OpenCPI
Installation Guide

DRAFT

Revision History

Revision	Description of Change	Date
1.0	Creation	2014-06-23
1.01	Add all FPGA and embedded system (Zed) content	2014-07-07

Table of Contents

1 References

This document assumes a basic understanding of the Linux command line (or "shell") environment. It does not require a working knowledge of OpenCPI, although if anything goes wrong with the installation, more experience with OpenCPI may be required. The reference below is an overview of OpenCPI and may prove useful.

Title	Published By	Link
OpenCPI Technical Summary	OpenCPI	Public URL: https://github.com/opencpi/opencpi/raw/master/doc/OpenCPI_Technical_Summary.pdf

2 Overview

This document describes how to install OpenCPI. The installation is in layers that are mostly optional depending on which platforms are being used. The core installation is the development "host" that allows both for local software-based execution of OpenCPI applications and components, as well as cross-building for other platforms.

The default installation platform for OpenCPI development hosts is (as of June 2014) CentOS6 Linux x86_64 (64-bit). Other Linux variants and 32-bit systems have been used successfully, but CentOS6 64-bit is the default, tested, installation for a development host. Development hosts can either be actual physical systems or virtual machine installations.

Additional installation options are for other target processors and technologies such as the Xilinx Zynq SoC (with ARM processor cores and FPGA resources), or various FPGAs. Preference is given to *cross-building* for embedded CPUs when possible, rather than self-hosting development tools on such platforms, in order to limit the complexities of installing tools on different development hosts.

The document is divided into sections for different classes of target platforms:

- Development hosts (virtual and physical)
- Embedded CPUs
- FPGA platforms.
- GPU platforms

In each section, when appropriate, the physical/electrical installation issues will be addressed as well as issues for installing any required tools on the development host.

A final section lists specific reference platforms that are commonly used and frequently tested.

In several sections is noted that certain features are "roadmap items". This means that the need for the feature is clear and recognized, and it is on the list of features to be implemented in some future release.

3 Installing OpenCPI on Development Hosts

The basic installation of (Linux) development host systems enables the development and execution of OpenCPI components and applications on the development host itself (called *native* execution). It also establishes the host platform for the tools that enable development for other target platforms, including embedded CPUs, FPGAs, and GPUs.

The installation process takes these steps, in three phases:

Phase 1: Hardware and OS installation

1. Installing the hardware or creating a bare virtual machine.
2. Installing and configuring the operating system from CD or CD image file.
3. Updating the operating system to the latest patch level, and enable “sudo”.

    At this this point we have a basic up-to-date OS installation. There are many ways to get here, but we outline a basic approach that works on CentOS6.

Phase 2: Download OpenCPI sources, and install prerequisite software packages.

4. Install the “git” software package, and use it to download the OpenCPI source distribution.
5. Install standard prerequisite packages using the package update and installation tools of the operating system distribution
6. Configure the (non-default aspects of) the OpenCPI environment.
7. Install and build some prerequisite packages that need special OpenCPI-supplied installation scripts.

    At this point we have installed all prerequisites and have a clone/copy of the OpenCPI code tree on our OS installation.

Phase 3: Build and test OpenCPI

8. Build OpenCPI's core tools, libraries, components and example applications.
9. Execute some tests to verify that the installation is working.

These steps result in a development system with tools and runtime software ready to support development and native execution (on the development system) of OpenCPI components and applications. In most cases, steps 5 through 8 can be done with a single command.

> *It is a roadmap item to have a minimal, precompiled “component development kit” that requires fewer steps, and less file space.*

The following section describes phase 1 (Installing the Hardware and OS for OpenCPI Development). Subsequent sections describe phase 2 (Obtaining and Configuring OpenCPI and Prerequisites), and Phase 3 (Building OpenCPI and Testing Native Execution). For other target systems and platforms see the sections on embedded systems and FPGA platforms.

3.1 Installing the Hardware and OS for OpenCPI Development

Since a development host has no special hardware requirements, it must simply support the recommended and supported development host operating systems (currently CentOS6 Linux 64-bit). Some development tools (especially those for FPGAs) require large memories and exploit multiple CPU cores and thus the minimum memory should be at least 8GB, with 16B or more preferred. For test purposes, VMs with 2GB have been successful, but slow.

If the development host will also host other embedded CPU or FPGA cards acting as OpenCPI target platforms, the appropriate slots, cooling and power supplies should be considered.

If the development host will also be the runtime host for ethernet-attached devices (such as the Ettus N210), it is sometimes useful to use dedicated ethernet ports for such devices. In this case a host system with multiple ethernet ports/interfaces should be considered. This minimizes interference between Ethernet traffic to the locally-attached platforms, and general LAN or WAN/Internet traffic.

OpenCPI development is commonly hosted on laptops, server machines with card slots, and virtual machines hosted on other operating systems. One example system is a CentOS6 64-bit virtual machine running under the "Parallels" virtual machine system on Apple MacBook laptops. Another is a Dell server with well-powered PCI Express slots for hosting a number of FPGA and/or GPU boards.

3.1.1 Obtaining the CD Image File for the OS Installation

The normal operating system installation starts with a CD image downloaded from centos.org (or one it its mirrors). It is currently available from a mirror on this site:

`http://isoredirect.centos.org/centos/6/isos/x86_64`

The actual file name to download is:

`CentOS-6.5-x86_64-LiveCD.iso`

The **`md5sum.txt`** file in the same directory provides the md5 checksum for the file to check against after downloading using the md5 command (on MacOS or Linux):

```
% md5 CentOS-6.5-x86_64-LiveCD.iso
MD5 (CentOS-6.5-x86_64-LiveCD.iso) =
8c37390fa5d932d03feb05fba13fe92e
```

For a physical system, you can burn this file onto a real CD, and then boot from that physical CD. For a virtual machine, you can usually designate that the CD image file be mounted to the virtual machine as a virtual CD device.

Creating a virtual machine usually involves answering a few questions about the to-be-created VM, and then booting it from the Installation CD image file. Using the "parallels" VM system under MacOS, the most import questions to answer are the amount of memory to give to the VM, and the number of cores to provide. Each VM system does it slightly differently.

3.1.2 Booting from, and Running the CentOS6 installation CD

Booting the development host system from the CD proceeds the same whether it is a physical system or a VM. The CD boot comes up as a CentOS6 system hosted on the CD itself, and there is an icon on the desktop called “Install to Hard Drive”, which installs the CentOS6 OS on the hard drive (or virtualized HD for a VM). Double-clicking on this “Install to Hard Drive” icon will start the generic CentOS6 installation process: see the figure below.

