

OpenCPI Installation Guide

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
1.02	Update details for CentOS6 and Zed	2015-01-31
1.03	Update new simplified ZedBoard Installation	2015-02-27
1.1	Add ML605 details, change to use new/std doc template, bug fixes	2015-06-26
1.2	Add system.xml details	2016-05-18
1.3	Minor update for 2017.Q2	2017-09-07
1.4	Update for 2018.Q3, simplified CentOS and ZedBoard installation	2018-08-27
1.5	Update for 1.5	2019-04-30

Table of Contents

1	References.....	4
2	Overview.....	5
3	Installing OpenCPI on Development Hosts.....	6
3.1	Installing the Hardware and OS for OpenCPI Development.....	7
3.1.1	Obtaining the CD Image File for the OS Installation.....	7
3.1.2	Booting from, and Running the Installation CD/Image.....	8
3.1.3	Enabling Your User for “sudo” and Installing “git”.....	11
3.1.4	Summary of CentOS6/7 Installation Steps Prior to Installing OpenCPI.....	12
3.2	Obtaining the OpenCPI Code Base.....	13
3.3	The Single Command Installation Process.....	15
3.3.1	Installing Required Standard Software Packages for OpenCPI.....	16
3.3.2	Installing Prerequisite Packages for OpenCPI.....	16
3.3.3	Building the OpenCPI Framework and its Built-in Projects.....	17
3.3.4	Testing the Software Aspects of the Installation.....	17
3.4	Configuring your Environment to Run OpenCPI.....	18
4	Embedded Systems as Target Hosts.....	21
4.1	The Digilent ZedBoard with Xilinx Zynq SoC Processor.....	23
4.1.1	Establishing the Cross-Building Environment for OpenCPI targeting Zynq.....	23
4.1.2	Creating and Populating a Directory to Create a Bootable SD Card.....	26
4.1.3	Setting Up the ZedBoard Hardware to Run OpenCPI.....	29
5	FPGA Platforms Based on PCI Express Cards.....	35
5.1	Installation of PCI Express-based FPGA Cards.....	36
5.1.1	Ensure Sufficient Power and Cooling for the Card.....	36
5.1.2	Configure any Required Jumpers and/or Switches on the Card.....	36
5.1.3	Enable Bitstream Loading, and JTAG Access.....	37
5.1.4	Plug in the Card and Power up the System.....	37
5.1.5	Load an OpenCPI Bitstream into the Power-up Flash Memory on the Card.....	38
5.1.6	Reboot the System and Test OpenCPI's Ability to See the Card.....	38
5.2	Xilinx ML605 PCI-Express Card as an OpenCPI HDL Platform.....	39
5.2.1	Hardware Setup for ML605.....	39
5.3	Altera ALST4 PCI-Express Card as an OpenCPI HDL Platform.....	40
5.4	Altera ALST4X PCI-Express Card as an OpenCPI HDL Platform.....	41
6	FPGA Simulation Platforms.....	42
6.1	Modelsim.....	42
6.2	Xilinx Isim.....	43
6.3	Xilinx xsim.....	43

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 Overview	OpenCPI	Public URL: https://opencpi.github.io/releases/1.5.0.rc/doc/OpenCPI_Overview.pdf

2 Overview

This document describes how to install OpenCPI from the source code repository. 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.

This document does not describe installation of OpenCPI based on packaged binary distributions, e.g. from CentOS or RedHat RPM packages. That is described in other documents at <https://opencpi.github.io>.

The default installation platform for OpenCPI development hosts is CentOS7 Linux x86_64 (64-bit). Other Linux variants and 32-bit systems have been used successfully, but CentOS7 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 exist for other target processors and technologies such as the Xilinx Zynq SoC (with ARM processor cores and FPGA resources), and various FPGAs. For embedded CPUs, preference is given to *cross-building* when possible, rather than self-hosting development tools on such platforms, in order to limit the complexities of installing tools on different development hosts.

This 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 it 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 on (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 and CentOS7. If your system is already installed, this phase is not needed.

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. *[roadmap item: officially support installation without **git**]*
5. Install standard required 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 9 can be done with a single command.

This installation process is based on source code that is downloaded and built on the development host. Other prebuilt/binary installations are available at <http://opencpi.github.io>.

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*

The quick description of this OS installation section is: *install CentOS6 or CentOS7 with the “git” software installed and your user id enabled for “sudo”.*

This step is only necessary if you are installing the OS from scratch. If your OS is already installed and up to date, your account is already **sudo**-enabled, and the **git** command is available on your system, you can skip to the [Obtaining the OpenCPI Code Base and Installing Prerequisites](#) section below.

To just deal with the **sudo** and **git** issues on an installed OS, skip to: [Enabling Your User for “sudo” and Installing “git”](#)

Since a development host has no special hardware requirements, it must simply support the recommended and supported development host operating systems (currently CentOS7 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 16GB or more preferred. For test purposes, VMs with 2GB have been successful, but slow.

If the development host system 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 CentOS7 64-bit virtual machine running under the “Parallels” virtual machine system on Apple MacBook Pro 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 of its mirrors). Two examples of CentOS installations are described here. Many installation scenarios are possible.

