change Configuration

Examine the default.conf and micN.conf files in the /etc/sysconfig/mic directory. If the default configuration meets the requirements of the system, continue to Step 4. Otherwise, edit the configuration files in /etc/sysconfig/mic (refer to Section 4, "Configuration").

2.4 Step 4: Start the Intel® MPSS Service

The default configuration specifies that each Intel® Xeon Phi™ coprocessor card is booted when the Intel® MPSS service is started. To start the Intel® MPSS service, execute the Linux* service command:

```
user_prompt> sudo service mpss start
```

The call to service will exit when it determines the Intel® Xeon Phi™ coprocessor cards have either booted successfully or failed to boot, and the status of the cards will be displayed.

3 Intel® MPSS Boot Process

Booting the Linux* kernel on the Intel® Xeon Phi™ coprocessor card requires a number of steps. Figure 1 shows the sequence of steps that are performed during the Intel® MPSS boot process.

Figure 1 Boot process for Intel® MPSS

3.1 Booting the Intel® Xeon Phi™ Coprocessor Card

This section describes the key steps that are performed during the Intel® MPSS boot process on the Intel® Xeon Phi™ coprocessor card.

3.1.1 Kernel Command Line

On most Intel® based systems, loading and executing the Linux* kernel image is controlled by the **grub** boot loader. In the **grub** configuration file, each possible kernel definition contains a number of parameters to be passed to Linux* through its kernel command line. In the Intel® MPSS boot system, this is done by the **mpssd** parsing its configuration files. The kernel command line is created based on values in the configuration files and placed in */sys/class/mic/mic<id>/cmdline* for the driver to retrieve it.

3.1.2 Instruct the Driver to Boot the Intel® Xeon Phi™ Coprocessor Card

The **mpssd** requests the Intel® Xeon Phi™ coprocessor card to start executing the Linux* image by writing a boot string to the */sys/class/mic/mic<id>/state* file. This file is a link into the MIC driver through a Linux* sysfs entry. The format of the request must be:

```
boot:linux:<Linux* image file name>
```

The options **reset** and **shutdown** may also be written to **state** entry and will be discussed later.

The second part of the boot argument indicates to boot a Linux* image. It may also be set to **elf** to indicate booting a standard ELF format file but documenting this is beyond the scope of this document.

When the driver receives the boot request, it first checks to see whether the card is in the **ready** state. If the card is not ready to boot it will return an error through the write call to the sysfs entry and not attempt to boot the card. Otherwise it sets the state of the Intel® Xeon Phi™ coprocessor card to **booting**.

The driver then saves the image file name for later retrieval through the */sys/class/mic/mic<id>/image* sysfs entry. It also sets the mode to indicate it is booting a Linux* image

The driver will copy the kernel command line setting request by the **mpssd** daemon, along with a number of addresses in host memory required by various drivers in the Linux* image. It then copies the requested Linux* image file into the Intel® Xeon Phi™ coprocessor's memory.

The last step is to write to the Intel® Xeon Phi™ coprocessor register instructing it to start executing the injected image.

3.1.3 Linux* Kernel Executes

Executing the Linux* kernel code functions as it does on any Intel® based machine. It initializes hardware, starts kernel services, and sets all the CPUs to the "online" state. When the kernel is ready, it initializes its attached initial ram disk image and starts executing the **init** script in the image.

As on any Intel® based Linux* system, the initial ram disk contains the loadable modules required for the real root file system. Many of the arguments passed in the kernel command line are addresses required for the modules to access host memory. The **init** script parses the kernel command line for needed information and loads the driver modules.

The last step is for the **init** script to check the **root=** parameter in the kernel command line for the type of device containing the root file system, and take the appropriate actions.

3.1.4 Root is the Initial Ram Disk

Setting the root to be the initial ram disk is for debug purposes only. The initial ram disk contains only a minimal set of tools and utilities.

3.1.5 Root is a Ram Disk Image

If the root is set to be a ram file system, the **init** script must first download the file system information from the host. It makes a request to the **mpssd** daemon through an Intel® Symmetric Communications Interface (SCIF) connection and receives a compressed cpio format file.

After the file has been downloaded, the **init** script creates a **tmpfs** (Linux* ram disk file system type) in Intel® Xeon Phi™ coprocessor card memory and extracts the file system information into it. This image must contain everything needed to start a fully functional Linux* system.

The ram disk image is activated as the root device by calling the Linux* **switch_root** utility. This special utility instructs the Linux* kernel to remount the root device on the tmpfs mount directory, release all file system memory references to the old initial ram disk and start executing the new */sbin/init* function.

/sbin/init performs the normal Linux* user level initialization. All the information required must have already been in the compressed cpio file.

3.1.6 Root is a NFS Export

If a NFS mount is indicated to supply the root device, the **init** script will initialize the **mic0** virtual network interface to the IP address supplied on the kernel command line and mount the NFS export from the host.

As in the ram disk image, the NFS mount is activated as the root device by calling the Linux* **switch_root** utility. This special utility instructs the Linux* kernel to remount the root device on the NFS mount directory, release all file system memory references to the old initial ram disk and start executing the new */sbin/init* function.

/sbin/init performs the normal Linux* user level initialization. All the information required must have already been in the NFS export.

3.1.7 Notify the Host that the Intel® Xeon Phi™ Coprocessor System is Ready

The last step of any of the three initializations is to notify the host that the coprocessor card is ready for access. It does this by writing to its */sys/class/micnotify/notify/host_notified* entry. This causes an interrupt into the host driver which updates the card's state to **online**.

4 Configuration

This section focuses on configuring Intel® Xeon Phi™ coprocessor cards, including configuration files, kernel command line parameters, and authentication.

4.1 Configurable Components

On a typical Linux* system, the installation and configuration process is performed as a series of questions posed by the system and answered by the installer/operator. Since the Intel® Xeon Phi™ coprocessor cards do not have a file system of their own, this process is replaced by the installation of RPMs containing the required software on the host and then configured by a combination of editing of configuration files and using the **micctrl** utility.

The configuration parameters have three categories:

1. Parameters that control loading the Intel® Xeon Phi™ coprocessor Linux* kernel onto the card and initiating the boot process.
2. Parameters to define the root file system to be used on the card.
3. Parameters to configure the host end of the virtual Ethernet connection.

The current configuration parameters can be displayed with the **micctrl --config** command. For example, the default configuration on most systems looks like the following:

```
mic0:
============================================================
  Linux Kernel:   /lib/firmware/mic/uos.img
  BootOnStart:    Enabled
  Shutdowntimeout: 300 seconds

  ExtraCommandLine: highres=off pm_qos_cpu_dma_lat=75

  UserAuthentication: Local

  Root Device:   Dynamic Ram /opt/intel/mic/filesystem/mic0.image
       BaseDir:   /opt/intel/mic/filesystem/base.filelist
       CommonDir: /opt/intel/mic/filesystem/common.filelist
       MicDir:    /opt/intel/mic/filesystem/mic0.filelist
       Overlay:   /opt/intel/mic/coi/config/coi.filelist

  Network:       Static Pair
       Hostname:  sys1-mic0tinynet.com
       Host IP:   172.31.1.254
       MIC IP:    172.31.1.1
       Host MAC:  4a:70:e7:0c:2c:57
       MIC MAC:   22:88:36:2b:0f:97
       Net Bits:  24
       NetMask:   255.255.255.0

  Console:       hvc0
  VerboseLogging: Enabled
```

4.2 Configuration Files

This section briefly discusses configuration file formats and use of the Include parameter to **micctrl**.

4.2.1 File Location and Format

Configuration is controlled by per card configuration files located in the */etc/sysconfig/mic* directory. Each card has an associated micN.conf configuration file, where N is the integer ID of that card (i.e.: mic0.conf, mic1.conf, etc.).

Each of the configuration files contains a list of configuration parameters and their arguments. Each parameter must be on a single line. Comments begin with the '#' character, and terminate at the end of the same line.

4.2.2 Including Other Configuration Files

Parameter syntax:

```
Include <config_file_name>
```

Each of these configuration files can include other configuration files. The **Include** parameter lists the configuration file(s) to be included. The configuration file(s) to be included must be found in *etc/sysconfig/mic*. The configuration parser processes each parameter sequentially. When the **Include** parameter is encountered, the included configuration file(s) are immediately processed. If a parameter is set multiple times, the last instance of the parameter setting will be applied.

By default, the */etc/sysconfig/mic/default.conf* file is included at the start of each coprocessor specific file (e.g. mic0.conf, mic1.conf, etc.). This allows the coprocessor specific files to override any parameter set in **default.conf**.

The last entry in the **default.conf** file is typically (and by default) the line:

```
Include conf.d/*.conf
```

This is a special rule, specifying that all the files in the */etc/sysconfig/mic/conf.d* directory will be included.

4.3 Configuring Boot Parameters

The host system boots the Intel® Xeon Phi™ coprocessor card by injecting the Linux* kernel image and kernel command line into coprocessor memory and then instructing the coprocessor to start. To perform this operation, the host system reads the configuration files, and builds the kernel command line from relevant parameters. By default, the boot parameters are placed in the per-card micN.conf files, allowing each card to be configured independently of the other cards. If a boot parameter is placed in the defaults.conf file, then it will apply to all cards unless overridden

4.3.1 What to Boot

Parameter syntax:

```
OSimage <linux_kernel_image>
```

The OSimage parameter specifies the Intel® Xeon Phi™ coprocessor Linux* OS boot image. The default value is */lib/firmware/mic/uos.img*. The system owner can specify a different kernel image by editing this parameter. The change takes effect upon executing either **service mpss start** or **micctrl -b**.

4.3.2 When to Boot

Parameter syntax:

```
BootOnStart <Enabled | Disabled>
```

The **BootOnStart** parameter controls whether the Intel® Xeon Phi™ coprocessor is booted when the Intel® MPSS service starts. If set to `Enabled`, the **mpssd** daemon will attempt to boot the Intel® Xeon Phi™ coprocessor card when **service mpss start** is called.