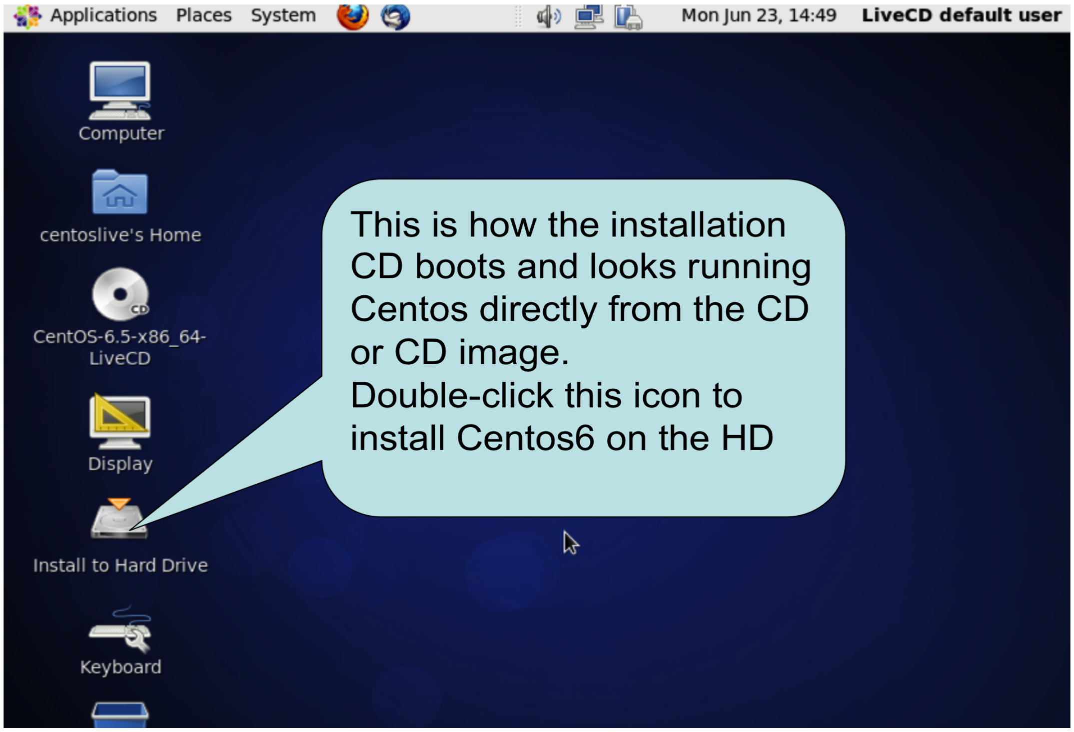

Figure 1: CentOS6 Installation Screen after booting from Installation CD

The “Install to Hard Drive” will then ask questions about:

- the keyboard language
- storage device types
- whether to preserve the HD contents (answer “no” for fresh or VM systems)
- the network host name of the system
- timezone
- root password
- whether to “use all space” on the HD (answer “yes” for fresh or VM systems)
- finally, whether to “write changes to disk” to make the HD installation

This results in the HD having a CentOS6 installation with the screen as follows:

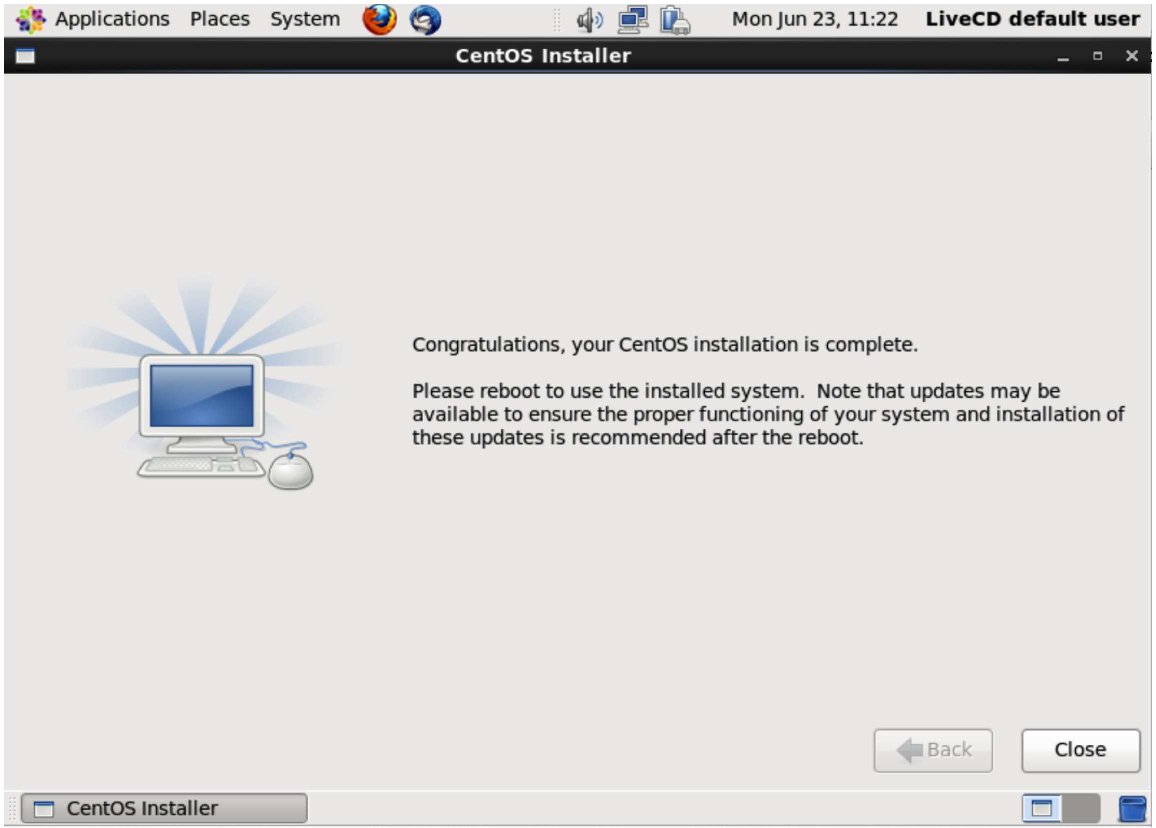

Figure 2: Screen after Installation to HD from CD or CD Image File

Pressing “Close” at this screen exits the initial CD-to-HD installation process, and you should now reboot using the “System->Shut Down...” menu item, and choose the “Restart” option. This will result in your new HD-based Centos6 installation booting for the first time.

3.1.3 Performing Final Installation Steps from the HD-based CentOS6 installation

When CentOS6 boots from the HD for the first time, it of course asks more installation questions. This phase of installation is done by what CentOS6 calls the “Setup Agent”. It will ask these questions:

- Agree to the license
- Create a user (use the “Advanced” button if you need to control UIDs*)
- Time of day, and whether to get time from the network-based time servers
- Enable Kdump by reserving some physical memory (mostly for kernel debugging)

The “forward” button is in the lower right corner of the screen.

After these questions are answered, the login screen comes up and you should login as the user you just created (not root). Assuming you have a good network connection, you should update your software using the “System->Administration->Software Update” menu item, and, assuming there are updates, the “Install Updates” button. You should repeat this process until there are no updates to install since there can be dependencies that require more than one “Software Update”. When there are no more updates, you should restart again using “System->Shut Down...->Restart. After restarting it is conceivable that there are more updates, so you should try “Software Update” one more time to verify there are no more updates.

At this point you have a fully updated (for the default set of software packages) CentOS6 system. If you are on a VM system with support for “snapshots”, you should probably take one at this point.