For a physical system, you can burn this CD image file (a.k.a. ISO file) onto a real CD/DVD, and then boot from that physical CD/DVD. For a virtual machine, you can usually designate that the CD/DVD image file be mounted to the virtual machine as a virtual CD/DVD 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/DVD image file. For most VM systems, the most important 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.

Booting the development host system from the CD or CD image file proceeds the same whether it is a physical system or a VM.

For the case of using the Parallels VM system on Macs we select “customize settings before installation”, and set memory at 4GB and 2 cores/CPUs.

3.1.1.1 *CentOS6: with the Live CD Installation Image (CentOS7 is below)*

The CD image for CentOS6, for the “live CD” installation, is currently available from a mirror on this site:

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

The trailing slash is required in the above URL. The actual file name to download is:

CentOS-6.7-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 the **md5sum** command on Linux:

```
% md5sum CentOS-6.7-x86_64-LiveCD.iso
339b607c497fe6b524da36803420ca61 CentOS-6.7-x86_64-LiveCD.iso
```

This installation, when booted from the CD image, runs Linux itself, with a desktop environment, that can then be used to install your system.

3.1.1.2 *CentOS7: with the Minimal CD Installation Image*

The CD image for the minimal CentOS7 installation is available from mirrors at:

http://isoredirect.centos.org/centos/7/isos/x86_64/

The trailing slash is required in the above URL. The actual file name to download is:

CentOS-7-x86_64-Minimal.iso

This installation is suitable for command-line only (no GUI) headless systems, and booting from the CD (image) immediately runs the installer. Other CentOS7 installation images are available with more packages preinstalled on the image.

3.1.2 *Booting from, and Running the Installation CD/Image*

3.1.2.1 *CentOS6 with Live CD Image*

The Live CD/DVD boot-up 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.

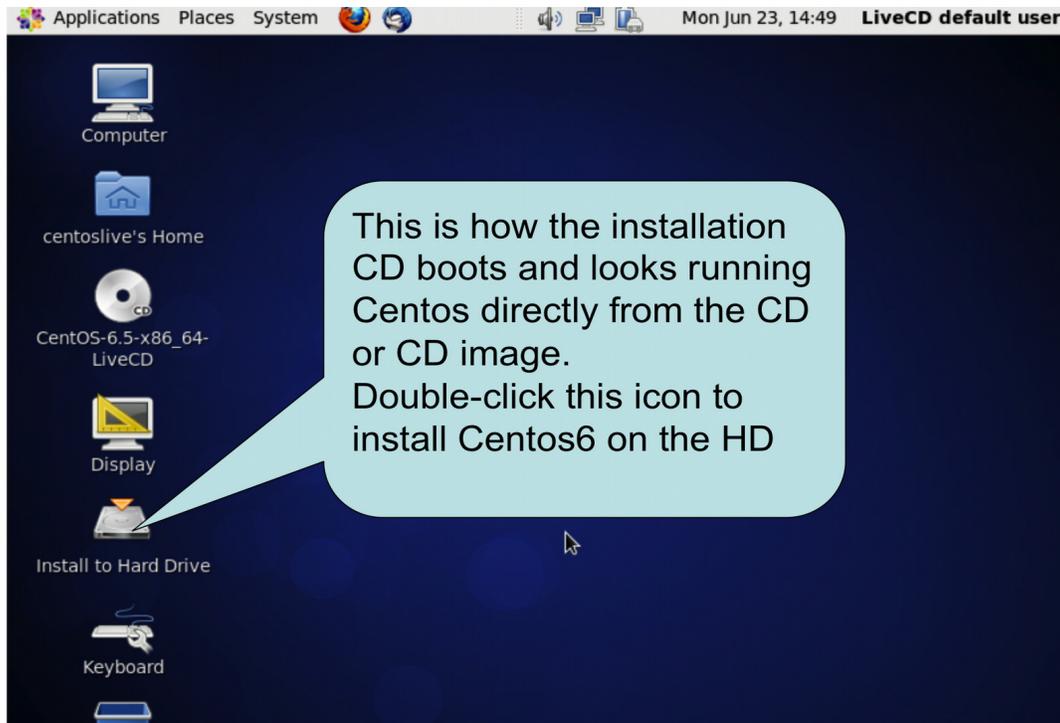


Figure 1: CentOS6 Installation Screen after booting from Installation CD

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

- the keyboard language (we chose *English (United States)*)
- storage device types (we chose Basic Storage Devices)
- whether to discard the HD contents (we answer Yes, discard any data)
- the network host name of the system (we chose livecd.centos.6.7)
- timezone
- root password
- whether to “use all space” on the HD (we chose “Use All Space”)

Finally, select “write changes to disk” to make the HD installation complete.

This results in the (possibly virtual) HD-based system 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. You should now:

- Remove/eject the installation CD/CD-image to ensure it is not used on the reboot.
- 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, and asking more installation questions like:

- License agreement (agree)
- Create User

In certain NFS mounting scenarios it is useful to create a user with the same user-id number and group-id number as the host system you will use as an NFS server. In this case select the “Advanced...” button in the CreateUser screen and create the user there.

- Date and Time (use NTP or set manually)
- Kdump (disable in most cases)

At this point you can log in as the newly created user.

3.1.2.2 *Performing Software Update on the New CentOS6 Installation*

Assuming you have a good internet 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.