4.3.3 VerboseLogging Kernel Command Line Parameter

Parameter syntax:

```
VerboseLogging <Enabled | Disabled>
```

The **VerboseLogging** parameter specifies whether the **quiet** kernel command line parameter is passed to the Intel® Xeon Phi™ coprocessor on boot. The **quiet** kernel parameter suppresses most kernel messages during kernel boot. **VerboseLogging** is enabled by default. Disabling **VerboseLogging** will reduce boot times.

Changes to **VerboseLogging** take effect upon executing **service mpss start**.

NOTE: This parameter may be deprecated in future releases.

4.3.4 Console Kernel Command Line Parameter

Parameter syntax:

```
Console "<console device>"
```

Intel® MPSS software provides a PCIe bus virtual console driver. Its device node (hvc0) is the default value assigned to the **Console** parameter. Other possible values are intended for internal use.

Changes to **Console** take effect upon executing **service mpss start**.

4.3.5 ExtraCommandLine Kernel Command Line Parameter

Parameter syntax:

```
ExtraCommandLine "<string>"
```

The **ExtraCommandLine** parameter specifies kernel command line parameters to pass to the Intel® Xeon Phi™ coprocessor kernel on boot. Drivers for the coprocessor use a number of kernel command line parameters generated by the host driver. Default parameters may be subject to change in future releases.

NOTE: This parameter is mainly intended for internal use. Exercise caution when editing this parameter.

Changes to **ExtraCommandLine** take effect upon executing **service mpss start**.

4.3.6 PowerManagement Kernel Command Line Parameter

Parameter syntax:

```
PowerManagement "<string>"
```

The **PowerManagement** parameter is a string of four attributes passed directly to the kernel command line for the card's power management driver. The **mpssd** daemon and **micctrl** utility do not validate any of the parameters in this string or its format. Consult power management documentation for correct values.

Changes to **PowerManagement** take effect upon executing **service mpss start**.

4.3.7 RootDevice Kernel Command Line Parameter

Parameter syntax:

```
RootDevice <type> [<location> [</usr location>]]
```

The **RootDevice** parameter defines the type of root device to mount. The **type** argument is a string specifying the device type. The **location** argument is the location information of the file system for the Intel® Xeon Phi™ coprocessor card.

Current supported types are **RamFS**, **StaticRamFS, NFS, SplitNFS** and **InitRD**.

The **InitRD** type boots to the initial ram disk image included in the downloaded kernel. This option exists for debug purposes only. **InitRD** does not take a **location** argument.

The **RamFS** type builds a compressed cpio image when a booting coprocessor requests its ram file system image from mpssd. The resulting image is placed at the file location specified by **location** and then downloaded to the **init** process on the booting coprocessor.

The **StaticRamFS** type causes **mpssd** to download the compressed cpio image at **location** to a booting coprocessor. The **StaticRamFS** boot will fail if the image file is not already present at **location**.

Such an image may have been previously created by booting with **RootDevice** type **RamFS**. Optionally, when **RootDevice** is **StaticRamFS**, the **micctrl --updateramfs** command causes a compressed cpio image to be built and placed at the **location** argument to **StaticRamFS**.

The **NFS** type instructs the init script to mount the NFS share specified by the **location** argument as the root file system. The location must be a fully qualified NFS mount location with the format "server:location". For example it may be 10.10.10.12:/export/mic0 or host:/opt/intel/mic/filesystem/mic0.export.

The **SplitNFS** type is the same as **NFS** except it also provides a separate NFS share at **/usr location** to mount as the **/usr** directory on the card.

The effects of changes to **RootDev** take effect upon executing **service mpss start**.

For more information, refer to Section 6.1, "The File System Creation Process".

4.4 Root File System

Every Linux* system needs a root file system with a minimal set of files. Other nonessential files may be on the root or they may be on secondary mounts. Most modern Linux* OS releases assume the root file system will be large enough to install the complete release into. The Intel® Xeon Phi™ coprocessor embedded file system currently follows the same rule.

Files on the root fall into three categories: the binaries installed with the system, the files in the */etc* directory, which are used for configuring parameters uniquely to an individual system, and the set of files for the users of the system.

Intel® MPSS provides a set of configuration parameters that are used in building the root file system image. When a root file system image is (re)built it is controlled by the **RootDevice** parameter. Refer to Section 4.3, "Configuring Boot Parameters" and Section 6.1, "The File System Creation Process" for more information.

4.4.1 File Location Parameters

Parameter syntax:

```
BaseDir <location> <descriptor file>
CommonDir <location> <descriptor file>
MicDir <location> <descriptor file>
OverLay <location> <descriptor file>
```

These parameters collectively specify all the files from which a root file system cpio image is to be built. Each parameter has two required arguments. The **location** parameter is a host subdirectory. The **descriptor file** parameter identifies a file that describes where files in the directory subtree at **location** are to be placed in the Intel® Xeon Phi™ coprocessor card's file system, and the permissions of those files. For more

information on the format of this file, refer to Section 6.3, "Adding Files to the Root File System".

The **BaseDir** parameter is the location of the Intel® Xeon Phi™ coprocessor binaries that were installed with the Intel® MPSS RPMs. The files in this directory should never be changed since the next install will overwrite any changes.

The **CommonDir** parameter defines a set of files that the system administrator wishes to have on all the Intel® Xeon Phi™ coprocessor card file systems. There are no files installed in this directory and added files will be maintained between updates to the Intel® MPSS installation.

The **MicDir** parameter defines the per card information required to make the Intel® Xeon Phi™ coprocessor card unique. There are no files installed in this directory and most of its content is created by the configuration process. Specifically, user access and network configuration each has its own set of configuration parameters.

The **Overlay** parameter is the only one of the set that can be used multiple times. Each time it is listed, it may specify an additional set of files to add to the file system. This parameter is used to add additional software to be automatically included. For an example of its syntax, refer to */etc/sysconfig/mic/conf.d/coi.conf*.

4.4.2 User Access

Parameter syntax:

```
UserAuthentication None
UserAuthentication Local <low uid> <high uid>
```

User authentication in the Intel® Xeon Phi™ coprocessor Linux* system is controlled through populating the standard Unix* */etc/passwd*, */etc/shadow* and */etc/group* files in the Intel® Xeon Phi™ coprocessor root file system.

NOTE: Although the mechanisms for populating these files copy the passwords defined for users on the host system, it is recommended to control access through the secure shell.

NOTE: The passwords do not work with a Suse host because it uses a different encryption algorithm than the card.

The **UserAuthentication** configuration parameter specifies two default sets of users. If **None** is specified, the */etc/passwd* file on the card will default to one containing the **root**, **sshd**, and **micuser** accounts. If **Local** is specified, the */etc/passwd* file on the card will have entries for all users on the host having a user ID (uid) between **low uid** and **high uid**.

Changing **UserAuthentication** can be done in two ways. The first method is to edit the entry in a card specific configuration file and then run **micctrl --resetconfig**. The second, and easier, method is to use the **micctrl --configuser** command.

NOTE: Every user to be populated to the Intel® Xeon Phi™ coprocessor file system should set their secure shell configuration files before running either the **micctrl --resetconfig** or **micctrl --configuser** commands.

NOTE: The current algorithm for setting the **Local** type is not optimal. The **micctrl** utility will attempt to find all users in the host's */etc/passwd* file and */home* directories and populate them to the Intel® Xeon Phi™ coprocessor file system.

After the initial user transfer to the card, additional users may be added to the Intel® Xeon Phi™ coprocessor file system with the **micctrl --useradd** command. For more information, refer to Section 5, "The micctrl Utility".

For Intel® Xeon Phi™ coprocessor cards where it is required to strictly control user access, it is recommended to set **UserAuthentication** to **None** and add each user as required.

4.4.3 Service Startup

During boot, the embedded Linux* OS on the Intel® Xeon Phi™ coprocessor card executes the script files in the */etc/rc3.d* directory. These entries are links to the actual script files in the */etc/init.d* directory. The links are named with the standard Linux* custom starting with an 'S' for start or 'K' for stop followed by the position parameter and then the file name from the init.d directory. The position parameter is a number from 01 to 99 establishing the order the scripts are executed in.

```
Service <name> <start> <stop> <state>
```

The MPSS stack installs several pieces of software with various service scripts. The system administration may not want all of them to actual start at boot. To support this functionality, the configuration files specify the creation of the files in */etc/rc3.d* based on the **Service** configuration parameter. Each file in */etc/init.d* will require a **Service** entry in an Intel® Xeon Phi™ coprocessor configuration file.

The **name** argument is the name of the actual script found in the */etc/init.d* directory.

The **start** argument defines the order the service start relevant to other scripts. It will be a value from 1 to 99. As an example the network interface must be initialized before the secure shell daemon can be started. The **network** script is assigned a start value of 21 and sshd is assigned 80.

The **stop** argument is the opposite of the start parameter and is generally set to 100 minus the **start** value. This will assure on shutdown the secure shell daemon at 5 will shut down before the network is unconfigured at 79.

The **state** argument determines whether the links specifies an 'S' for start or 'K' for stop. It follows the **chkconfig** utility convention of **on** for start and **off** for stop.

4.4.4 Network Access

Parameter syntax:

```
Hostname <name>
HostMacAddress <address>
MicMacAddress <address>
Bridge <name> <type> <IPaddr> <NetBits> [<MTU>]
Network <type> …
```

On the host operating system files are added to the network configuration based on the host OS type (RedHat or Suse). On the card file systems the files added are:

/etc/sysconfig/network/ifcfg-mic0

/etc/sysconfig/hostname

/etc/ssh/ssh_host_key

/etc/ssh/ssh_host_key.pub

/etc/ssh/ssh_host_rsa_key

/etc/ssh/ssh_host_rsa_key.pub

/etc/ssh/ssh_host_dsa_key

/etc/ssh/ssh_host_dsa_key.pub

/etc/resolv.conf

/etc/nsswitch.conf

/etc/hosts

All network configuration parameters take effect upon executing **service mpss start**.

4.4.4.1 Host Name Assignment

The **Hostname** parameter defines the value assigned to the host name on the Intel® Xeon Phi™ coprocessor card. The initial value from the **micctrl --initdefaults** command is set to the hostname with a dash and the card name appended to it. The host name string may be edited in the card specific configuration.