3.1.4 Enabling Your User for “sudo”

A number of scripts supplied by OpenCPI require that the user be enabled for the “sudo” command. You should add your user account to the list of accounts that are allowed to use “sudo”. For CentOS6 Linux, this can be accomplished in a terminal window (from Applications->System Tools->Terminal). With <RootPasswd> and <User> being replaced by your root password and your user name, do:

```
% su
Password: <RootPasswd>
# echo <User> ALL = ALL >> /etc/sudoers
# exit
%
```

The “% “ and “# “ are just command prompts: you don't type them. Be careful to use the two “>>” characters to append the line to the file. Of course if you are experienced with Linux, you may do this many other ways.

> You now have an installed, up-to-date operating system, with a user account that is sudo-enabled.

3.2 Obtaining the OpenCPI Code Base and Installing Prerequisites

OpenCPI has several software prerequisites, but only one of them is required to be installed *before* the OpenCPI code base is installed: **git**. The git program is used to download a local copy of the code base, and after that, a script *in* the code base is used to finish the installation of OpenCPI. To obtain "git", the following command should be issued at the command prompt in the terminal window:

```
% sudo yum -y install git
```

This ensures that you have the "git" package installed on your system so that we can use the "git" command. Git is the distributed revision control system used by OpenCPI. We will only use a few git features and commands for installation.

3.2.1 Obtaining the OpenCPI Codebase.

Assuming you are in a terminal window, in the directory where the codebase should go (in the directory named **`opencpi`**, *which the following command will create*), type:

```
% git clone git://github.com/opencpi/opencpi.git
```

This will create an **`opencpi`** subdirectory and populated it with the current OpenCPI code base: a "git clone" of the code base that can be easily updated in the future.

By default, the "git clone" operation downloads the "latest and greatest" or "bleeding edge" version of the code. This may or may not be what you want. After downloading, if you want a specific, perhaps more stable release, you use the "git tag" command to lists the tagged releases available, then set the code base to the one you want using "git checkout", using the release tag (as listed by "git tag") as an argument.

The OpenCPI releases are identified by calendar quarter, and patch release within that quarter. The release compatibility policy is to maintain component binary compatibility within releases for a quarter, and source compatibility (requiring rebuilding) for new quarterly releases. Patch releases within a quarter still require rebuilding OpenCPI itself since the distribution is source-based for now. Early releases of a next-quarter release are identified with minor releases starting with "rc" for "release candidate". Here are some examples of OpenCPI release tags:

```
OpenCPI-2014.Q3.3
OpenCPI-2014.Q4.rc0
```

To set the release of the codebase you download, you use the "git checkout" command with the tag as an argument:

```
% git checkout OpenCPI-2014.Q3.3
```

This will result in some messages about "detached HEAD", which can be ignored unless you are modifying OpenCPI itself, rather than just using it for component or application development. To do development on OpenCPI itself, and potentially submit patches, etc., you need to know more about git. To simply return the codebase to be the latest version, you can do that by using "master" as the tag.

```
% git checkout master
```

3.2.2 Installing Prerequisite Packages for OpenCPI

The next step is to run a script in the code base that will install other required packages. The script does this in two ways, depending on whether the package:

- is a standard optional package for CentOS6 using the CentOS “yum install” command, or
- needs OpenCPI-supplied scripts to do a custom installation

This script will first create the **`/opt/opencpi`** directory since that is where the second category of prerequisite packages are installed. To install the prerequisites, type:

```
% cd opencpi
% scripts/install-prerequisites.sh
```

This step and the next two can all be executed in one step using the “install-opencpi.sh” script for those who are impatient!

The standard packages will be installed globally on the system, and thus be available to all environments and applications. If they are already installed, nothing will be done. To see which packages are installed, look at this script and the ones it calls. The second category, installed with OpenCPI-supplied installation scripts, will be installed in a place (**`/opt/opencpi`**), where they will be used by OpenCPI, but will not be available globally and will not affect any other installation of the same package. If you need this directory to be somewhere else, make a symbolic link in **`/opt/opencpi`** that points to that other place.

> *It is a roadmap item to allow a directory name other than /**`opt/opencpi`** to be used.*

You now have an OpenCPI installation with all prerequisites installed, and are ready to build OpenCPI.

3.3 Building and Testing OpenCPI

Now that all the prerequisites and installed, we can build OpenCPI.

3.3.1 Building OpenCPI for Native Execution

Building OpenCPI builds several different things in different steps, but all with one script.

- Build the core software infrastructure libraries, and utility command executables.
- Build the software component libraries. (components in the default library usable for applications)
- Build some example applications
- Build the OpenCPI Linux device driver

All these steps are executed by the “build-opencpi.sh” script:

```
% scripts/build-opencpi.sh
```

When this script succeeds, all software aspects of OpenCPI have been built and can execute in the development host environment. No OpenCPI code has been executed at this point.

You now have an OpenCPI installation with all of OpenCPI, including libraries, commands, software components and applications, installed and ready to run. This does not include FPGA tools, code, or bitstreams, which are installed/built elsewhere.

3.3.2 Basic Testing of the OpenCPI Installation

Run the **`scripts/test-opencpi.sh`** script to perform a simple “smoke test” to see if OpenCPI appears to be installed correctly. This script runs a set of tests that are simple pass-fail tests that should complete successfully.

You now have an OpenCPI installation that appears to work correctly when running native applications.

3.3.3 Initializing your Environment to run OpenCPI tools, Scripts and Commands

Up to now, all the installation scripts have relied upon an environment that is set up with a standard script without any customization. This script is called **`env/default-env.sh`**. You should make a copy of this script (say in **`my-env.sh`**) for your self and make any changes necessary for your environment. For native execution, no changes may be necessary, but when third party tools are required for other platforms, their locations and/or license file locations typically need to be set in this file.

This script must be sourced (not executed) in any command shell or window that will use OpenCPI for development. This is accomplished by:

```
% source my-env.sh
```

It may be convenient to put this line in your “~/.profile” or “~/.bashrc” files, but do this with care since it may affect all command windows and scripts, which may cause other problems interacting with other package environments.

The “jkvm6.sh” script is an example that shows a number of possible settings, especially for FPGA tools.

3.4 Installation Summary

The primary OpenCPI development platform is CentOS6. Others may work, but are not tested.

The steps to be taken to install OpenCPI are:

1. Obtain an Operating System Installation CD (or for VMs, a CD image file)
2. Install the operating system
3. Update the operating system to the current patch level.
4. Enable the current user for “sudo” permission.

   The above is a generic OS installation that you may already have done.

5. Install the “git” software
6. Get the OpenCPI code base using “git clone”
7. Run the installation script for installing prerequisites.
8. Run the script to build OpenCPI
9. Run the script to do the installation runtime test of OpenCPI's

The actual minimum steps are, after 1) installing/updating/rebooting the OS, 2) enabling “sudo” and 3) running in the directory where the “opencpi” subdirectory will be *created*:

```
% sudo yum -y install git
% git clone git://github.com/opencpi/opencpi.git
% cd opencpi
% scripts/install-opencpi.sh
```

4 Embedded Systems as Target Hosts

In this document we use the term “embedded systems” for processors and systems that will execute OpenCPI components and applications, but are generally not used to build or compile OpenCPI or components. Our primary example is the Digilent ZedBoard, which has a Xilinx Zynq SoC chip which contains 2 ARM cores for software and an FPGA section for “gateware”.