3.1.2.3 *CentOS7 with Minimal CD Image*

When this image boots it immediately runs an installer that asks for things like language, and root password. It is better to not create users at this point, but to simply run the installer. When the installed system boots, it is in command line (shell) mode. At this point you must at least enable the network interface, if not done during the installer, using the **nmtui** tool (the default network interface is **eth0**), and then update your system using the

```
# yum update
```

command. To ensure you are up to date, you must reboot (using the **reboot** command) after **yum update**, and then, after the reboot, run **yum update** again, and repeat this process until it says there are no updates.

On the command line you can add users, using the simple **adduser** command. An example would be:

```
# adduser -m -N -r -u 501 -g 20 user1
```

You only need to specify the user and group ids if you are trying to match them to an existing installation for convenient NFS mounting. Otherwise you can simply use:

```
# adduser -m user1
```

This would use default behavior to create the account and home directory.

3.1.3 *Enabling Your User for “sudo” and Installing “git”*

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). On the CentOS7 minimal installation you are already at the command line. 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.

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 you 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.

3.1.4 Summary of CentOS6/7 Installation Steps Prior to Installing OpenCPI

- Download the CentOS CD image file, and check the md5 digest.
- For physical systems, burn a CD from that file.
- Boot the physical or VM system from the CD or CD image file.
- Run the installer using the “Install to HardDrive” icon and answer questions.
- Boot your newly installed system and answer more questions.
- Upgrade the software repeatedly until there are no more updates available.
- Enable your user ID for the **sudo** command.
- Install the **git** configuration management software.

You now have an installed, up-to-date operating system, with a user account that is sudo-enabled and the **git** software configuration management package installed.

3.2 Obtaining the OpenCPI Code Base

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

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

This will create an **opencpi** subdirectory and populated it with the current OpenCPI code base: i.e., a “git clone” of the code base that can be easily updated in the future. You should change into this directory for further commands:

```
% cd opencpi
```

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 list the tagged releases available, then set the code base to the one you want, using **git checkout**, with the release tag (as listed by **git tag**) as an argument.

The OpenCPI source code releases are tagged with the following format:

```
OpenCPI-<year>.Q<quarter>-v<major>.<minor>.<patch>
```

E.g. a recent release was:

```
OpenCPI-2018.Q4-v1.4.0
```

The release compatibility policy is to maintain component binary compatibility between patch releases *within* the minor release, and source compatibility (requiring rebuilding) for new minor releases within the major release. Patch releases within a minor release still require rebuilding OpenCPI itself since the distribution is source-based. Early releases of a next minor release are identified with minor releases starting with “rc” for “release candidate”, e.g.:

```
OpenCPI-2019.Q2-v.1.5.0.rc0
```

To set the release of the codebase after you have downloaded, you use the **git checkout** command with the release tag as an argument:

```
% git checkout OpenCPI-2018.Q4.v1.4.0
```

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 (this is not recommended for those not interested in developing OpenCPI itself):

```
% git checkout master
```

Whenever you check out a different tag *after* any building activity, you should perform a clean operation on the code base:

```
% make cleaneverything
```

Summary of Steps to Prepare the OpenCPI Code Base:

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

```
% cd opencpi
```

```
% git tag
```

```
% git checkout OpenCPI-2018.Q4.v1.4.0 # use tag from git tag output
```

You now have an OpenCPI source tree configured (checked out) for a specific tagged version of OpenCPI.

3.3 The Single Command Installation Process

After the above steps to obtain the code base (`git clone` and `cd` to the `opencpi` subdirectory), you are in the top-level directory of the source distribution (a.k.a. the git clone). At this point one command can be used to perform the rest of the installation, including running some tests to ensure that the installation is working. This command is:

```
./scripts/install-opencpi.sh [<sw-platform>]
```

The optional argument is a software platform that the installation process should target. The default software platform is the one you are running on. This `<sw-platform>` option is typically used for embedded, cross-compiled platforms like `xilinx13_4` for the Zynq/Arm SoC-based systems.

During the several phases of this script, it will perform some tests that require the user to enter the `sudo` password, unfortunately requiring monitoring of the process. It is *not* recommended nor supported to perform the whole installation under-`sudo/as-root`.

If this command completes and succeeds, you are ready to use OpenCPI for the targeted platform and can proceed with the environment setup described in the section [Initializing your Environment for OpenCPI](#). The rest of this section provides more detail about how this script performs the installation. After running this script:

You now have an OpenCPI installation built and tested, ready to be used after some additional environment setup steps. This does not include FPGA tools, code, or bitstreams, which are installed/built elsewhere.

The following description is for the curious or when the above script fails. If it succeeds the rest of this section is just a detailed description of what the script does. This script performs these functions, in order, with each depending on the previous ones:

1. Install standard packages required for OpenCPI development, globally on the system, from the OS's repository. This is accomplished using the underlying `install-packages.sh` script described below. Important examples of such packages are “make” and “python”. For CentOS or Redhat Linux, this script uses the `yum install` command.
2. Build and/or install prerequisite packages for OpenCPI, in an area only used by this installation of OpenCPI (i.e. *sandboxed*). This will cross-compile these packages if necessary for the targeted software platform. The `install-prerequisites.sh` script performs this function and is described below. This usually involves downloading source tarballs for the package and (cross) building them.
3. Build the OpenCPI framework (libraries and executables) itself. This step and the following one are accomplished using the `build-opencpi.sh` script described below.
4. Build the built-in projects that are part of the OpenCPI source distribution, for all software assets (RCC workers and ACI applications).