4.4.4.2 MAC Address Assignment

Configuring the virtual network interface between Intel® Xeon Phi™ coprocessor cards and the host is a non-trivial process and differs based on the required topology. However, as a prerequisite, both ends of the virtual network need to have **MAC** addresses assigned.

```
MacAddrs Static
MacAddrs Random
MacAddrs <host MAC>:<card MAC>
```

The current driver associated with the Intel® MPSS release creates MAC addresses based on the serial number of the Intel® Xeon Phi™ coprocessor card. If the card is an older version with an invalid serial number, it will randomly assign the MAC address.

System administrators may override the default created by the driver with the **MacAddrs** configuration parameter. The **micctrl --mac** command can be used to set this parameter.

Both the driver and micctrl will create MAC addresses where the lower bit being set indicates the host side of the virtual network connection.

By default, the current driver assigns addresses starting with an IEEE assigned 4C:97:BA, to easily identify the Intel® Xeon Phi™ coprocessor card interfaces.

4.4.4.3 Static Pair (Default) Topology

In the static pair topology, every Intel® Xeon Phi™ coprocessor card is assigned to a separate subnet known only to the host. Each card specific configuration file will contain the **Network** parameter with the format:

```
Network StaticPair <card IP> <host IP> <hosts> \
<netbits> [<mtu>]
```

The IP address for the Intel® Xeon Phi™ coprocessor and host ends of the virtual network connection must be a fully qualified IP address and the first three quads of the addresses must match. The **micctrl --initdefaults** or **micctrl --resetconfig** commands assign a default value to the top two quads of “172.31”. The third quad indicates the card number. The Intel® Xeon Phi™ coprocessor end of the virtual connection is assigned “1” for the last quad and the host end is assigned “254”. For example, the host end of the **mic0** card will have the IP address “172.31.0.254” and the card end will be assigned the IP address “172.31.0.1”.

The **hosts** argument indicates whether the system administrator wants **micctrl** to modify the host’s /etc/hosts file. If it is set to **yes** the file will be modified; **no** indicates to not modify it.

The **netbits** argument specifies the prefix parameter to be used to formulate the NETMASK to assign to the interface. For a static pair configuration it should never be necessary to change this parameter.

Optionally, the **mtu** parameter will specify the packet size to use over the virtual network connection. The default value of 64k has been shown to provide the highest network performance and it should not be necessary to add this parameter.

It is up to the system administrator to correctly route the virtual Ethernet nodes to the external network or each other.

The static pair network configuration can be changed by editing a card’s configuration file **Network** parameter, which is updated when the **micctrl --resetconfig** command is executed. The recommended method of changing the **Network** parameter is to use the **micctrl --network** command (see the **micctrl** section of this document). The **--network** method explicitly knows the previous configuration and can remove it before creating

the new one. In either case, all of the network control files will be created when the operation is done.

4.4.4.4 Internal Bridge Topology

Linux* provides a mechanism for bridging network devices to a common network. The terminology "internal bridge", in the context of Intel® Xeon Phi™ coprocessor configuration, refers to the process of bridging together multiple Intel® Xeon Phi™ coprocessor card virtual network interfaces, on the same host.

Internal bridge configuration is specified by adding a **Bridge** parameter to the default.conf file to specify the bridge information, and then changing the **Network** parameter to point to it.

To create an internal bridge the **type** argument must be set to **Internal**.

The IP address must be set to a fully qualified address.

The **NetBits** parameter is set to 24 by default and will rarely need to be changed.

The Optional **MTU** parameter sets the packet size used over the virtual Ethernet. The driver uses a default size of 64k if this value does not change. Testing has shown this to be the optimal size for the virtual network.

Each Intel® Xeon Phi™ coprocessor's configuration must contain a **Network** entry pointing to the added bridge with the format:

```
Network StaticBridge <bridgename> <IP> <hosts>
```

The **bridgename** argument must match the name of a bridge specified with a **Bridge** parameter.

The IP address must be a fully qualified address.

The **hosts** argument indicates whether the system administrator wants **micctrl** to modify the hosts /etc/hosts file. If it is set to "yes" the file will be modified and "no" indicates to not modify it.

The resulting configuration files will use the bridge configuration for the **MTU** and **NetBits** to ensure they match.

The static pair network configuration can be changed by editing a card's configuration file **Network** parameter updated when the **micctrl --resetconfig** command is executed. The recommended method of changing the **Network** parameter is to use the **micctrl --network** command (see the **micctrl** section of this document). The **--network** method explicitly knows the previous configuration and can remove it before creating the new one. In either case, all the network control files will be created when the operation is done.

4.4.4.5 External Bridge Topology

The Linux* bridging mechanism can also bridge the Intel® Xeon Phi™ coprocessor virtual connections to a physical Ethernet device. In this topology, the virtual network interfaces become configurable to the wider subnet. The Intel® Xeon Phi™ coprocessor

configuration must become aware of the network bridge before virtual network interface can be attached to it.

The **Bridge** configuration parameter with the **type** argument set to **External** creates this mapping. If the corresponding bridge networking configuration file (ex: ifcgf-br0) does not exist then configuring this parameter will cause it to be generated. It will not generate the physical interface file to attach it to the physical network. The system administrator will need to perform this step. For example, on Red Hat, a file to link the **eth0** interface to the bridge would be */etc/sysconfig/network-scripts/ifcfg-eth0*:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

On Suse releases, the physical port name will also need to be added to the **BRIDGE_PORTS** entry in the bridge configuration file.

The rest of the arguments to the **Bridge** configuration parameter are the same as the internal bridge configuration with exception of **MTU** size. The default value is set to 1500 bytes to match default physical network settings.

Like the internal method, the **Network** parameter connects a virtual network interface to the defined bridge.

In both the **Bridge** and **Network** parameters, external bridges may have the IP address set to **dhcp**.

4.4.4.6 Assigning the Netmask

The **NetBits** parameter has been deprecated and is now a part of the **Bridge** and **Network** parameters. Running **micctrl --initdefaults** will convert this parameter to be included in the new ones.

4.4.4.7 Assigning the MTU size

The **MTU** parameter has been deprecated and is now a part of the **Bridge** and **Network** parameters. Running **micctrl --initdefaults** will convert this parameter to be included in the new ones.

4.4.4.8 Host SSH Keys

The secure shell utilities recognize a Linux* system on the network by its “host key files”. These files are found in the */etc/ssh* directory. The **micctrl --initdefaults** command uses the **ssh-keygen** utility to generate key values if they are not found for the card. These values, like the MAC addresses, are considered to be highly persistent, and the **micctrl** command will retain their values if they exist.

In some clusters, detecting and protecting against “man in the middle” and other such attacks might not be required. In this case, the system administrator may use the **micctrl --hostkeys** command to set the host SSH keys to be the same, cluster wide.

4.4.4.9 Name Resolution Configuration

Name resolution on the card is set by creating the */etc/nsswitch* file and copying the */etc/resolve.conf* file from the host to the Intel® Xeon Phi™ coprocessor file system.

5 The micctrl Utility

The **micctrl** utility is a multi-purpose toolbox for the system administrator. It provides these categories of functionality.

- Card state control – boot, shutdown and reset control while the **mpssd** daemon is running.
- Configuration file initialization and propagation of values.
- Helper functions for modifying configuration parameters.
- Helper functions for modifying the root file system directory or associated download image.

The **micctrl** utility requires a first argument specifying the action to perform, followed by option-specific arguments. The arguments may be followed by a list of Intel® Xeon Phi™ coprocessor card names, which is shown in the syntax statements as [mic card list]. If no cards are specified, the **mic.ko** driver must be loaded and the existing card list is probed. Otherwise, the card will be a list of the card names. For example, the list may be "mic1 mic3", if these are the cards to control.

5.1 Card State Control

Starting the **mpssd** daemon may initiate booting of the Intel® Xeon Phi™ coprocessor cards. On a system with multiple cards it would be intrusive to shut down and restart the daemon to reboot one of the cards. It would force all of them to be rebooted. Therefore, the **micctrl** utility provides mechanisms for individual card control.

Micctrl controls Intel® Xeon Phi™ coprocessor card state and queries card state via the **sysfs** entry */sys/class/mic/<micname>/state*. The *micname* value is literally the name of the mic card and will be in the format *mic0*, *mic1*, etc.

Reading from the state will show the current run state of the corresponding Intel® Xeon Phi™ coprocessor card. Writing to the *state* may cause the corresponding card to change states, and is restricted to the root user.

5.1.1 Booting Intel® Xeon Phi™ Coprocessor Cards

Command syntax:

```
micctrl -b [-w] [mic card list]
micctrl --boot [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor card(s) must be in the "ready" state. This command writes the string "boot:linux:<image>" (where image is the OSimage configuration

parameter) to the */sys/class/mic/<micname>/state* sysfs file. The driver will inject the indicated Linux* image into the cards memory and start it booting.

The optional **-w** parameter may be specified to instruct the **micctrl** command to wait until the specified cards have either entered the "online" or "failed" states. The **wait** option will timeout after 300 seconds.

5.1.2 Shutting Down Intel® Xeon Phi™ Coprocessor Cards

Command syntax:

```
micctrl -S [-w] [mic card list]
micctrl --shutdown [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor card must be in the "online" state. This command writes the string "shutdown" to the */sys/class/mic/<micname>/state* sysfs file. The driver instructs the card to perform an orderly shutdown and wait for completion. It will then reset the card to place it again in the boot ready state.

The optional **-w** parameter may be specified to instruct the **micctrl** command to wait until the specified cards have entered the "ready" state. The **wait** option will timeout after 300 seconds.