When using any embedded system, the first step is to install the appropriate cross-compilation tools on your development system. While some embedded systems can actually host their own tools, we generally avoid this in order to avoid burdening the embedded platform with such tools, and also avoid problems when the OpenCPI development environment must be ported to a new platform.

If no Linux-hosted cross-tools are available for the embedded target, then the OpenCPI build/development environment must be ported to that platform, and any incompatibilities must be addressed. The OpenCPI development environment does not have many dependencies, but they must all be addressed on the new development platform. This is a partial list of dependencies that some aspects of the OpenCPI development environment require:

- C and C++ compilers
- Bash and Make
- Python
- Miscellaneous POSIX utilities such as “tr”, “sed”, “cp”, etc.

When other cross-tools and/or FPGA tools are required for a target platform, other prerequisites may be required.

Along with using cross-compilers we generally assume that the embedded system has a network interface that will allow it to mount and access the file systems on the development system where the OpenCPI codebase is built and cross-built. This also requires that the development system be enabled as a file server, and any associated firewall issues are addressed. When such network access is unavailable, then a small subset of the OpenCPI cross-built environment must be copied to the embedded system.

> *It is a roadmap item to automatically prepare such an “embedded runtime package” for OpenCPI.*

One key build option for OpenCPI is whether to build all libraries and executables with static linking or dynamic linking. When built with static linking, executables are truly standalone and don't require libraries to be copied to the embedded system.

For each embedded system that is supported, we divide the instructions into two steps:

1. Install cross development tools and cross-build OpenCPI on the development system.
2. Install the minimum environment on the embedded system to access OpenCPI via network, and set up the hardware as needed for OpenCPI.

4.1 The Digilent ZedBoard with Xilinx Zynq SoC processor

This hardware platform is the smallest and cheapest platform that can support software and FPGA development using OpenCPI.

> *Supporting the even-smaller "MicroZed" platform (with the Z-7010 device) is considered feasible and is a roadmap item.*

This section describes how to build OpenCPI for the ZedBoard and enable building applications and components for it. It is what you do on a development system (not on the ZedBoard) before doing anything on the ZedBoard hardware. The basic steps are summarized at the end of this section.

4.1.1 Establishing the Cross-Building Environment for OpenCPI targetting Zynq

4.1.1.1 Install Xilinx ISE and EDK tools

There are two aspects of building for Zynq: user mode code and kernel driver code. Both rely on the cross-compiler supplied in the Xilinx tools release (the EDK). Thus the first prerequisite is to install Xilinx ISE and EDK tools. We will refer to the Xilinx release number as $XRN, and the pathname where Xilinx releases are installed as $XROOT. In a typical installation, $XRN might be 14.6, and $XROOT might be /opt/Xilinx, so the current release would be installed in the

```
/opt/Xilinx/14.6
```

directory. This procedure has not been tested on ISE releases prior to 14.6 and is unlikely to work on prior releases. Be sure to include the EDK in the installation (when running the Xilinx installer) since that is where the cross compiler and some other required files come from. At the current time OpenCPI does not support the Xilinx Vivado tools, only ISE and EDK. The XRN and XROOT variables are notional for this document. You do not need to actually set or use those variables.

We assume a Xilinx ISE+EDK installation on CentOS6/64-bit Linux, the standard OpenCPI development host. Xilinx officially supports Red Hat Enterprise Workstation 6 (64-bit), of which CentOS6/64-bit is a free "clone", without any support.

We also assume a successful OpenCPI development host installation as described above. In the standard environment for OpenCPI development, the OCPI_BASE_DIR environment variable specifies where the OpenCPI code base resides.

In summary, the prerequisites for the ZedBoard (and other Zynq-based platforms) are:

1. A Xilinx ISE+EDK installation at $XROOT/$XRN, e.g. **`/opt/Xilinx/14.6`**
2. A working OpenCPI installation on CentOS6/64-bit at $OCPI_BASE_DIR

4.1.1.2 Establish and Customize your Cross Build Environment

To build OpenCPI for Zynq/ARM software development, we use the Xilinx-EDK-supplied ARM/Zynq cross-build tools (cross from Centos6 x86_64 to Zynq). There is nothing specific to the ZedBoard platform about these Zynq tools. The **`jkzed.sh`** file in the $OCPI_BASE_DIR is an example you can copy (say into **`myzed.sh`**) and customize as necessary for your own environment (perhaps changing the name of the license file). Your script may set the OCPI_XILINX_DIR, OCPI_XILINX_VERSION, and OCPI_XILINX_LICENSE_FILE environment variables if they are not the defaults. Assuming you have already built OpenCPI for Centos6/64-bit, you need to source this customized environment script in a new shell/window, in $OCPI_BASE_DIR, e.g.:

```
% source myzed.sh
```

This establishes a cross-build environment by setting environment variables that point to the cross-compilation tools in the Xilinx EDK installation.

4.1.1.3 Obtaining and Configuring the Xilinx Linux Source Tree for Zynq

The last preparation before building for zed is to download/clone the Xilinx Linux distribution source tree to support building the OpenCPI Linux kernel driver.

Since this Linux code is Xilinx-version-specific and is in sync with the ISE/EDK releases, we choose to place this linux source tree at the top level of the Xilinx release directory $OCPI_XILINX_DIR (i.e. not under a version directory). To clone and prepare this Lilnux distribution to enable building the OpenCPI kernel driver you run the following script in a shell/window where you have already set up the Zynq cross-build environment. The download/clone the Xilinx Linux source tree, run:

```
% platforms/zed/getXilinxLinux.sh
```

This will clone, checkout, and configure the Xilinx Linux source tree, at $OCPI_XILINX_DIR/$OCPI_XILINX_VERSION/git/linux-xlnx, so the OpenCPI kernel driver can be built against it. In this script, the checkout label is the latest one for the ISE version you have installed. You can look at the labels like this:

```
% (cd $OCPI_XILINX_DIR/git/linux-xlnx; git tag | grep 14.6)
xilinx-v14.6
xilinx-v14.6.01
xilinx-v14.6.02
```

The **`getXilinxLinux.sh`** script chooses the last one which should be the latest and greatest. This is to ensure that we are using any patches Xilinx has posted for this ISE release. Remember that we are only building the OpenCPI driver against this Xilinx Linux source tree, and *not* actually *building* the kernel.

> *Some users have had success with an ISE installation at 14.5, but using the Xilinx kernel associated with 14.6, which required hard-wiring this script to use 14.6 even though the installed ISE version was 14.5.*

For those that need to actually build and reconfigure the linux kernel, and replace the prebuilt one (as well as the device tree), there are other steps to take. This document instructions focuses on the default Linux kernel as supplied by Xilinx. Remember to rebuild the OpenCPI kernel driver whenever the kernel is changed, reconfigured or updated.