5. For the installation platform (not embedded/cross-compiled platforms), run a number of tests to verify the installation. This step uses the **ocpitest** command.

These underlying steps can be run individually for troubleshooting purposes, but are otherwise unnecessary. The rest of this section describes them in more detail, but can be skipped if the **install-opencpi.sh** script succeeds.

3.3.1 Installing Required Standard Software Packages for OpenCPI

This step (#1 above) uses the underlying script:

```
scripts/install-packages.sh [<sw-platform>]
```

Most OS distributions are associated with an internet-based repository of software packages, and have a way to install packages from that repository. For Redhat and CentOS Linux systems, the **yum** command is used to access software packages in the repository for that OS. Most embedded/cross-compiled OSs do not have such a repository, but might. When this type of package is installed on your system, it is globally visible and usable and is not removed when the OpenCPI directory you created for the installation is removed. In this way this script simply *adds* standard software to your system if it not already there.

Since different OSs use different package management systems, this script uses the appropriate commands for the OS you are running on and adds the packages required for OpenCPI development. It also uses a list prepared for each software platform as to which packages are in fact required for OpenCPI development on that platform.

Since installing packages on to your system typically requires **sudo** privileges, a password prompt usually happens when this script is run.

3.3.2 Installing Prerequisite Packages for OpenCPI

This step (#2 above) uses the script:

```
scripts/install-prerequisites.sh [-f] [<sw-platform>]
```

We use the term **prerequisite** to be software required by OpenCPI software or its built-in projects, that must be compiled specifically for OpenCPI, for all software platforms whether cross-compiled or not. These are built and used specifically for OpenCPI, in directories inside the OpenCPI installation's file hierarchy and are thus not used nor visible to other software on the system.

When this script is used for a cross-compiled platform, it first ensures that the prerequisites are also installed for the development system you are running on. It also checks whether the targeted software platform has any platform-specific prerequisites (such as a cross-compiler not otherwise needed), and builds/installs those before installing the generic prerequisites required for all software platforms.

This script checks for a timestamp indicating that all the prerequisites for a platform have been built/installed and does not do it again unless the force option (**-f**) is specified. It also recognizes when the download for each prerequisite has been done

before and does not re-download software, even if the force option (**-f**) causes it to be built/installed again.

Prerequisites are usually downloaded from the internet based on URLs for each prerequisite. If your organization does not allow this (or is not connected to the internet), and has a staging server or file share for vetted downloads, some environment variables can be used to redirect the download process to an alternative local server instead of the server indicated by the internet URL associated with the prerequisite. See the environment setup section below for these options.

3.3.3 *Building the OpenCPI Framework and its Built-in Projects*

This step (#3 and #4 above) uses the script:

```
scripts/build-opencpi.sh [<sw-platform>]
```

This building script will build:

- The core software infrastructure libraries, and utility command executables.
- The OpenCPI Linux kernel/device driver
- The software components in libraries in the built-in projects.
- Some example applications in the built-in projects.

This script first builds the OpenCPI framework executables, libraries and drivers, and then builds the software aspects of the built-in projects for the targeted software platform. It does not do any HDL (FPGA) building in the built-in projects since that depends on what FPGA tools may be available, and is thus done manually by users after installation, based on their environment.

3.3.4 *Testing the Software Aspects of the Installation*

This step (#5 above) uses the script:

```
scripts/test-opencpi.sh
```

It does not test cross-built platforms — only the installation (development) platform. Thus this step is skipped if the **<sw-platform>** argument to the top-level installation script is a cross-built platform. A variety of tests are run. One is to test loading the kernel driver, which requires **sudo** privileges and typically prompts for a password.

3.4 Configuring your Environment to Run OpenCPI

To set up your environment as a user of OpenCPI, you must use the **opencpi-setup.sh** script that is in the **cdk** subdirectory. OpenCPI currently only supports the "bash" shell. There are two ways to perform this setup step:

If you want to manually set up your environment in each shell window as you need it, you simply source the script where it lives. E.g. if OpenCPI was downloaded into the

~/opencpi

directory, you would issue the command:

```
source ~/opencpi/cdk/opencpi-setup.sh -s
```

Sourcing this script with no arguments (or the **-h** or **--help** options), will display more options for special cases. If you want to set up the environment on each login, you would add this same line to your **~/ .profile** file (or **~/ .bash_login** or **~/ .bash_profile**). Note that this will only take effect when you login, or when you start a new "login shell" using the **-l** option to bash, like:

```
bash -l
```

There are several additional environment variables that are required in some circumstances. They are usually set in your **.profile** file prior to sourcing the **opencpi-setup.sh** file.

For native execution, no such environment variables may be necessary, but when third party tools are required for other platforms, their locations and/or license file locations may need to be set this way. Such variables are described in the sections below about configuring for use of such tools.

Environment variables that can affect runtime behavior may be set either permanently in the **~/ .profile** file or simply set as needed. They are described in detail in other documents, but are summarized here also.