5.1.3 Rebooting Intel® Xeon Phi™ Coprocessor Cards

Command syntax:

```
micctrl -R [-w] [mic card list]
micctrl --reboot [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor card must be in the "online" state. This command sequentially performs the shutdown and boot functions described in Sections 5.1.1 and 5.1.2.

The optional **-w** parameter may be specified to instruct the **micctrl** command to wait until the specified cards have entered the "ready" state. The **wait** option will timeout after 300 seconds.

5.1.4 Resetting Intel® Xeon Phi™ Coprocessor Cards

Command syntax:

```
micctrl -r [-w] [mic card list]
micctrl --reset [-w] [mic card list]
```

The Intel® Xeon Phi™ coprocessor card can be in any state. This command writes the string "reset" to the */sys/class/mic/<micname>/state* sysfs file. The driver will perform a soft reset on the card by setting the correct card PCI mapped register.

NOTE: Performing a reset may result in the loss of file data that has not been flushed to a remote file. It is therefore recommended to perform a shutdown where possible instead of a reset.

The optional **-w** parameter may be specified to instruct the **micctrl** command to wait until the specified cards have entered the "ready" state. The **wait** option will timeout after 300 seconds.

5.1.5 Waiting for Intel® Xeon Phi™ Coprocessor Card State Change

Command syntax:

```
micctrl -w [mic card list]
micctrl --wait [mic card list]
```

The **wait** option waits for the status of the Intel® Xeon Phi™ coprocessor card to be either "online" or "ready". It also allows for a brief pause to the "ready" state during **mpssd** startup. It is intended for users to verify the **mpssd** startup, shutdown, or reset procedure is complete. It has a built-in timeout value of 300 seconds.

5.1.6 Intel® Xeon Phi™ Coprocessor Card Status

Command syntax:

```
micctrl -s [mic card list]
micctrl --status [mic card list]
```

The **status** option displays the status of the Intel® Xeon Phi™ coprocessor cards in the system. It the status is "online" or "booting" it also displays the name of the associated boot image.

5.2 Configuration Initialization and Propagation

This section discusses the micctrl command options for initializing configuration files, and propagating, resetting, and cleaning configuration parameters.

5.2.1 Initializing the Configuration Files

Command syntax:

```
micctrl --initdefaults [mic card list]
```

The Intel® MPSS installation does not provide configuration files described earlier in Configuration. Instead, these files are created by the **micctrl --initdefaults** command. **micctrl --initdefaults** can be run anytime but will not change files if they already exist and have valid information.

The **--initdefaults** option first checks to see if the */etc/sysconfig/mic/default.conf* file is present. If not, it creates the default version of it. Then, for each supplied card, it checks for the existence of the card-specific configuration file */etc/sysconfig/mic/<micname>.conf*. If it is not present, it creates a default version with an **Include** parameter including the **default.conf** file.

The **--initdefaults** option then proceeds to parse the per card configuration files. For each parameter that is not set, it will add a default value to the per card configuration file. At the same time it will check for deprecated parameters and transform them to the updated parameters. For example: the deprecated **FileSystem** parameter is updated to **RootDevice RamFS**.

The operation associated with each newly added is also performed at configuration time. For example, the addition of the **UserAuthentication Local** parameter triggers the creation of the card's */etc/passwd*, */etc/shadow*, and */etc/group* files, along with any corresponding user directories and ssh key files.

5.2.2 Propagating Changed Configuration Parameters

Command syntax:

```
micctrl --resetconfig [mic card list]
```

Changes to the configuration files are propagated with the **micctrl --resetconfig** command. The **--resetconfig** option first removes the files in **MicDir** created by the configuration process, with the exception of the highly persistent ssh host key files. It then regenerates those files according to the parameters in the */etc/sysconfig/mic/<micname>.conf and /etc/sysconfig/mic/default.conf* files. This process will not add default parameters, but only causes the changed parameters to be propagated.

5.2.3 Resetting Configuration Parameters

Command syntax:

```
micctrl --resetdefaults [mic card list]
```

In the event of a failed or problematic configuration process, the best remedy may be to start again. The **micctrl --resetdefaults** command deletes the configuration files and executes the same process as the **--initdefaults** option.

Since **--initdefaults** only affects the files known to the configuration, it does not delete any files the system administrator has added to a card's file system.

5.2.4 Cleaning Configuration Parameters

Command syntax:

```
micctrl --cleanconfig [mic card list]
```

Since Intel® MPSS configuration commands will replace configuration parameters that have been deprecated, the **--resetdefaults** and **--resetconfig** commands may not restore the deprecated commands of some previous version of Intel® MPSS. Recalling the earlier example in Section 5.2.1, "Initializing the Configuration Files", the **--resetdefaults** and **--resetconfig** commands will not automatically revert "RootDevice RamFS" back to the "FileSystem" parameter. If this is the desired goal, then removing the whole card configuration may be required.

The **--cleanconfig** option not only removes a card's configuration files, but also removes all files in the **MicDir** parameter directory along with the other values specified by **RootDevice**.

5.3 Helper Functions for Configuration Parameters

This section discusses command options for adding and removing users and groups.

5.3.1 Change the Networking Configuration Parameters

5.3.1.1 MAC Address Assignment

Command syntax:

```
micctrl --mac=serial
micctrl --mac=random
micctrl --mac=<MAC address>
```

The **–mac** option allows system administrators to set the value of the **MacAddrs** parameters. If the Intel® Xeon Phi™ coprocessor has a valid serial number, then using the **--mac=serial** option will tell the configuration to use the MAC address created from the card's serial number.

Using the **--mac=random** option will configure the interface to use whatever address the driver assigns.

System administrators may set any valid MAC address in the format XX:XX:XX:XX:XX:XX. This address will be assigned to the Intel® Xeon Phi™ coprocessor end of the virtual network interface, and this number, with the last segment incremented by one, will be assigned to the host end of the virtual network.

5.3.1.2 Static Pair

Command syntax:

```
micctrl --network=static [--ip=<IP>] [--mtu=<mtu>] \
[--hosts=<yes|no>] [mic card list]
```

The **--network** option set to **static** without specifying a bridge name provides an easy method for changing the **Network** configuration parameter to a static pair type. If no IP address is included, then the network interfaces for the listed cards is set to the default values provided by the original **micctrl --initdefaults** command.

If the **IP** address specified is the first two quads, then **micctrl** will finish the address by setting the third quad of each address to the cards ID plus one and the forth to 1 on the card and 254 on the host. For example, on a two card system specifying an **IP** value of "10.10" will create mic0 values of "10.10.1.1" and "10.10.1.254" and mic1 values of "10.10.2.1" and 10.10.2.254".

It is possible to explicitly set the values assigned to the IP addresses. The **IP** argument must be set with the format **cardIP,hostIP:cardIP,hostIP….** Each **cardIP - hostIP** pair specifies a card's values. For example, if there are two cards in the system and the entry is "10.10.10.1,10.10.10.2:10.10.11.1,10.10.11.2" then mic0 will be assigned the card IP address of 10.10.10.1 and host of 10.10.10.2. Mic1 will be assigned 10.10.11.1 and 10.10.11.2.

The default size of MTU is set to max IPV4 size of 64k. This value can be changed by specifying the **MTU** value. This should only be useful for performance testing since testing has shown the default value to be most performant.

Setting the **hosts** value indicates to **micctrl** to whether it should add an entry for the Intel® Xeon Phi™ coprocessor IP address to the host's /etc/hosts file or not. If this value is set to **no** then the system administrator should add it to the /etc/hosts file.

5.3.1.3 Internal Bridging

Command syntax:

```
micctrl --addbridge=<brname> --type=internal \
 --ip=<IP> [--netbits=<bits>] [--mtu=<MTU>]
micctrl --network=static --bridge=<name> [--ip=<IP>] \
 [--hosts=<yes|no>] [mic card list]
```

The internal bridging network topology connects the Intel® Xeon Phi™ coprocessor cards in a system together with one IP address for the host. This is accomplished by first creating the bridge interface on the host and then connecting the virtual network interfaces to it.

Micctrl creates bridge interfaces with the **addbridge** option. The **name** of the bridge must be specified and generally on Linux* is **br0** or **br1** . Setting the **type** to **internal** will cause **micctrl** to always create the correct network configuration files for the host.

Like static pair, you may specify the netbits value and the mtu value but it has no real effect. Each of the virtual Ethernet interfaces added to the bridge will inherit these values from the bridge specification.

Virtual network interfaces are added to the bridge with the **micctrl --network** command. The **bridge** argument must be present and the **IP** argument must specify IP addresses with the first 3 quads matching those of the bridge IP address.

The IP address must be a fully qualified dot notated address. If more than one card is specified, each card will get the same IP address with the cards number added to the fourth quad. For example, if the address 10.10.10.12 is specified, the mic0 will receive 10.10.10.12 and mic1 10.10.10.13.

Setting the **hosts** value indicates to **micctrl** to whether it should add an entry for the Intel® Xeon Phi™ coprocessor IP address to the host's /etc/hosts file or not. If this value is set to **no** then the system administrator should add it to the /etc/hosts file.

5.3.1.4 External Bridging

Command syntax:

```
micctrl --addbridge=<brname> --type=external \
 --ip=<IP> [--netbits=<bits>] [--mtu=<MTU>]
micctrl --network=static --bridge=<name> [--ip=<IP>] \
 [--hosts=<yes|no>] [mic card list]
micctrl --network=dhcp --bridge=<name> \
 [--hosts=<yes|no>] [mic card list]
```

The **external** bridge type definition differs from **internal** only in that **micctrl** is more cautious about creating the hosts network configuration files. If it finds the ifcfg file for the bridge is already present it will not change it. It has assumed the bridge is already connected to a physical Ethernet device. If the ifcfg file does not exist then one is created just like an **internal** bridge file. It is a task of the system administrator to modify the configuration file for the physical Ethernet port to attach to the bridge.

External bridges may also be declared as **dhcp** instead of **static**. During boot, the Intel® Xeon Phi™ coprocessor Linux* OS will attempt to retrieve an IP address from a DHCP server.

5.3.1.5 Modifying Existing Network Definitions

Command syntax:

```
micctrl --modbridge=<brname> --ip=<IP> \
 [--netbits=<bits>] [--mtu=<MTU>]
micctrl --delbridge=<brname>
```

If changes to any of the bridges are required, the **modbridge** option to **micctrl** will allow the change of any combination of **ip, netbits** or **MTU.** In addition any changes to **MTU** or **netbits** will be propagated to any of the virtual network configuration files.

If a bridge is no longer needed, the **delbridge** option will remove it from the Intel® Xeon Phi™ coprocessor configuration. If the bridge is not external, it will also remove the corresponding host network configuration file.

If parameters have been set wrong with **micctrl --network,** restore the network interface back to the static pair default with **micctrl --network=static** and then try again.

5.3.1.6 Extra Networking Notes

On Suse, upon completion of all network change commands, run **service networking restart**. **Micctrl** does not yet understand how to flush the name server cache.

5.3.2 Change the UserAuthentication Configuration Parameter

Command syntax:

```
micctrl --configuser=none [-ids] [mic card list]
micctrl --configuser=local [--low=<low uid>] \
[--high=<high uid] [-ids] [mic card list]
```

The **--configuser** option provides an easy method for changing the **UserAuthentication** configuration parameter. It performs the same process as **--resetconfig** for this single parameter.

When specifying the **local mode**, low and high user ID values may optionally be supplied. The default values are 500 and 65000, for low and high user ID, respectively. Although any 32 bit user ID may be entered, it is not recommended to use less than 500 for the low value. This is the threshold at which most Linux* releases start user ID allocation. Migrating the user IDs from system level accounts to a card is not recommended.

The optional **-i**, **-d**, and **-s** parameters are mutually exclusive; trying to use more than one will result in an error.

The **-d** option indicates to remove all the current users in the cards file system before resetting the user authentication mode. This is the default for the "**None**" value.

The **-s** option indicates to save, or not delete, all the current users before changing the user authentication mode. This is the default for the "**local**" mode.

The **-i** option prompts the user to delete or save each current user before resetting the user authentication mode.

It is legal to specify changing the **UserAuthentication** parameter to the old value. Tying the same value to the **-i** option gives the system administrator the chance to clean up the current user list.

NOTE: Every user to be populated to the Intel® Xeon Phi™ coprocessor file system should set their secure shell configuration files before running this command.

5.3.3 Adding Users to the Intel® Xeon Phi™ Coprocessor File System

Command syntax:

```
micctrl --useradd=<name> --uid=<uid> --gid=<gid> \
[--home=<dir>] [--comment=<string>] [--app=<exec>] \
[--sshkeys=<keydir>] [mic card list]
```

The **--useradd** option adds the specified user name to the */etc/passwd* and */etc/shadow* files on the Intel® Xeon Phi™ coprocessor file system. The system administrator must specify the correct user and group IDs for the user that is being added.

The **--home** argument specifies the user's home directory in the card file system, and will cause the directory to be created. The default home directory for use <name> is */home/<name>.*

The **--comment** argument specifies a comment string to be added to the comment field in */etc/passwd*. The default comment string is the user names.

The **--app** argument specifies the default application executed by the user. The default app is */bin/sh*.

The **--sshkeys** argument specifies the host directory in which the user's secure shell key files are to be found. The default is */home/<name>/.ssh*.

In addition, a default **.profile** file will be added for the user.

5.3.4 Removing Users from the Intel® Xeon Phi™ Coprocessor File System

Command syntax:

```
micctrl --userdel=<name> [mic card list]
```

The **--userdel** option removes the specified user from the card's */etc/passwd* and */etc/shadow* files. It also removes the directory stored in the **home** field of the */etc/passwd* file.

5.3.5 Adding Groups to the Intel® Xeon Phi™ Coprocessor File System

Command syntax:

```
micctrl --groupadd=<name> --gid=<gid> [mic card list]
```

The **--groupadd** option adds the specified group name and ID to the card's */etc/group* file.

5.3.6 Removing Groups from the Intel® Xeon Phi™ Coprocessor File System

Command syntax:

```
micctrl --groupdel=<name> [mic card list]
```

The **--groupdel** option removes the specified group name entry from the card's */etc/group* file.

5.3.7 Changing a User's Password

Command syntax:

```
micctrl --password=<name> … [mic card list]
```

This option, while visible in the **micctrl** utility, has not yet been implemented.

5.3.8 Setting the Root Device

Command syntax:

```
micctrl --rootdev=RamFS --target=<location> \
[mic card list]
micctrl --rootdev=StaticRamFS --target=<location> \
[mic card list]
micctrl --rootdev=NFS --target=<location> \
 --server=<name> -d –c [mic card list]
micctrl --rootdev=SplitNFS --target=<location> \
 --usr=<usr location> --server=<name> -d –c \
 [mic card list]
micctrl --rootdev=InitRD [mic card list]
```

The **--rootdev** option changes the configured **RootDevice** parameter. It defines whether the Intel® Xeon Phi™ coprocessor file root system will be mounted from the initial ram disk, a downloaded ram file system, or a NFS export.

5.3.8.1 Ram Root File System

Specifying a rootdev type of either **RamsFS** or **StaticRamFS** instructs the booting card to mount its root file system by downloading a file specified by the **target** argument. Target is the name of the compressed cpio image to be used for the Intel® Xeon Phi™ coprocessor file system. If it is not specified the default value of */opt/intel/mic/filesystem/micN.*image will be used. The init script will create a ram disk, load the files from **target** into it and do a switch_root to the new file system.

The difference between **RamFs** and **StaticRamFS** is **RamFS** will build **target** each and every time download is requested from the **BaseDir, CommonDir, MicDir** and any

Overlay parameters. The static definition will use **target** as it exists and error if it does not exist.

5.3.8.2 NFS Root File System

Specifying the **NFS** or **SplitNFS** options instructs the booting card to mount its root file system from the NFS export specified by **target** and **server**. If server is not specified, then **micctrl** will use the IP address of the host as the server name. If target is not specified, then the default value of */opt/intel/mic/filesystem/micN.*export will be used. The NFS export should include **rw** and **no_root_squash** in its definition. If the **-d** option is included, the old root device **target** value will be deleted. If **-c** is included, **micctrl** will create it.

SplitNFS differs from **NFS** in that it also creates the correct files for a shared NFS export for the */usr* directory specified by **usr location**. If it is not specified, then it will default to */opt/intel/mic/filesystem/usr.export*. The system administrator must also add this to the exports file.

5.3.8.3 Initial Ram Disk Root File System

The **InitRD** specification instructs the boot card to stop booting in its initial ram disk environment. This is mostly used for debugging and is not recommended.

5.3.9 Adding a NFS Mount

Command syntax:

```
micctrl --addnfs=<NFS export> --dir=<mount dir> \
[--server=<server>] [mic card list]
```

The **--addnfs** option adds an NFS mount entry to the Intel® Xeon Phi™ coprocessor card's */etc/fstab* file. It specifies the NFS export and the mount directory. If the optional server argument is not specified, it places the IP address of the host in the server field.

5.3.10 Removing a NFS Mount

Command syntax:

```
micctrl --remnfs=<mount dir> [mic card list]
```

The **--remnfs** option searches the */etc/fstab* for the Intel® Xeon Phi™ coprocessor card for the specified mount point and removes the mount point from the file.

5.3.11 Specifying the Host Secure Shell Keys

Command syntax:

```
micctrl --hostkeys=<keydir> [mic card list]
```

The **--hostkeys** option removes the host keys randomly generated by the **--initdefaults** command and replaces it with the files from the specified directory. Since these files are considered to be highly persistent they should stay resident unless the **--resetdefaults** or **--cleanconfig** option is performed.

5.3.12 Setting Startup Script Parameters

Command syntax:

```
micctrl --service
micctrl --service=<name> --state=<on|off> \
 [--start=<num>I [--stop=<num]] [mic card list]
```

The Intel® Xeon Phi™ coprocessor Linux* OS, like any Linux OS, executes a series of scripts on boot, which are located in /etc/init.d. To determine which of the installed scripts are executed on any boot, links to these scripts are created in run level directory. The card's OS runs at level 3, defining the run level directory to be */etc/rc3.d*.

On most Linux* systems, the service scripts to be executed are enabled or disabled using the **chkconfig** command. On the MPSS stack this is performed by the **micctrl --service** command.

The **state** option must be set to **on** or **off** and determines whether the script will execute on boot. Services already included in the configuration may have their state changed without specifying new **start** or **stop** values.

The **start** and **stop** parameters must be between 1 and 100, and determine the order in which the services are executed. If **stop** is not specified, then it will be set to **100 – start**.

Add on software containing a service script will include the **Service** parameter associated with it. Modifying the default value included in its own configuration file will cause an overriding entry to be set in the **micN.conf** file.

Micctrl --service may be called with no arguments and will display a list of current service settings. A typical fresh install will display:

```
    Service       Start Stop  State
      ------------------------------------
mic0: coi         95    5     on
      fileperms   1     99    on
      network     21    79    on
      sshd        80    20    on
      pm          90    10    on
      blcr        95    5     off
      mictune     99    1     on
mic1: coi         95    5     on
      fileperms   1     99    on
      network     21    79    on
```

```
sshd        80    20    on
pm          90    10    on
blcr        95    5     off
mictune     99    1     on
```

5.3.13 Overlaying File System with More Files

Command syntax:

```
micctrl --overlay
micctrl --overlay=<type> --state=<on|off|delete> \
 --source=<dir>I --target=<target> [mic card list]
```

The **Overlay** parameter specifies a block of files to be included in the Intel® Xeon Phi™ coprocessor file system. The **type** parameter specfies two different methods of including files known with the values **filelist** or **simple**.

If the type is set to **filelist,** then files from the **source** directory will be placed on the card's file system, based on entries in a file specified by the **target** parameter. The **filelist** format allows for copying random, or disassociated files to the card. It also provides a mechanism for creating special file types and setting absolute privileges. For more information on its format, check Section 6.1.

If the type is set to **simple** then the files from the **source** directory are copied directly to the card's file system under the **target** directory.

The **state** argument determines if the files are to be copied or not. Additional software will be added by creating a configuraton file in the */etc/sysconfig/mic/conf.d* directory with an **Overlay** parameter included. Because **micctrl** cannot modify these files, changing the default value of **state** will be done by duplicate entries in the **micN.conf** files. If **state** is on, the files will be copied to the card's file system; if **state** is off, the files will not be copied. Calling **micctrl --overlay** with state set to **delete** will cause the entry to be removed from the **micN.conf** file.

Micctrl --overlay may be called with no arguments and will display the current overlay status. An unchanged system will have an overlay for **coi** and its output will be:

```
mic0: Filelist /opt/intel/mic/coi /opt/intel/mic/coi/config/coi.filelist on
mic1: Filelist /opt/intel/mic/coi /opt/intel/mic/coi/config/coi.filelist on
```

5.3.14 Base Files Location

Command syntax:

```
micctrl --basedir
micctrl --basedir=<basedir> [--list=<filelist>] \
 [mic card list]
```

The **basedir** argument modifies the **BaseDir** parameter in the configuration files. This parameter points to the base set of files installed by the Intel® Xeon Phi™ coprocessor

software. The **BaseDir** parameter is normally in the **default.conf** file. Changing it, by specifying the **basedir** argument, will add an entry to the **micN.conf** files and override the default value.

If the **basedir** directory does not exist then the location of the old **BaseDir** parameter will be copied to the new location. The **filelist** file will also be copied. The original files will not be deleted.

The optional **list** argument defines the location of the **filelist** file. If it is not specified then it will be assigned the value **<basedir>.filelist**.

If **basedir** is not specified then a list of the cards and their **BaseDir, CommonDir** and **MicDir** parameters will be displayed.

5.3.15 Common Files Location

Command syntax:

```
micctrl --commondir
micctrl --commondir=<commondir> [--list=<filelist>] \
 [mic card list]
```

The **commondir** argument modifies the **CommonDir** parameter in the configuration files. This parameter points to a directory of files to be downloaded to all Intel® Xeon Phi™ coprocessor cards in a host system. The **micctrl --initdefaults** option creates an empty common directory if it does not exist. The **CommonDir** parameter is normally in the **default.conf** file. Changing it, by specifying the **commondir** argument, will add an entry to the **micN.conf** files and override the default value.

If the **commondir** directory does not exist then the location of the old **CommonDir** parameter will be copied to the new location. The **filelist** file will also be copied. After the files have been copied, the configurations for all known cards in the system are checked for references to the old **CommonDir** values and if no references exist the files will be deleted.

The optional **list** argument defines the location of the **filelist** file. If it is not specified then it will be assigned the value **<commondir>.filelist**.

If **commondir** is not specified then a list of the cards and their **BaseDir, CommonDir** and **MicDir** parameters will be displayed.

5.3.16 Coprocessor Unique Files Location

Command syntax:

```
micctrl --micdir
micctrl --micdir=<micdir> [--list=<filelist>] \
 [mic card list]
```

The **micdir** argument modifies the **MicDir** parameter in the configuration files. This parameter points to a directory of files to be downloaded to all Intel® Xeon Phi™ coprocessor cards in a host system, containing files unique to each card instance. The **micctrl --initdefaults** option creates many default files here. Other **micctrl** commands modify many of these files.

If the **micdir** directory does not exist then the location of the old **MicDir** parameter will be copied to the new location. The **filelist** file will also be copied. The files in the old **MicDir** parameter will be deleted.

The optional **list** argument defines the location of the **filelist** file. If it is not specified then it will be assigned the value **<micdir>.filelist**.

If **micdir** is not specified then a list of the cards and their **BaseDir, CommonDir** and **MicDir** parameters will be displayed.

5.3.17 Coprocessor Linux Image Location

Command syntax:

```
micctrl --osimage
micctrl --osimage=<osimage> [mic card list]
```

The **osimage** option sets the location of the **OSimage** parameter. The **OSimage** defines the Linux* operating system image to boot. The default value of this location is */lib/firmware/mic/uos.img* and is installed by the Intel® MPSS software.

5.3.18 Boot On Intel® MPSS Service Start

Command syntax:

```
micctrl --autoboot
micctrl --autoboot=<yes \ no> [mic card list]
```

The **autoboot** argument sets the **BootOnStart** configuration parameter. This parameter controls whether or not a coprocessor boots when the Intel® MPSS service is started. A **yes** value will indicate to boot the card. If no value is specified then a list of the cards and the current values of **BootOnStart** will be displayed.

5.4 Other File System Helper Functions

5.4.1 Updating the Compressed CPIO Image

Command syntax:

```
micctrl --updateramfs [mic card list]
```

The **StaticRamFS** root file system image is only changed when the system administrator requests it. In many cluster systems this image will be built externally and put in place.

The **--updateramfs** option updates the image from the same parameters used by the **RamFS** specification. The new image will be used the next time the card boots.

5.4.2 Updating the NFS Root Export

Command syntax:

```
micctrl --updatenfs [mic card list]
micctrl --updateusr [mic card list]
```

When new software is added through updating the MPSS stack or add-on products, those changes must be propagated, or installed into the NFS root file system exports. The **micctrl --updatenfs** command performs this process.

If the root device type has been set to **SplitNFS** it may also change files located in the common /usr directory. The **--updateusr** command line option will complete this process.

6 Adding Software

No installation is static. Additional software and products need to be added. The system administrator must therefore add files to the downloaded root file system to meet user needs.

6.1 The File System Creation Process

As previously described in Section 4.3.7, the **RootDevice** parameter defines the type of root device to mount. When the **type** argument to **RootDevice** is **RamFS** or **StaticRamFS**, a file system image is pushed to the card as its ram file system root file system. In this section we describe the process of building such a root file system.

The process is driven by the configuration parameters **BaseDir, CommonDir, MicDir** and **Overlay**. The **descriptor file** argument to each of these configuration parameters identifies a file whose contents are interpreted as a **filelist:** a list of the files to be placed into the file system image. The **location** argument of each of these configuration parameters identifies a directory subtree that is the source of the files to be so placed. The **filelists** are processed in the order **BaseDir, CommonDir, MicDir** and **Overlay**.

There are six **filelist** directive types:

```
dir <name> <perms> <uid> <gid>
file <name> <source> <perms> <uid> <gid>
slink <name> <to> <perms> <uid> <gid>
nod <name> <perms> <uid> <gid> <type> <major> <minor>
pipe <name> <perms> <uid> <gid>
sock <name> <perms> <uid> <gid>
```

Each directive type is specific to one of six types of files available on a Linux* file system.

6.1.1 The dir Filelist Directive

The **dir** directive specifies a directory with name "**name**" is to be created in the card file system. The **perms**, **uid**, and **gid** arguments specify the file's permissions, user ID and group ID respectively. A typical entry is:

```
dir /tmp 0777 0 0
```

The example defines the directory */tmp* to be owned by user **root** and group **root**, and with global permissions for everybody.

6.1.2 The file Filelist Directive

The **file** directive specifies to create the file with **name** is to be created in the card file system image. The **perms**, **uid**, and **gid** arguments specify the file's permissions, user ID and group ID respectively.

The **source** argument to **file** is concatenated to the **location** argument of the **BaseDir, CommonDir, MicDir** or **Overlay** configuration parameter to identify a file whose contents are copied to file **name** in the card file system image.

For example, given the **MicDir** configuration parameter:

```
MicDir /opt/intel/mic/filesystem/mic0 \
/opt/intel/mic/filesystem/mic0.filelist
```

and the following **filelist** directive in /opt/intel/mic/filesystem/mic0.filelist:

```
file /etc/passwd etc/passwd 644 0 0
```

The file */etc/passwd* will be added to the card file image and populated with the contents of the file */opt/intel/mic/filesystem/mic0/etc/passwd*. It will be owned by user root and group root, and with global read permission and root modification permission.

6.1.3 The slink Filelist Directive

The **slink** directive specifies that a symbolic link with **name** is to be created in the card file system image, and linked to **source**. The **perms**, **uid**, and **gid** arguments specify the symbolic link's permissions, user ID and group ID respectively.

A typical use of symbolic links is found in the Linux* startup scripts. The filelist associated with the **MicDir** configuration parameter includes the following:

```
slink /etc/rc3.d/S80sshd ../init.d/sshd 0755 0 0
```

This directs the creation of a symbolic link on the cards file system accessing the */etc/init.d/sshd* file when */etc/rc.d/S80sshd* is accessed.

6.1.4 The nod Filelist Directive

The **nod** directive specifies that a device node with **name** and of **type** is to be created in the card file system image. **type** must be either the character 'b' for block device or 'c' for character device. The arguments **major** and **minor** must be integer values defining the correct values of the node. The **perms**, **uid**, and **gid** arguments specify the device node's permissions, user ID and group ID respectively.

Most device nodes are created dynamically by a device driver. However, some legacy devices still require a hard coded entry. For example the **filelist** for **BaseDir** includes the following entry, which specifies the creation of a device node for the console:

```
nod /dev/console 0600 0 0 c 5 1
```

6.1.5 The pipe Filelist Directive

The **pipe** directive specifies that a named pipe device file with **name** is to be created in the card file system image. The **perms**, **uid**, and **gid** arguments specify the pipe's permissions, user ID, and group ID, respectively.

6.1.6 The sock Filelist Directive

The **sock** directive specifies that a socket device file with **name** is to be created in the card file system image. The **perms**, **uid**, and **gid** arguments specify the socket's permissions, user ID, and group ID, respectively.

6.2 Creating the Download Image File

The download image file for the **RamFS** root device type is created by processing the configuration directives **BaseDir, CommonDir, MicDir** and any **Overlays**, in that order. As the configuration directives are processed, a tree of filenames and their information is built.

NOTE: In a future release, any repeated entry will "overlay" or replace the previous declaration. This is not yet implemented.

When the tree is completely processed, **mpssd** or **micctrl --updateramfs** will create a cpio entry for the file and append it to the filename specified by the **RootDevice** directive. When processing is complete it then compresses the file.

6.3 Adding Files to the Root File System

Adding a file to the root file system can be done in two ways. The system administrator can add an entry to some existing **filelist,** indicating the location of the file. Alternatively, the system administrator can add new **Overlay** configuration parameter with **location** and **descriptor file** arguments that describe the files to be added.

6.3.1 Adding Files by Copying

When adding a file to an existing **filelist**, the first decision is whether the file should be accessible by all the cards or only a particular one. If it is required for all cards to have access, then copy the file to a location under the directory specified by the **location** argument to the **CommonDir** configuration parameter, and amend its **filelist**. Otherwise, copy the file to a location under the directory specified by the **location** argument to the **MicDir** directory, and amend its **filelist**.

If a directory had to be created for the added file, do not forget to insert the appropriate **dir** entry prior to the new **file** entry.

6.3.2 Adding an Overlay

The process for adding an **Overlay** set is similar to the previous section. The file must be placed in the correct location specified under **Overlay** and added to the **filelist** file specified. The power of the **Overlay** is that it may be called from a new configuration file in */etc/sysconfig/mic/conf.d*. Refer to the next section for an example.

6.3.3 Example: Adding a New Global File Set

The Intel® Coprocessor Offload Infrastructure (COI) component of the Intel® MPSS is a good example of how to add a set of software to the Intel® Xeon Phi™ coprocessor file systems. The Intel® COI configuration file is installed as */etc/sysconfig/mic/conf.d/coi.conf*. It contains the **Overlay** configuration parameter:

```
# COI download files

Overlay Filelist /opt/intel/mic/coi /opt/intel/mic/coi/config/coi.filelist on
```

The **location** argument to **Overlay** is */opt/intel/mic/coi*, the directory under which Intel® COI files are installed. The **descriptor file** argument, */opt/intel/mic/coi/config/coi.filelist*, identifies a **filelist** that is installed as part of Intel® MPSS, and which describes the files to include in the Intel® Xeon Phi™ coprocessor file systems in support of Intel® COI. *coi.filelist* has the following contents:

```
dir /bin 755 0 0

file /bin/coi_daemon device-linux-release/bin/coi_daemon 755 0 0

dir /etc 755 0 0

dir /etc/init.d 755 0 0

file /etc/init.d/coi config/coi 775 0 0

dir /lib64 755 0 0

file /lib64/libcoi_device.so device-linux-release/lib/libcoi_device.so 755 0 0

slink /lib64/libcoi_device.so.0 libcoi_device.so 777 0 0
```

From this we can see:

- COI requires the */bin* directory to exist.
- The COI daemon is at */bin/coi_daemon.*
  - It is found on the host file system at */opt/intel/mic/coi/device-linux-release/bin/coi_daemon.*
  - It requires read/write/execute permissions for root.
  - It requires read/execute permissions for other.
  - It runs under the root (0) user and group IDs.
- COI requires the */etc* and */etc/init.d* directories.
- The COI startup script is to be copied to */etc/init.d/coi.*

  - It can be found on the host at */opt/intel/mic/coi/config/coi.*
  - Set permissions, user ID, and group ID as described.
- COI requires the */lib64* directory to exist.
- Install **libcoi_device.so** library.
- Create a symbolic link to **libcoi_device.so**.

Since Intel® COI installs a new daemon on the file system, it needs a startup script for it. The COI startup script appears in the following code example.

```
#!/bin/sh


coiexec=/bin/coi_daemon


case "$1" in
 start)
      if [ ! -f $coiexec ]; then
        exit 1;
      fi

      $coiexec &
      ;;
esac
```

7 Linux SYSFS Entries

The driver supplies configuration and control information to host software through the Linux* Sysfs file system. The driver presents two sets of information:

- Driver global information is presented in the */sys/class/mic/ctrl* directory.
- Information unique to an Intel® Xeon Phi™ coprocessor instance is presented in the */sys/class/mic/micN* directories, where N is an integer number (0, 1, 2, 3, etc.) that identifies the card instance.

7.1 The Global Mic.ko Driver SYSFS Entries

7.1.1 Revision information

Sysfs Entries:

/sys/class/mic/ctrl/host_revision

/sys/class/mic/ctrl/uos_revision

/sys/class/mic/ctrl/version

These entries are read-only. The **host_revision** and **uos_revision** entries are added by the build process and contain the source code tags for the host driver and embedded Linux* OS for the card. These values ensure the absolute identification of the source code producing the resulting software.

The **version** sysfs entry displays a string containing the ID of the build producing the current installed software.

7.1.2 Other Global SYSFS Entries

Sysfs Entries:

/sys/class/mic/ctrl/peer2peer

/sys/class/mic/ctrl/vnet

The **peer2peer** entry contains the state of the Intel® Symmetric Communications Interface (SCIF) peer-to-peer transfer code. It will be either set to **enable** or **disable.**

The vnet entry will contain the number of active links to the virtual Ethernet.

7.2 The Intel® Xeon Phi™ Mic.ko Driver SYSFS Entries

7.2.1 Hardware Information

Sysfs Entries:

/sys/class/mic/micN/family

/sys/class/mic/micN/sku

/sys/class/mic/micN/stepping

/sys/class/mic/micN/active_cores

/sys/class/mic/micN/memsize

These sysfs entries are all read-only.

The **family** sysfs node reports the family the Intel® Xeon Phi™ coprocessor belongs to. At this time the family should always report the string "**Knights Corner**".

The **sku** sysfs node returns a string defining the device type. As an example it may report the string "C0-3120/3120A".

The **stepping** node returns the processor stepping. Typically it will be **B0, B1, C0,** etc.

The **active_cores** node reports the actual number of working cores on the card.

The **memsize** node returns the size of memory on the Intel® Xeon Phi™ coprocessor card.

7.2.2 State SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/state

/sys/class/mic/micN/mode

/sys/class/mic/micN/image

/sys/class/mic/micN/cmdline

/sys/class/mic/micN/kernel_cmdline

The **state** and **cmdline** sysfs nodes are read and write. The others are read-only.

The **state** sysfs node will show one of the following values:

- **ready** card is ready for a boot command
- **booting** card is currently booting
- **no response** card is not responding
- **boot failed** card failed to boot

- **online** card is currently booted
- **shutdown** card is currently shutting down
- **lost** booted card is not responding
- **resetting** card is processing soft reset
- **reset failed** card cannot be reset – non recoverable

Additionally, if the state is **booting, online** or **shutdown**, the state is modified by the information from the **mode** and **image** sysfs nodes. The **mode** will be either **linux** or **elf.** The image file will contain the name of the file used to boot into the associated mode.

Writing to the **state** sysfs node requests the driver to initiate a change in state. The allowable requests are to boot, reset or shutdown the Intel® Xeon Phi™ coprocessor card.

To boot a card, the string to write has the format "boot:linux:<image name>". The **mpssd** daemon uses its **OSimage** parameter to fill in the image name. For example the default Linux* image for the Intel®Xeon Phi™ coprocessor will create the string "boot:linux:/lib/firmware/mic/uos.img". After a successful boot the **state** will be **online**, **mode** will be **linux,** and **image** will be **/lib/firmware/mic/uos.img**.

The **cmdline** parameter is set by user software, normally **mpssd** or **micctrl**, to pass kernel command line parameters to the Intel® Xeon Phi™ coprocessor Linux* boot process. Current parameters include root file system, console device information, power management options and verbose parameters. When the **state** sysfs node requests the card to boot, the driver adds other kernel command line information to the string and records the complete string that was passed to the booting embedded Linux* OS in the **kernel_cmdline** sysfs node.

7.2.3 Statistics

Sysfs Entries:

/sys/class/mic/micN/boot_count

/sys/class/mic/micN/crash_count

These entries are read-only. **The boot_count** sysfs node returns the number of times that the Intel® Xeon Phi™ coprocessor card has booted to the online state. The **crash_count** sysfs node records the number of times that the card has crashed.

7.2.4 Debug SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/platform

/sys/class/mic/micN/post_code

/sys/class/mic/micN/scif_status

/sys/class/mic/micN/log_buf_addr

/sys/class/mic/micN/log_buf_len

/sys/class/mic/micN/virtblk_file

The **platform, post_code,** and **scif_status** entries are read-only; the log_buf_addr, log_buf_len, and virtblk_file entries are read and write.

The **platform** sysfs node should always return a zero value.

The **post_code** sysfs node returns the contents of the hardware register containing the state of the boot loader code. Reading it always returns two ASCII characters. Possible values of note are the strings "12", "FF" and any starting with the character '3'. A string of "12" indicates the Intel® Xeon Phi™ coprocessor is in the ready state and waiting for a command to start executing. A string of "FF" indicates the coprocessor is executing code. A string starting with the character 'C' indicates the coprocessor is in the process of training memory. Any other value should be transitory. Any other value remaining for any length of time indicates an error and should be reported to Intel.

The **log_buf_addr** and **log_buf_len** parameters inform the driver of the memory address in the Intel® Xeon Phi™ coprocessor memory to read its Linux* kernel log buffer. The correct values to set are found by looking for the strings "log_buf_addr" and log_buf_len" in the Linux* system map file associated with the file in the **OSimage** parameter, and are typically set by the **mpssd** daemon.

The **virtblk_file** sysfs node indicates the file assigned to the virtio block interface.

7.2.5 Flash SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/flashversion

/sys/class/mic/micN/flash_update

/sys/class/mic/micN/fail_safe_offset

These nodes are all read-only. The **flashversion** sysfs node returns the current version of the flash image installed on the card by the **micflash** utility. The other two are used by the **micflash** command.

7.2.6 Power Management SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/dpc3_enabled

The **dpc3_enabled** node reports the current setting of the **dpc3** power management setting. If power management is causing errors, writing a "0" to this setting will disable power management.

7.2.7 Other SYSFS Entries

Sysfs Entries:

/sys/class/mic/micN/extended_family

/sys/class/mic/micN/extended_model

/sys/class/mic/micN/fuse_config_rev

/sys/class/mic/micN/meminfo

/sys/class/mic/micN/memoryfrequency

/sys/class/mic/micN/memoryvoltage

/sys/class/mic/micN/model

/sys/class/mic/micN/stepping

/sys/class/mic/micN/stepping_data

These sysfs nodes are all read-only and return the contents of a particular hardware register. They are used by the **micinfo** command.

8 Configuration Examples

8.1 Network Configuration

Linux* provides functionality for creating a new network interface for physical interfaces to slave to. Network packets received on any of the slaves are passed unchanged to the bridge. The bridge is assigned the IP address associated with the system it exists on.

Network packets arriving on any of the physical interfaces are passed to the bridge. If the destination for the packet is the IP address assigned to the bridge, it is passed to the TCP/IP stack on the system. If it is any other destination, the bridge performs the role of a network switch and passes it to the correct physical interface for retransmit.

The Intel® MPSS software uses the bridging functionality to place the virtual network for each card on the same subnet allowing standard networking software to function seamlessly.

8.1.1 Internal Bridge Example

Internal bridging is a term created to describe a networking topology with Intel® Xeon Phi™ coprocessor cards connected through a bridged configuration. The advantage of the internal bridge over the default static pair network configuration is the ability for the cards to communicate with each other as well as the host.

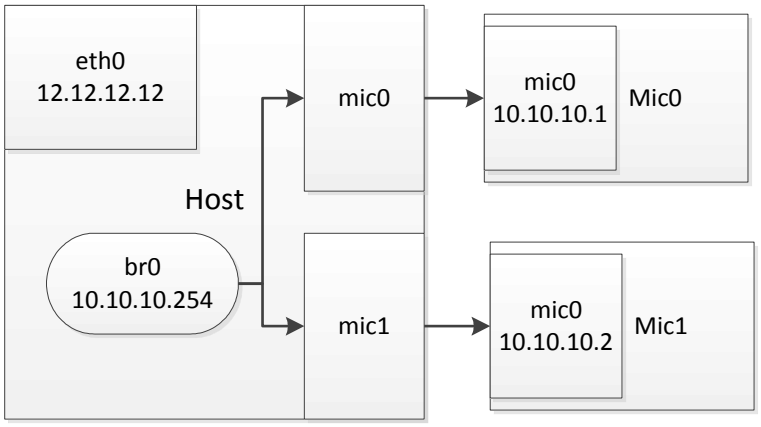

Figure 2 Internal bridge network

If the cards must communicate with the broader physical network, the bridge must be routed to physical Ethernet.

The diagram illustrates the internal bridged network topology. In this example the host and the cards can all communicate through the 10.10.10 subnet. The host can communicate outside through the 12.12.12 subnet but the cards cannot. The series of commands to create this topology would be:

```
user_prompt> sudo micctrl --addbridge=br0 \
--type=internal --ip=10.10.10.254
user_prompt> sudo micctrl --network=static --bridge=br0 \
--ip=10.10.10.1
```

The **micctrl --addbridge** command performs a series of steps starting with removing the old network configuration. Then, if the host does not have a configuration for the designated bridge in */etc/sysconfig/network-scripts/ifcgf-br0,* it creates it containing:

```
DEVICE=br0
TYPE=Bridge
ONBOOT=yes
DELAY=0
NM_CONTROLLED="no"
BOOTPROTO=static
IPADDR=10.10.10.254
NETMASK=255.255.255.0
```

The **micctrl** utility then executes an “ifup br0” command to enable the new bridge interface. After the bridge is enabled the */etc/sysconfig/mic/default.conf* file has the additional line:

```
Bridge br0 Internal 10.10.10.254 24
```

The value of 24 at the end of the line is the default **NetBits** or **PREFIX** value of 24 defining a netmask of FFFFFF00. If the **--mtu** option had been used on the **micctrl** command line then it would have followed the 24.

The **micctrl --network** command slaves the host ends of the virtual network, connects to the designated bridge **br0**, and replaces the network configuration files for the Intel® Xeon Phi™ coprocessor cards with a configuration for the new IP addresses. The first step of this process is to shut down the current virtual network connections with the **ifdown** command, then create new configuration files. The */etc/sysconfig/network-scripts/ifcfg-mic0* becomes:

```
DEVICE=mic0
ONBOOT=yes
NM_CONTROLLED="no"
BRIDGE=br0
```

The */etc/sysconfig/network-scripts/ifcfg-mic1* file contains the same information with the exception of **DEVICE** being set to **mic1** instead of **mic0**. If the associated bridge configuration had set an MTU value other than the default, this file would also contain an MTU value assignment.

When this is complete **micctrl** executes **ifup** commands on both **mic0** and **mic1**. At the end of this process, the **brctl show** command can be used to check the status of the bridge. Its output should be:

```
bridge name     bridge id               STP enabled     interfaces
br1             8000.66a8476a8f15       no              mic0
                                                        mic1
```

The **ifconfig** command relevant output should be:

```
br0     Link encap:Ethernet
        inet addr:10.10.10.254  Bcast:10.10.10.255  Mask:255.255.255.0

mic0    Link encap:Ethernet

mic1    Link encap:Ethernet
```

These commands show the mic0 and mic1 virtual network interfaces are slaved to bridge br0. Bridge br0 has been assigned the host IP address and the slaves do not have their own IP addresses.

The old Network configuration parameters in each card's configuration file are then replaced with the new line. For example the */etc/sysconfig/mic/mic0.conf* file now has the Network configuration line:

```
Network StaticBridge br1 10.10.10.1 yes
```

The */etc/sysconfig/mic/mic1.conf* file will have the same line with the exception of the IP address being assigned the value 10.10.10.2.

Micctrl then creates the network configuration files for the Intel® Xeon Phi™ coprocessor file system. It will first create the network interface configuration file */opt/intel/mic/filesystem/mic0/etc/sysconfig/network/ifcfg-mic0* with the contents:

```
IPADDR=10.10.10.1
HWADDR=06:05:04:03:02:01
PREFIX=24
```

The */opt/intel/mic/filesystem/mic0/etc/sysconfig/network/ifcfg-mic1* file is the same with correct IPADDR and NWADDR values.

Next it will create the */opt/intel/mic/filesystem/mic0/etc/hosts* with the content similar to:

```
127.0.0.1       zappa-mic0.music.local mic0 localhost.localdomain localhost
::1             zappa-mic0.music.local mic0 localhost.localdomain localhost
10.10.10.254    host zappa.music.local
10.10.10.2      mic1 zappa-mic1.music.local mic1
```

NOTE: The entries contained in the configuration file Hostname parameter are related to the IP addresses of other reachable network interfaces, along with the shortened form host, mic0, mic1, etc.

The last of the process is to add entries to the */etc/hosts* file on the host. This last process could have been avoided if the **micctrl --network** command had included the option **--hosts=no**.

The next boot of the Intel® Xeon Phi™ coprocessor cards, by either **service mpss start** or **micctrl -b** will use the new network configuration and the cards will be able to ssh to each other.

8.1.2 Basic External Bridge Example

External bridging is a term used in the Intel® MPSS software to describe a network topology where the virtual network interfaces are bridged to a physical network interface. This will be the desired configuration in clusters. The topology diagram shows individual hosts and their Intel® Xeon Phi™ coprocessor cards become a part of the larger subnet.

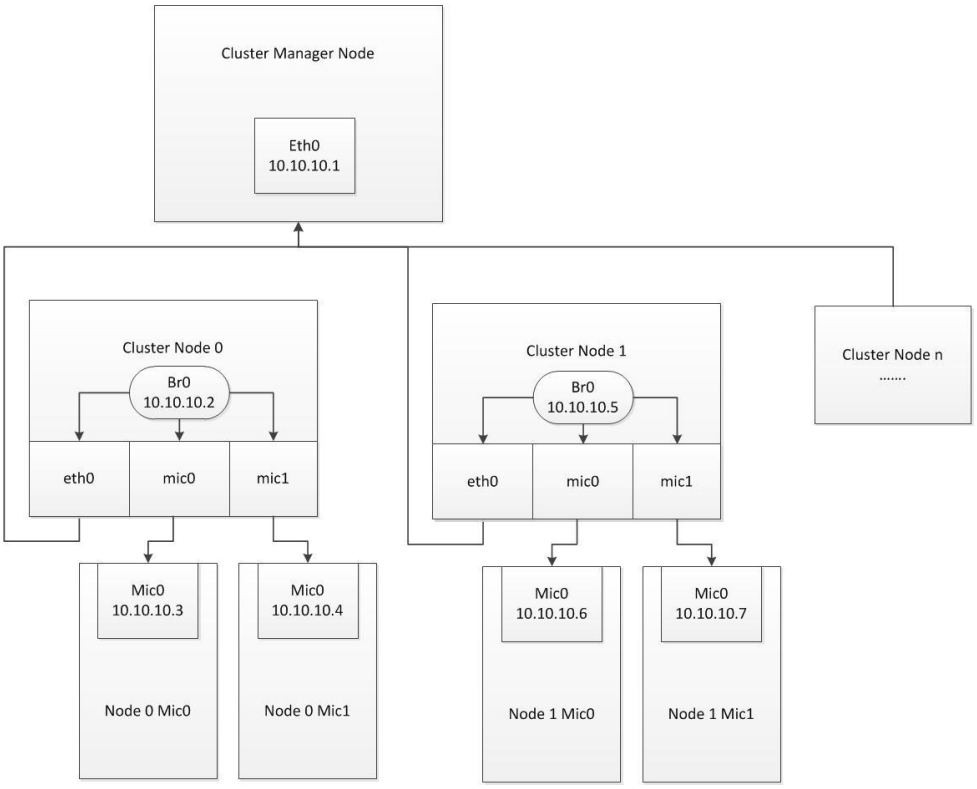

Figure 3 External bridge network

This discussion will illustrate the additional steps required above an internal bridge topology.

On the host node0 the commands to configure the virtual network interfaces are:

```
user_prompt> sudo micctrl --addbridge=br0 \
--type=external --ip=10.10.10.2

user_prompt> sudo micctrl --network=static --bridge=br0 \
--ip=10.10.10.3
```

And on the host node1 the commands are:

```
user_prompt> sudo micctrl --addbridge=br0 \
--type=external --ip=10.10.10.5
user_prompt> sudo micctrl --network=static --bridge=br0 \
--ip=10.10.10.6
```

Normally, the system administrator will have configured the bridge br0 and tied the eth0 interface to it before these configuration calls. If they did not, then **micctrl** will create the */etc/sysconfig/network-scripts/ifcfg-br0*. It will not, however, create the configuration file for the eth0 physical network interface; the system administrator will need to do this step. An example of */etc/sysconfig/network-scripts/ifcfg-eth0* file is:

```
DEVICE=eth0
NM_CONTROLLED=no
TYPE=Ethernet
ONBOOT=yes
BRIDGE=br0
```

When this is completed, perform a **service network restart**.

In the process of configuring an external bridge, **micctrl** assumes that the network packet MTU size must be set to the Ethernet standard 1500 bytes. This value significantly slows down data transfer across the virtual Ethernet interfaces. If the physical network hardware allows, it would be better to increase the MTU size to the biggest possible value. Typically most hardware will support at least 9000 byte mtu sizes.

This could have been set by the original **micctrl --addbridge** command with the addition of the **--mtu=9000** argument. If this was not done and the system administrator wishes to increase the value, this may be done with the command:

```
user_prompt> sudo micctrl --modbridge --mtu=9000
```