At this point you have a checked out and configured the Linux kernel source tree that corresponds to the kernel we will run on the ZedBoard, and we can build the OpenCPI

Linux kernel driver against this tree. The environment variable **OCPI_KERNEL_DIR** is set in the standard zynq setup file like this:

```
export OCPI_KERNEL_DIR=\
 $OCPI_XILINX_DIR/$OCPI_XILINX_VERSION/git/linux-xlnx
```

4.1.1.4 Build Software for Zed using the Xilinx-supplied Cross-Compiler Tools

After the above environment and Linux kernel setup, the normal OpenCPI build should function and succeed, but will now be cross-building for the Zedboard.

```
% scripts/build-opencpi.sh
```

This should succeed and produce no unusual warnings. It is essentially equivalent to:

```
% make && make rcc && make examples && make driver
```

It will build the Zynq versions of all OpenCPI libraries, executables, components, examples, and driver for the ZedBoard platform (in fact for any Zynq platform). This will *not* interfere with the libraries, executables and components already built for the native Cent06/64-bit environment. With OpenCPI, all compilation results are placed in target-specific directories, so building for multiple targets in the same tree is supported and expected.

4.1.1.5 Build FPGA libraries, components, platform and bitstreams

In order to build the HDL/FPGA code, including test bitstreams, you do:

```
% make hdl HdlPlatforms="isim_pf zed"
```

This will build all the primitive libraries, components, bitstreams for ZedBoard's Zynq chip and for isim (the ISE simulator). If you had already built OpenCPI HDL/FPGA code for other Xilinx targets, the isim_pf aspect of the build may already have been done and thus will not be rebuilt. For previous OpenCPI HDL developers: the OpenCPI “HdlTarget” for the ZedBoard is "zynq" and the “HdlPlatform” is "zed".

This is the end of building all the OpenCPI assets for development and execution on the ZedBoard.

4.1.2 Creating and Populating a Directory to Create a Bootable SD Card

The final software step to perform on the development system is to create and populate a directory to be copied to an SD card that will be plugged into the Zed board for booting. We use the Xilinx Linux binary releases at:

```
http://www.wiki.xilinx.com/Zynq+Releases
```

We simply use the prebuilt Xilinx Linux release corresponding to the Xilinx ISE tools release. (Vivado will come later...). We prepare an SD card by copying the files from the Xilinx release onto the card. If the SD card is not formatted correctly, you just need a single WIN95/FAT32 partition, and make a file system on it (using cfdisk and mkfs.vfat). If it is already formatted (like the SD card that comes with the Zed board), you simply copy files onto the SD card (when it is mounted on your development system). If your development system does not have an SD card slot, then you can use an SD card USB adapter.

This setup is the minimal required SD card contents. You can add anything else to it of course. Our standard development setup mounts the development system using NFS, so very few files really need to be on the SD card itself.

The following instructions assume that the SD card is plugged in and mounted on a Centos6/64-bit system (in an SD slot or a USB SD card adapter).

Xilinx provides releases at www.wiki.xilinx.com that are versioned the same as the Xilinx tools releases. We assume the version of Xilinx tools installed is at least 14.6. If not, replace "14.6" with the version you have installed. Releases before this may well not work.

These instructions prepare a directory tree for an SD card locally (in the **`platforms/zed`** directory), to be copied to a real SD card, which can then be unmounted and unplugged from the development system, plugged into the Zed board, and used to boot the Zed board.

These instructions assume you are now in the **`platforms/zed`** *directory.*

There is one file that must be customized for your network environment. You should make a copy of the "defaultsetup.sh" file into "mysetup.sh" and customize it, in particular, specifying at least:

- The NFS "share name" of your development system.
- The directory name relative to that mountable file system where OpenCPI lives.
- The system to be used as an NTP server.
- Your timezone name and specification (see the tzset(3) man page on Centos6).

At least in Centos6, the following command will print what the timezone should say:

```
% tail -1 /etc/localtime
```

xsThe network address of your development system will be specified later. To create a directory that can be copied to an SD card, you use the **`makeSD.sh`** script, which takes as an argument the current Xilinx tool ISE/EDK release.

```
% ./makeSD.sh 14.6
```

This script downloads the binary kernel release from Xilinx, unpacks it, creates a SD-$XRN directory locally (removing it first if it is there), populates it, including the **`mysetup.sh`** that you have just customized.

4.1.2.1 Copy the prepared SD file tree to the actual SD card.

Assuming you have your formatted SD card mounted at /media/xyz, and that you have a spare copy of everything that was on it, you would do:

```
<mount the SD card on /media/xyz>
% rm -r -f /media/xyz/*
% cp -R -p SD-14.6/* /media/xyz
<unmount the SD card to ensure it was completely written>
```

This creates a bootable card for the Zed board with the released Xilinx kernel and boot loader, and a generic "setup" script for getting OpenCPI going. The **mysetup.sh** that is customized for your network environment was also added to the SD card. It calls the "ocpizedsetup.sh" script with the appropriate arguments for your environment and does whatever other setup tasks you want. If you later update your **mysetup.sh** file, you can always copy it to the SD card later from the ZedBoard system itself by doing:

```
% cp /mnt/net/xxxxxx/opencpi/platforms/zed/mysetup.sh \
    /mnt/card
```

Where **xxxxxx** is the relative location of OCPI_BASE_DIR from your NFS mount.

Note that the original contents of the SD card can be found on the Digilent web site under the "out of box" project at http://www.zedboard.org/node/241. But you don't need this if you are just following these directions.

At this point you have a development environment ready for the ZedBoard, have built all the libraries, executables, components, and examples for both software and FPGA, and have prepared a bootable SD card.

4.1.2.2 Summary of Pre-Hardware Steps to Prepare to Run OpenCPI on a ZedBoard

From the top level directory of an OpenCPI installation a.k.a. OCPI_BASE_DIR, which is *already functional* for the Centos6/64-bit system, in a fresh/new shell/window, do these steps:

```
% cp jkzed.sh myzed.sh
{ customize myzed.sh for your build environment}
% source myzed.sh
% platforms/zed/getXilinxLinux.sh
% scripts/build-opencpi.sh
% make hdl HdlPlatforms='zed isim_pf'
% cd platforms/zed
% cp defaultsetup.sh mysetup.sh
{ customize mysetup.sh for your network environment }
% ./makeSD.sh
{ mount SD card onto something like /media/xyz, and perhaps remove all files }
% rm -r -f /media/xyz/*
% ./fillSD.sh /media/xyz
% sudo umount /media/xyz
```

Now everything is built and ready for the ZedBoard: you have a bootable SD card for using OpenCPI on a ZedBoard.

The next section describes how to install the hardware and set it up to execute based on the OpenCPI system you just built and the SD card you created.

4.1.3 Setting up the ZedBoard Hardware to Run OpenCPI

This section describes how, after installing Xilinx tools, installing and building OpenCPI (software and FPGA aspects) for Zynq/Zed, and preparing an SD card, you actually set up your ZedBoard itself to run OpenCPI applications and components on the Zed itself.

These steps do not enable any particular devices that might be attached to the FPGA/PL part of Zynq.

Always be careful of static electricity when touching the board. Touch some metal shell first each time. Start with the power cable disconnected. The metal shell of the network connector is a good thing to touch first.