Table 1: Environment Variables for Runtime Configuration

Name	Description
OCPI_CDK_DIR	The variable set by sourcing the opencpi-setup.sh script indicating the location of the OpenCPI installation. This variable should not be set directly.
OCPI_LOG_LEVEL	The log level (amount of logging) output by the runtime system. The default if not set logs unusual/unexpected events to stderr .
OCPI_PROJECT_PATH	A colon-separated set of projects to be considered in addition to those that are registered.
OCPI_SYSTEM_CONFIG	The runtime system XML configuration file. Default is \$OCPI_CDK_DIR/./system.xml
OCPI_DMA_MEMORY	When the OpenCPI kernel driver is not used, this environment variable allows super-user privileged processes to specify physical DMA memory to use. The format is <mbytes>M\$0x<address> , with mbytes in decimal, and address in hexadecimal.
OCPI_SMB_SIZE	This variable specifies the size (in bytes, decimal) of the memory pool used for buffers in software containers.

The OpenCPI system configuration file is an XML file that allows additional parameters to be specified that affect the runtime system. Its location is usually **\$OCPI_CDK_DIR/./system.xml**, but its location may also be specified by the environment variable **OCPI_SYSTEM_CONFIG**. It is organized hierarchically according to the major runtime software module hierarchy. An example is:

```
<opencpi>
  <container>
    <ocl load='0' />
    <rcc load='1' />
    <hdl load='1'>
      <device name="PCI:0000:05:00.0" esn="000013C1E1F401" />
      <device name="PCI:0000:04:00.0" esn="91d28408" />
    </hdl>
    <remote load='1' />
  </container>
  <transfer smbsize='128K'>
    <pio load='1' />
    <!-- <dma load='1' /> -->
    <socket load='1' />r
  </transfer>
</opencpi>
```

The two top level categories of configurable modules are:

- **container:** for the runtime modules (drivers) that support different authoring models
- **transfer:** for the runtime modules (drivers) that support different dataplane transport mechanisms.

Each driver has a boolean **load** attribute that specifies whether the module should be loaded (and thus enabled) in the runtime environment. Otherwise all attributes are module-specific.

The **device** child element can appear under any of the container driver elements and specifies attributes specific to individual devices supporting that container type. In the example above, an **esn** attribute is applied to two HDL devices, which is used to indicate the electronic serial numbers of the JTAG cables attached to specific PCI devices.

The **smbsize** attribute to the **transfer** element specifies a default value for the size in bytes of the buffer memory pool for each dataplane endpoint. It can also be applied to each module below it which overrides the default.

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 “gateway”.

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 challenges associated with porting the OpenCPI development environment to a new platform.

If no Linux-hosted cross-tools are available for the embedded target, then the OpenCPI build/development environment must indeed 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 and linkers
- 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 (in a development context) 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 between the embedded system and the development host. This configuration typically uses **nfs**, with the embedded system acting as an **nfs** client, and the development system as the **nfs** server. If the development system is not yet configured for **nfs**, the appropriate software is installed (on CentOS/RedHat systems) using the command:

```
% sudo yum install nfs-utils
```

When network access to the development system is unavailable, inconvenient, or inappropriate, then a small subset of the OpenCPI cross-built environment is copied to the embedded system to enable execution without any network. This is called “standalone” or “embedded” mode for OpenCPI on the embedded system.

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 many libraries to be copied to the embedded system.

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

1. Install cross development tools and cross-build OpenCPI on the development system.
2. Create a bootable kit (usually an SD card) to install on the embedded system, using the development system.
3. Set up the hardware as needed for OpenCPI.
4. Install the bootable kit (usually plugging in an SD card) on the embedded system.

4.1 The Digilent ZedBoard with Xilinx Zynq SoC Processor

This hardware platform is the smallest and least expensive 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.

The steps below describe the process of building OpenCPI and preparing a bootable SD card from scratch, including assets from a Xilinx Linux kernel release as well as those built in OpenCPI. However, there is a shortcut that allows you to bypass all these steps if you just want to quickly run OpenCPI on a ZedBoard: a directory in the OpenCPI code base contains a prebuilt SD card directory that can simply be copied to a real SD card that can then be inserted into a ZedBoard before restarting it.

Assuming the SD card (at least its first partition usually entitled “BOOT”) is mounted on `/media/xyz` (or sometimes `/run/media/<username>/BOOT`), the following commands will remove its current contents and copy the OpenCPI contents onto it and unmount it so you can remove it and plug it into a ZedBoard. It is assumed that you are in the directory where the OpenCPI software platform is defined, in some project, e.g.:

```
<some-project>/rcc/platforms/<sw-platform>
```

For the ZedBoard in particular, one common software platform to use is `xilinx13_4`, which is the Xilinx-provided Linux release released shortly after the 14.7 version of the Xilinx ISE tools. This particular platform is in the OpenCPI built-in project called `core`, so from the top of the OpenCPI source tree, its path is:

```
projects/core/rcc/platforms/xilinx13_4
```

Assuming you are in this directory with the normal OpenCPI environment set up, for the `zed` hardware platform, you would do:

```
% rm -r -f /media/xyz/*
% cp -R -p release/OpenCPI-SD-zed/* /media/xyz
% umount /media/xyz
```

To use this shortcut, you still need to at least arrange for console access to the ZedBoard, as described in the section below.

After removing the SD card, plug it into a properly configured (powered off) ZedBoard, power on and run some simple applications that are already there. If this will serve your purposes (initially), you can skip the rest of this section and proceed to the section for ZedBoard hardware setup. Of course you can add any of your own files to the SD card.

Summary: you can use the pre-built SD card contents unless:

- You need to use a more up-to-date OpenCPI version.
- You need a more up-to-date or different Xilinx Linux release.

4.1.1 Establishing the Cross-Building Environment for OpenCPI targeting Zynq

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.

4.1.1.1 *Install Xilinx ISE and EDK Tools*

There are two aspects to 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 14.7, and \$XROOT might be **/opt/Xilinx**, so the current release would be installed in the

/opt/Xilinx/14.7

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. The XRN and XROOT variables are notional for this document. You do not need to actually set or use those variables. The Xilinx Vivado tools can also be used, with the compiler provided by the “SDK” add-on.

We assume a Xilinx ISE+EDK or Vivado+SDK installation on CentOS6 or CentOS7 64-bit Linux, the standard OpenCPI development host. Xilinx officially supports Red Hat Enterprise Workstation 6 (64-bit), of which CentOS/64-bit is a free “clone”, without any official support.

We also assume that you have created a clone of the OpenCPI git repository, placed it in a directory on a development system (e.g. /home/myself/opencpi), and built and run OpenCPI on that development host system. Instructions for this are above.

The OpenCPI ZedBoard installation includes certain artifacts that are derived from one or more Xilinx Linux kernel releases. Under normal circumstances these can be used as they exist in the OpenCPI tree. The **core/platforms/xilinx13_4/release** file is a link to the Xilinx Linux release (inside the OpenCPI tree) that will be used during the remaining installation steps. If you need to change or customize the Xilinx Linux kernel release, see the **platforms/zynq/README.linux** file. We also assume a successful OpenCPI development host installation (and runtime tests) as described above.

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

- A working OpenCPI installation on CentOS/64-bit
- A Xilinx ISE+EDK installation at \$XROOT/\$XRN (e.g. **/opt/Xilinx/14.7**) or a Vivado+SDK installation at \$XROOT/Vivado/\$XRN (e.g. **/opt/Xilinx/Vivado/2015.4**)

4.1.1.2 *Establish and Customize your Cross Build Environment*

To build OpenCPI for Zynq/ARM software development, we use the Xilinx-supplied ARM/Zynq cross-build tools (cross from CentOS x86_64 to Zynq). There is nothing

specific to the ZedBoard platform about these Zynq tools. You may set the OCPI_XILINX_DIR, OCPI_XILINX_VERSION, and OCPI_XILINX_LICENSE_FILE environment variables if the defaults are not correct. You do **not** need to run any Xilinx-supplied initialization scripts. Using these Xilinx ISE+EDK tools for cross-compilation has been tested in ISE 14.6 and 14.7. Using Vivado+SDK has been tested using version 2015.4 and later.

4.1.1.3 *Installing for the Zed Embedded Target.*

The same script is used to prepare OpenCPI for development targeting as was used for the development system itself. The only difference is to provide the software platform as an argument to the script. E.g. to prepare for development using the xilinx13_4 software platform you can run:

```
./scripts/install-opencpi.sh xilinx13_4
```

This builds and installs prerequisites for this (cross)platform and then builds the OpenCPI framework libraries and executables, as well as the software assets in the built-in projects that are part of OpenCPI.

This will build the Zynq versions of all OpenCPI libraries, executables, components, examples, and kernel driver for the Zynq Linux platform. This will *not* interfere with the libraries, executables and components already built for the native CentOS/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.

Building the OpenCPI kernel driver relies on the release of the Xilinx Linux kernel that is already established in the software platform's directory, e.g.:

```
projects/core/rcc/platforms/xilinx13_4
```

4.1.1.4 *Build FPGA Libraries, Components, Platform and Bitstreams*

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

```
% make hdl Hd1Platforms="isim zed"
```

This will build all the primitive libraries, components, bitstreams for ZedBoard's Zynq chip and for isim (the ISE simulator). If using the Vivado simulator, use **xsim** rather than **isim**. If you had already built OpenCPI HDL/FPGA code for other Xilinx targets, the **isim** or **xsim** 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 using ISE is **zynq_ise** and the **HdlPlatform** is **zed_ise**. Using Vivado, the target is **zynq** and the platform is **zed**. This step can take an hour or more.

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 can be plugged into the ZedBoard for booting. The contents of the SD card directory tree is a combination of:

- Binary files from a Xilinx Linux Kernel release from <http://www.wiki.xilinx.com/Zynq+Releases>
- Binary files built and stored in the OpenCPI tree that were patched from the Xilinx release and Linux source code.
- Core files from the OpenCPI tree built from sources.
- Component library files from the OpenCPI tree (RCC and HDL).

This resulting SD card can be used standalone (independent of the network) or based on a network connection that allows the **zed** platform to communication with the development system. The network-based development setup mounts the development system from the ZedBoard using NFS, so usually very few files really need to be on the SD card itself. In standalone mode, OpenCPI can be used with the files on the SD card, and with no network.

To create an SD card tree run the following from the top level of the OpenCPI project:

```
make deploy Platforms=zed:xilinx13_4
```

These instructions prepare a directory tree for an SD card locally (in `cdk/zed/zed-deploy/sdcard-xilinx13_4`), to be copied to a real SD card, that can then be unmounted, plugged into the Zed board, and used to boot the ZedBoard.