4.1.3.1 Set up the hardware jumpers and switches

It is a good idea to have the Digilent ZedBoard Hardware Users Guide at:

```
http://www.zedboard.org/sites/default/files/documentations/
   ZedBoard_HW_UG_v2_2.pdf
```

The “Jumpers” section has good photographs showing the location of jumpers on the board. Only change the jumpers when the power is OFF. These are the required jumper settings:

- MIO 2: set to GND
- MIO 3: set to GND
- MIO 4: set to 3V3
- MIO 5: set to 3V3
- MIO 6: set to GND
  - J18: VADJ Set to 1.8V (the setting is labelled 1V8)
  - JP2: shorted
  - JP6: shorted (*note: we didn't have enough jumpers and this wasn't needed*)

All other jumpers should be left unshorted.

4.1.3.2 Insert the SD Card into the Socket on the Bottom of the Board.

Notice that the SD card is plugged in with its label down since that connector socket is on the back of the board.

4.1.3.3 Connect the Supplied Micro-USB-to-normal-USB-cable to a System

This cable is what provides access to the ZedBoard's serial console. It must be connected to a system with a terminal emulator set to 115200 board, 8 data bits, 1 stop bit, no parity. There are two different connectors on the board: be sure to use the one labelled: UART, and *not* the one labelled USB OTG.

4.1.3.4 Apply Power to the ZedBoard

With the power supply cable connected, turn the power switch on. The green POWER LED should come on. The system has actually booted itself, but without a console attached you can't really see anything.

If something smokes, you and the board are toast!

4.1.3.5 Get Access to the Serial System Console of the Zed, and Reboot it from There

Since the OpenCPI development environment is running on a Linux system (usually Centos6), you can just plug this USB cable into that Linux system, and it will automatically create a "/dev/ttyXXX" file for this connection when you plug it in and the Zed board is powered up. You need to determine this pathname and watch it come and go when you disconnect and reconnect the USB cable (or when the Zed board is powered down and up). Typically the file name is **`/dev/ttyACM0`**, but not always.

Without any other configuration of the Linux development host, you typically have to add read/write permission to this "/dev/tty*" file manually *each time the cable is connected and/or the ZedBoard is powered on*, using the command:

```
% sudo chmod a+rw /dev/ttyACM0
```

(assuming the **`/dev/ttyXXX`** file created for that cable is **`/dev/ttyACM0`**, which is the default for the first such hot-plugged USB/serial cable on Centos6). Again, this must be done each time the board is power cycled or disconnected.

Automating this (with udev configuration) is a roadmap item.

With the **`/dev/ttyXXX`** existing and with proper permissions, you must run a serial console terminal emulator program on the development system that is attached to the USB cable from the UART connector. There are many alternative terminal emulation applications to provide for this USB remote serial console for the ZedBoard (or any similar board). One that is available in the standard Centos6 repository is "screen", obtainable by simply using:

```
% sudo yum install screen
```

Then in any terminal window, you can do:

```
% screen /dev/ttyACM0 115200
```

To exit this program, you use the sequence: Control-a followed by backslash.

For Emacs users, you can use the serial terminal emulator built in to Emacs, by simply doing: **`M-x serial-term`** in a window, and providing the /dev/ttyACM0 name and 115200 baud rate when prompted. There are two modes in this window, "char" and "line". The default is "char" mode, where every character typed is immediately sent without any interpretation at all. This means no emacs commands work in that emacs window and you have to use the mouse to switch to another emacs window. The "line" mode is more line buffered, like "shell" mode, and all line editing and other emacs commands work fine. *But* the display can get confused in line mode so you have sometimes switch back and forth: control-c control-j switches to line mode, control-c control-k switches to char mode, control-c control-c sends a control C in char mode.

With a good terminal emulator connection, hit return a few times in the serial console window to see the "**`zynq login:`**" prompt, or, if the system was previously logged in and running, you might see the "**`root@zynq:~#`**" prompt. You must login as "root" with the password "root".

Sometimes if the screen or terminal emulator is confused, or if, after hitting "return", the prompt stays on the same line, try typing the command "clear", to clear the state of the terminal emulator.

In normal usage it is generally best to log into the board using SSH and leave the console window alone. After the board boots, you can log in with SSH without using the console at all if you know the DHCP network address — or see it on the console output.

You can now log in (root, root), and see that rebooting works fine by both trying the "reboot" command, as well as pressing the reset button on the board (labeled: PS_RST).

Now you know Linux can boot, and that you have console access.

4.1.3.6 Get the Network Working to Talk to the Development System

To enable the network connection to the development host, connect the ethernet connector on the ZedLoard to a LAN with DHCP. The green light that is part of the network connector socket should come on to indicate that you have a network connection.

Reboot the ZedBoard to have it come up properly on the network. You can reboot either by typing the "reboot" command, pressing the "PS-RST" button, or turning the power switch off and on. Power cycling might cause you to need to do another "sudo chmod a+rw /dev/ttyACM0". After reboot, and re-login on the console, you can use the **`ifconfig`** command to confirm that the "eth0" interface has an internet address, courtesy of DHCP. E.g. this shows we have the internet address 10.0.1.108.

```
root@zynq:~# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:0A:35:00:01:22
      inet addr:10.0.1.108  Bcast:0.0.0.0  Mask:255.255.255.0
      inet6 addr: fe80::20a:35ff:fe00:122/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:42 errors:0 dropped:1 overruns:0 frame:0
      TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:6793 (6.6 KiB)  TX bytes:1737 (1.6 KiB)
      Interrupt:54 Base address:0xb000
```

The network address is also printed out on the console during booting. To check connectivity, use **`ping`** in both directions: ping your development host from the ZedBoard console and ping your ZedBoard from your development host. If both pings work you have network connectivity.

4.1.3.7 Enable NFS on the Development Host for the "share" that includes OpenCPI

There are many "cookbook" files for setting up NFS on Linux systems. You need to export a mountable directory above the directory where OpenCPI is installed. An example might be, if OCPI_BASE_DIR was **/home/user/opencpi**, and **/home** was NFS-mountable, then the “share” would be **/home**, and the OpenCPI subdirectory would be **user/opencpi**. On Centos6, you can enable NFS using the System->Administration->Services menu item. You may also need to enable local clients in the System->Administration->Firewall menu item.

To verify you have NFS connectivity try to manually mount it on the ZedBoard by typing, on the Zed console:

```
% mount -t nfs -o udp,nolock,soft,intr <ipaddr>:<dir> /mnt/net
```

Where <ipaddr> is the address of the development host, and <dir> is the directory being exported/shared by the server. The OpenCPI directory will be underneath that directory.

If that succeeds, you have a good NFS connection. unmount it using the "umount" command since the automated scripts will mount it next.

4.1.3.8 Run the Setup Scripts after Booting to Access the Development System

When Linux boots on the Zed board from the SD card created as above, it creates a memory-based root file system, initialized from decompressing the **uramdisk.image.gz** file on the SD card. It does *not* mount the SD card itself as a file system, although that can be done after booting. This means that anything you write to the root file system is *not* persistent. To create any persistent file you must first mount the SD card file system and write files to that file system, usually mounted as **/mnt/card**. You can obviously do lots of customization to the boot process and have it do what you want, but we don't get into that here.