If you ran `make hdl HdIPlatforms="isim zed"` from above and want to include the `.bit.gz`'s on your SD card you will need to manually copy the `.bit.gz`'s files to `cdk/zed/zed-deploy/sdcard-xilinx13_4/opencpi/xilinx13_4/artifacts/` or if you want all the `.bit.gz`'s you can run the following:

```
find . -path "*/target-zynq/*.bit.gz" -exec cp {} cdk/zed/zed-deploy/sdcard-xilinx13_4/opencpi/xilinx13_4/artifacts/ \;
```

There are two files that must be customized before making the SD card. One for standalone mode and one for network mode (NFS mounting of the development system). If you are not using the network/NFS mode you can ignore the second one.

4.1.2.1 Standalone Mode Startup Script

You should make copy of the `default_mysetup.sh` file into `mysetup.sh` (here in the `cdk/zed/zed-deploy/sdcard-xilinx13_4` directory) and customize it, in particular, specifying at least:

- The system to be used as a time server
- Your timezone.

At least in CentOS6 and CentOS7, the following command will print what the timezone should say:

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

These items are on the command line in **mysetup.sh** that runs the internal **zynq_setup.sh** script, which configures the ZedBoard system for OpenCPI each time it boots. Since the ZedBoard has no real time clock these are needed to set the time properly. If you truly have no network connection, you can set the time server to "-" and set the time manually when the ZedBoard system is booted.

When the SD card is created, this script will be copied to it, and it is run each time the system is booted (manually for now).

```
% cp default_mysetup.sh mysetup.sh
{ customize mysetup.sh }
```

4.1.2.2 *Network Mode Startup Script*

Network mode is when you mount the development system (and the OpenCPI tree), as an NFS server with the ZedBoard as NFS client. This provides easy and dynamic access to all of OpenCPI, and presumably any components and applications you may have. You can avoid customizing this file if you only want to use standalone mode, but you must make sure the **mynetsetup.sh** exists.

Make a copy of the **default_mynetsetup.sh** file into **mynetsetup.sh** (here in the **cdk/zed/zed-deploy/sdcard-xilinx13_4** directory) 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 the OpenCPI tree lives.
- The system to be used as a time server.
- Your timezone description.

All these items are on the command line in **mynetsetup.sh** that runs the **zednetsetup.sh** script, which configures the zed system for OpenCPI each time it boots, for network/NFS mode. The network address of your development system will be specified later, dynamically, as an argument to **mynetsetup.sh**.

```
% cp default_mynetsetup.sh mynetsetup.sh
{ customize mynetsetup.sh }
```

Note the **mynetsetup.sh** script you copied to the SD card takes the development system's IP address as an argument. If it is fixed, you could hard-wire it in this script.

4.1.2.3 *Initialize (partition, erase) the SD Card*

This step completely initializes, formats, and partitions the SD card, whereas the next step copies the contents onto it. This step creates a 200MB boot partition in FAT32 format, and also creates a second linux ext2/ext3 file system using the rest of the space on the card. Because it is partitioning the drive from scratch, there must be no mounted file systems on the drive when it is executed.

If you don't want a "fresh" SD card, but want to use an existing one that is correctly formatted for a ZedBoard boot disk, you can skip this step.

You must determine the linux device name of the SD card, usually something like /dev/sdb. The "**sudo fdisk -l**" command describes all disk devices on the system and can help in identifying the device for the SD card you plugged in. Using the wrong device name could erase the disk, *so be careful!* Also, you can use the "**mount**" command to see if any of the partitions of the disk are already mounted. They should be unmounted before running this script.

The following script completely reinitializes/partitions/erases the SD card for use with the ZedBoard. The single argument is the linux disk device name. The "sudo" is usually necessary unless the device has general write permission.

```
% sudo ./platforms/zynq/formatOpenCPIZynqSD.sh /dev/sdX
```

4.1.2.4 Copy the Prepared SD File Tree to the Actual SD Card.

The source of this step is the directory created by the **make deploy** above, which is **cdk/zed/zed-deploy/sdcard-xilinx13_4**. If you have just formatted the SD card with the previous step, you would need to mount it first using the mount command, or simply unplug it and plug it in again, which normally automatically mounts it.

After the "BOOT" partition is mounted (e.g. /dev/sdb1 is mounted on /media/boot) do:

```
% rm -r -f /media/boot/*
% cp -R -L -p cdk/zed/zed-deploy/sdcard-xilinx13_4/* /media/boot
```

If you have more files to add, do so now, then remove it using:

```
% umount /media/boot
```

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.