Our standard setup uses the built-in NFS client to access the OpenCPI code base from these types of systems (in a development environment), and the default Linux configuration built by Xilinx happily includes NFS client support.

After booting, and logging in as root (either via the serial console or ssh), the first command should be:

```
root@zynq:~# mount /dev/mmcblk0p1 /mnt/card
```

This allows you to use the contents of the SD card for something other than booting. After this you can source the startup script on the SD card, supplying the development host's IP address as an argument.

```
root@zynq:~# source /mnt/card/mysetup.sh 10.0.1.108
```

These two commands, executed once per boot (not per ssh login), are all that is necessary to enable OpenCPI (unless you take additional measures to execute them automatically at boot time). If this script works ok, then it proves you can run OpenCPI utilities (and applications).

After these commands are run after booting, any new logins will automatically run the appropriate setup actions for that login shell.

4.1.3.9 Confirm OpenCPI Software Operation.

The **mysetup.sh** script above already sets the path to find OpenCPI executables and software components. It may also change to some working directory. Below "xxxxxx" is the path to get to the OpenCPI development tree. The default setup script (that you modified), in fact changes to this directory in any case.

```
zynq> cd /mnt/net/xxxxxx/opencpi/tools/cdk/examples/xml
zynq> ocpirun -v -d -t 5 -mbias=rcc tb_bias
```

This should run the tb_bias application (using pattern->bias->capture) components, forcing it to run in software, and not the FPGA.

4.1.3.10 Confirm OpenCPI execution of an FPGA-based application

There is no bitstream loaded on power up, but one will be loaded automatically as needed. So, assuming your OCPI_LIBRARY_PATH points to the builtin bitstream (which the default setup file does), you can run:

```
zynq> ocpirun -v -d -t 5 -mbias=hdl tb_bias
```

which runs the same application with the bias component on the FPGA, with the appropriate bitstream being automatically loaded.

The roadmap items that will make this all a bit more convenient are:

- Cause the /mnt/card mounting to happen automatically at boot time, avoiding the need to mount it to access the startup script
- Perhaps cause the "setup" script to be run automatically at boot time, avoiding the command to run the startup script
- On the development system, provide udev rules to avoid the "sudo chmod" command when using the terminal emulator.

5 FPGA Platforms Based on PCI Express Cards

Any development host system that has PCI Express slots can host FPGA cards where the FPGA is attached directly to the PCI Express fabric. All PCI Express-based FPGA cards require that the development host installation is complete and functional, including the Linux kernel driver. At this point supported FPGA cards can be installed and used as OpenCPI processors that can host OpenCPI components built for those FPGAs.

Using only subsets of the complete OpenCPI development environment in these configurations is a roadmap item.

Each PCE Express-based FPGA card has its own installation issues, but most installations are similar. Assuming there is already OpenCPI support for the card, the steps necessary to bring it up as an OpenCPI processor are:

- Ensure sufficient power and cooling for the card
- Configure any required jumpers and/or switches on the card
- Enable bitstream loading, and JTAG access.
- Plug in the card and power up the system
- Load an OpenCPI bit file into the power-up flash memory on the card
- Reboot the system
- Ensure the card is now recognized and usable.

5.1 Installation of PCI Express-based FPGA Cards

This section describes the installation process in general that applies to all PCI Express-based FPGA cards. Following sections provide details for some of the cards that are supported for OpenCPI. We use the term "bitstream" in this section for the file containing an FPGA configuration to be loaded into the FPGA. Bitstream files are created by the OpenCPI development process when the target platform is an "HDL" platform, which usually means an FPGA.

5.1.1 Ensure sufficient power and cooling for the card

PCI Express-based FPGA cards have a range of cooling and power requirements, and some even require that the chassis and box they are plugged into be left open for access to connectors etc.: they are "lab cards" that remain fully exposed. Others are typical PCI-Express cards that simply plug into a slot, and the box can be closed. Frequently there are LEDs and other indicators, switches or displays that are useful to see when the card is fully exposed.

Some cards require extra power supply cables to supply more power than is available through the backplane connector. Some of these have their own "power blobs" that connect directly to AC outlets, while others have power cables that attach to the power harness in the box that usually supplies power to hard drives. For the latter case you may need an adapter cable for the power harness in the system box.

It is out of scope here to provide guidance on power and cooling issues, but ignoring the issues can frequently result in unreliable or broken hardware. Usually the hardware manuals of these cards provide sufficient guidance.

This step is complete when you have decided on how and where (which slot) the board should be plugged in, how it will receive sufficient cooling, and how its power supply requirements (and cables) will be satisfied.

5.1.2 Configure any required jumpers and/or switches on the card

Nearly all cards have hard-wired jumpers and switches that configure how the board should operate. For the purpose of OpenCPI, the most common options relate to how the board powers up, and how the PCI-Express interface behaves. For most cases you should configure the board so that it boots a bitstream from a part of flash memory that can be written with a new bitstream, usually via JTAG. Even during active bitstream development and loading via JTAG, it is required to have a baseline OpenCPI bitstream loaded in flash memory that is automatically loaded on power up and reset.

Some PCI-Express-based FPGA cards have an option to disconnect the PCI Express interface or to become the "root" of the PCI Express fabric. Neither of these options should be selected. The PCI-Express interface should be a normal peripheral endpoint (both master and slave) on the fabric.

Normally the bitstream should be reloaded from flash upon PCI-Express fabric reset, although there are cases where this is not strictly necessary.

5.1.3 Enable Bitstream Loading, and JTAG access.

As supported by OpenCPI, bitstreams can be loaded in one of 4 ways:

- On power-up and/or PCI-Express fabric reset, from on-board flash memory.
- On command, from on-board flash memory.
- On command, from JTAG
- On command from some other processor-accessible interface.

It is a roadmap item to support reloading a (part of) a bitstream via the PCI-Express interface itself.

If JTAG is required (which it normally is) a cable (and possibly a JTAG pod) must be connected to the board, and to a (usually) USB port on the development system.

When more than one card is in a system, and thus multiple JTAG cables are attached to multple USB port, the OpenCPI system must know which JTAG cable is connected to which card. This is done by making this association in the OpenCPI system configuration file, in **`/opt/opencpi/system.xml`**. An example is:

```
<opencpi>
 <container>
    <hdl>
      <device name="0000:05:00.0" esn="000013C1E1F401"/>
    </hdl>
  </container>
  ...(other xml)
</opencpi>
```

This example says: the OpenCPI HDL (FPGA) device found at PCI slot named "0000:05:00.0" should be associated with the USB-based JTAG pod with the electronic serial number (ESN) of **`000013C1E1F401.`** The way to find the serial number is vendor-specific, but the script: probeJtag will scan for JTAG cables for both Xilinx and Altera. Unfortunately not every JTAG pod has such a serial number, and thus more than one of them cannot be used in system. The "lspci" Linux utility can be used to list PCI-Express slots and information about the cards that are plugged into them.

This is only necessary when there is more than one such JTAG cable in a system.

5.1.4 Plug in the Card and Power up the system

After the cards are plugged in and the system is powered up, there is usually some board-specific LEDs that show that the power is good, the PCI Express bus is alive, and perhaps a factory default bitstream has been successfully loaded from flash. See the sections for individual boards.

5.1.5 Load an OpenCPI bitstream into the power-up flash memory on the card

PCI-Express cards will not be discovered by OpenCPI unless they are currently running a bitstream that was created by OpenCPI. Thus part of installation is to write a bitstream to the card's flash memory. As mentioned above, the board should be configured to boot from this flash on power up or PCI-Express reset. A good candidate for loading is the “tb_bias” bitstream that is built for all supported platforms. If you are in the top level of the OpenCPI tree, an example of loading this default bitstream is:

```
loadFlash alst4 hdl/platforms/alst4/tb_bias.bitz 01c4b5f5
```

This command loads the flash memory of an “alst4” PCI-Express card (the Altera Stratix4 development board) from the OpenCPI default bitstream for the alst4, using the JTAG pod whose serial number is 01c4b5f5.

There is always also a vendor-specific method for those that are experienced users of that vendors tools, but the actual file to load would not be the “.bitz” file but rather a vendor-specific file built as part of OpenCPI's build flow.

It is a roadmap item to specify the existence of a PCI-Express card in the system configuration file, and avoid the flash bitstream requirement. This mode is currently supported only on the ZedBoard embedded processor.

5.1.6 Reboot the System and Test OpenCPI's Ability to See the Card.

After the initial flash bitstream is installed on the card, reboot the development host system. Use the OpenCPI utility called **`ocpihdl`** to check that the board is found. The following command will search for recognizable cards and print out their information:

```
ocpihdl search
```

This command should print out the cards it could find, along with the PCI-Express address for each. If you have more than one of the same card, in different slots, you will see them both, although other than the slot, there is no other identifying information.

It is a roadmap item to display the electronic serial number of the card/chip itself, although not all boards or chips have such unique identification.

5.2 *Xilinx ML605 PCI-Express Card as an OpenCPI HDL platform*

The ML605 is a PCI-Express-based Xilinx development board with a Virtex-6 FPGA, along with a variety of useful peripherals. It costs $1795 (as of July 2014), and includes a license to the Xilinx tool set that is limited to targeting this exact board. It can be purchased either from Xilinx directly, or through Avnet. The URL page for this card is:

`http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm`

5.2.1 Hardware Setup for ML605

As with any exposed electronics, care should be taken with ESD (static electricity) when handling and configuring the board. To configure the hardware, use the document: “UG534: ML605 Hardware User Guide”.

There are many hardware options for how the ML605 can be used, but we focus here on the minimum changes from the default configuration that is supported by OpenCPI.

The default settings of switches and jumpers are listed in Appendix A. The non-default settings necessary for using the ML605 with OpenCPI are:

- Set the S2 DIP switches for “Slave SelectMAP Platform Flash XL” mode.
  Per Table 1-27, S2.1: On, S2.2: Off, S2.3: Off, S2.4: On, S2.5: On
- Set jumper J42 to PCIe present with width x4 (short pins 3 and 4).
  There is no technical reason why other PCI-Express widths could not be used, but the OpenCPI support for the ML605 has only been configured and tested for x4.
- Remove the “legs” from the ML605 card so it can be inserted into the motherboard.
- Ensure that SW2 (the external power switch) is set to OFF until after the board is plugged in.
- Connect *EITHER* (one but *NOT BOTH*) the J25 connector to the PC power harness (typically used for hard drives) OR the J60 connector to the AC power adapter.
- Connect a USB cable to the JTAG USB port, and leave the other end unconnected.

After the board is plugged into the PCI-Express slot, switch the SW2 power ON, and connect the JTAB/USB cable to a USB port on the development system. The SW2 switch should be on whenever the system is powered up.

If you have not yet loaded the initial OpenCPI bitstream into the flash memory on the ML605 card, it should boot the factory default bitstream. In this case the Linux **`lspci`** command should show the board as a Xilinx vendor board. After loading the initial OpenCPI bitstream into flash, and resetting the system, the **`ocpihdl search`** command should find it as an OpenCPI platform.

5.3 Altera ALST4 PCI-Express Card as an OpenCPI HDL platform

This platform is the PCI-Express Altera development card for Stratix4, which has the smaller **s4gx230** part on it. The OpenCPI name for this platform is “alst4”.

5.4 Altera ALST4X PCI-Express Card as an OpenCPI HdlPlatform

This platform is the PCI-Express Altera development card for Stratix4, which has the larger **s4gx530** part on it. The OpenCPI name for this platform is “alst4x”. It only differs from the ALST4 platform is that has a larger capacity FPGA chip on it.

6 FPGA Simulation Platforms

In OpenCPI, an FPGA simulator is “just another platform”. You build components for it as a platform, and run component-based applications running components on this platform. Using such simulators with OpenCPI is described in the OpenCPI Component Developer Guide. Other than simply installing the simulation software according to the vendor's instructions, there is little extra to do to use these simulators with OpenCPI. The one OpenCPI-specific task for using the simulators as platforms is to set up various environment variables in the OpenCPI environment setup script.

It is currently a roadmap item to run OpenCPI applications where some components are running in the simulator and some components are running on other platforms (software or FPGA). Presently OpenCPI applications that run on the simulator must have all the components running in the simulator. Note that the “file_read” and file_write” components do have implementations that run in simulators.

6.1 Modelsim

Modelsim is a third party simulator offered by Mentor Graphics Corporation. Although Altera provides a stripped down version of Modelsim, that version is not usable with OpenCPI since it does not support mixing VHDL and Verilog.

To use Modelsim with OpenCPI, you need the version that runs on the Linux OS running on your development host, and also that supports both VHDL and Verilog mixed in the same design. Centos6 is supported.

To use Modelsim you must set the following environment variables in your OpenCPI environment setup script (that you “source” in new shell command windows):

- OCPI_MODELSIM_DIR should be set to the Modelsim installation directory.
- OCPI_MODELSIM_LICENSE_FILE should be set to the name of the license file.

These instructions have been used with node-locked Modelsim licenses. Other license configurations my require other actions.

6.2 Xilinx Isim

The “isim” simulator comes as part of the Xilinx ISE tool set. It is fully supported as a target platform on which to execute OpenCPI subassemblies.

Using **isim** requires no additional setup beyond installing the Xilinx ISE tool set for any other Xilinx platform. The environment variables related to using isim are:

- OCPI_XILINX_DIR for the Xilinx ISE installation directory (e.g. /opt/Xilinx)
- OCPI_XILINX_VERSION_DIR for the version subdirectory (e.g. 14.6)
- OCPI_XILINX_LICENSE_DIR for the license file you are using.

As with ISE installations for hardware platforms, you must also “source” the environment script, “env/xilinx.sh” in your setup script. An example section of a setup script for using Xilinx ISE is:

```
export OCPI_XILINX_DIR=/home/jek/mac/Xilinx
export OCPI_XILINX_VERSION=14.6
export OCPI_XILINX_LICENSE_FILE=/home/jek/mac/Xilinx/Xilinx-License.lic
source ./env/xilinx.sh
```

6.3 Xilinx xsim

This is the simulator that comes with the Xilinx Vivado tool set. It is currently not supported for OpenCPI.