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

Assuming you not taking the shortcut mentioned at the top of this section:

From the top level directory of an OpenCPI source code installation a.k.a. **OCPI_ROOT_DIR**, that is already functional for the CentOS system, with a Xilinx ISE+EDK installation, in a fresh/new shell/window, do these steps:

```
% cp example_zed_env.sh myzed.sh
{ customize myzed.sh for your build environment, e.g. where Xilinx
tools are }
{ customize the Xilinx Linux release if needed - see README.linux if
you need to}
% source myzed.sh
% ./scripts/install-opencpi.sh xilinx13_4
% make hd1 Hd1Platforms='isim zed'
```

```

% make deploy Platform='zed:xilinx13_4'
% cd cdk/zed/zed-deploy/sdcard-xilinx13_4/opencpi
% cp default_setup.sh mysetup.sh
% cp default_netsetup.sh mynetsetup.sh
{ customize mysetup.sh and mynetsetup.sh for your environment,
etc. }
% cd ../../../../../../..
{ If you need a freshly formatted SD card:
  plug your SD card in and figure out what the disk device name is
  for it.
  unmount any mounted partitions on the drive/disk
  BE CAREFUL TO GET IT RIGHT SINCE THAT DISK WILL BE ERASED
  % sudo ./platforms/zynq/formatOpenCPIZynqSD.sh /dev/sdX
}
{ Plug in the SD card, e.g. mounted as /media/xyz }
% rm -r -f /media/xyz/*
% cp -r -L -p cdk/zed/zed-deploy/sdcard-xilinx13_4 /media/xyz
% umount /media/xyz

```

Now everything is built and ready for the ZedBoard, and 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. There are two usage modes, one is network-based, where the ZedBoard can talk to the development (CentOS) system via network and NFS, and the other is standalone/embedded mode where OpenCPI can be used even without a network connection.

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

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 the 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)
- JP6: shorted (only needed for "CES" silicon versions)
- JP2: shorted (needed to power USB peripherals or dongles).JP2: shorted
- JP6: shorted (*note: we didn't have enough jumpers and this wasn't needed*)

All other jumpers should be left unshorted. The power switch should be off.

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 baud, 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.

4.1.3.5 *Get Access to the Serial Console of the ZedBoard, and Reboot from There*

Since the OpenCPI development environment is running on a Linux system (usually CentOS6 or 7), 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 ZedBoard 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 ZedBoard 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. To eliminate this step you can add a *udev* rule to your system for the `/dev/ttyXXX` in question by issuing this command:

```
% echo 'KERNEL="ttyACM0", MODE="0666"' | \
sudo dd of=/etc/udev/rules.d/99-opencpi-tty.rules
```

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 on the ZedBoard. 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 CentOS 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 this 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 via the network 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. The **ifconfig** command on the ZedBoard will print the internet address.

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*

If you want to start in standalone mode, without needing a network or connection to the development host, skip to Standalone Mode section below.

To enable the network connection to the development host, connect the ethernet connector on the ZedBoard 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.

You can also use USB network dongles, with the appropriate cable adapters, plugged into the **USB OTG** connector.

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 (or "**eth1**" for USB Ethernet dongles) has an internet address, courtesy of DHCP. E.g. the output below 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" documents 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 ZedBoard 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 ZedBoard 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 also mounts the SD card itself as a file system, at **/mnt/card**. This means that anything you write to the root file system is *not* persistent. To create any persistent file you use SD file system, at **/mnt/card**.

Our standard setup uses the built-in NFS client on the ZedBoard to access the OpenCPI source code base from these types of systems (in a development environment).

After booting, and logging in as root (either via the serial console or ssh), the first command should be to **source** the startup script on the SD card, supplying the development host's IP address as an argument.

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

This command, 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 this commands is 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 **mynetsetup.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. In standalone mode, it changes to the **/mnt/card/opencpi/xml** directory.

To run a simple test application:

```
zynq> ocpirun -v -d -mbias=rcc testbias
```

This should run the **testbias** application (using file_read->bias->file_write) 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 in both network and standalone mode), you can run:

```
zynq> ocpirun -v -d -mbias=hd1 testbias
```

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

4.1.3.11 *Standalone Mode*

To run in standalone mode, which has no requirement of a network, but can still use one if present, you run, after booting, the following command:

```
# source /mnt/card/opencpi/mysetup.sh
```

This will set up the environment to run some applications locally based on files in the SD card, without accessing any files on the network.

The **opencpi** subdirectory on the SD card has executables and scripts in the "**bin**" directory, component artifacts in the "**artifacts**" directory, and xml applications in the "**xml**" directory.

The steps to run OpenCPI applications without and with FPGA components are in the previous two sections and work in standalone mode as well as network mode.

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.

Each PCI 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 specific 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.

Some boards only re-load the bitstream from flash memory on power cycling, while others can/will also do it on system (PCI fabric) reset.

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 multiple USB ports, 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 OpenCPI 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 a system. The `lspci` Linux utility can be used to list PCI-Express slots and information about the cards that are plugged into them.

This editing of the `system.xml` file 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

OpenCPI FPGA 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 “testbias” 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/testbias_ml605_base.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 then not be the “.bitz” file but rather a vendor-specific file probably 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 automatic 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, power cycle 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 B of the Xilinx document. 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 JTAG/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. As mentioned earlier, use the loadFlash command to load the default bitstream into flash memory:

```
% loadFlash ml605 hdl/platforms/ml605/testbias_ml605_base.bitz \  
  <esn-for-jtag>
```

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 HDL Platform

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 “bitstreams” 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 are a few additional steps 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 and CentOS7 are 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 may 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. 14.7)
- **OCPI_XILINX_LICENSE_FILE** 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 an environment setup script for using Xilinx ISE is:

```
export OCPI_XILINX_DIR=/home/user1/Xilinx
export OCPI_XILINX_VERSION=14.7
export OCPI_XILINX_LICENSE_FILE=/home/user1/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.