OpenCPI

ZedBoard Getting Started Guide

Version 1.5

Revision History

Revision	Description of Change	Date
v1.1	Initial Release	3/2017
v1.2	Updated for OpenCPI Release 1.2	8/2017
v1.3	Updated for OpenCPI Release 1.3	2/2018
pre-v1.4	Fixed inaccurate description for hardware jumper configuration, OpenCPI-SD-zed directory path, and MAC address modification instructions for multiple ZedBoards on the same network.	4/2018
v1.4	Update descriptions and paths	9/2018
v1.5	Deprecated Zipper	4/2019

Table of Contents

1 References

This document assumes a basic understanding of the Linux command line (or "shell") environment. The reference(s) in Table 1 can be used as an overview of OpenCPI and may prove useful.

Title	Link
Acronyms and Definitions	Acronyms_and_Definitions.pdf
Getting Started	Getting_Started.pdf
Installation Guide	RPM_Installation_Guide.pdf

Table 1: References

2 Overview

This document provides: steps for configuring a factory provided Digilent Zedboard with the OpenCPI run-time environment for executing applications, configuring a development system to build OpenCPI bitstreams targeting the *zed* platform, and examples of executing applications on the OpenCPI configured Zedboard.

3 Prerequisites

3.1 RCC Platforms

The RCC platforms xilinx13_3 and xilinx13_4 are both supported. When targeting xilinx13_4, replace 13_3 in directory names and other places where 13_3 is used or applicable. Boards such as the E310 and ZedBoard use the same xilinx13_4 RCC platform. If it is already installed and built it can apply to the ZedBoard.

3.2 RPM

This guide assumes that, at a minimum, the following RPMs are installed:

RPM Name	Description
All prerequisite RPMs	These packages have OpenCPI-specific patches and are provided as RPMs. This packaging ensures they will not conflict with other installed copies by using a nonstandard installation location of `/opt/opencpi/prerequisites`.
`angryviper-ide-*.x86 64.rpm`	The ANGRYVIPER IDE (Eclipse with plugins). See RPM Installation Guide.pdf, Appendix D for an alternative method to set up the IDE using an existing Eclipse installation.
`opencpi-*.x86_64.rpm`	Base installation RPM includes the runtime portion of the Component Development Kit (CDK) and the source for the ocpi.core and ocpi.assets Projects containing framework essential components, workers, platforms, etc.
`opencpi-devel-*.x86_64.rpm`	Additional header files and scripts for developing new assets as HDL and/or RCC.

`opencpi-sw-platform-xilinx13_3-*.noarch.rpm`
or
`opencpi-sw-platform-xilinx13_4-*.noarch.rpm`
	Additional files necessary to build the framework targeting specific RCC/software platforms, independent of the final deployed hardware.

`opencpi-hw-platform-zed-xilinx13_3-*.noarch.rpm`
or
`opencpi-hw-platform-zed-xilinx13_4-*.noarch.rpm`
	Additional files necessary to build the framework targeting specific hardware platform "X" when running RCC platform "Y" ("Y" can be "no sw"). This RPM also includes hardware-specific SD Card images when applicable.

3.3 Installation of provided OpenCPI projects: *core* and *assets*

This guide assumes the user has executed *ocpi-copy-projects*, accepting the default settings, to copy and register the *core* and *assets* projects from the /opt/opencpi/projects for building bitstreams for the Zedboard. Reference the Getting Started Guide for details on *ocpi-copy-projects*. While registering of the projects is performed during the execution of ocpi-copy-projects, changes to the registry can be made via `ocpidev un/register project` or the ANGRYVIPER GUI.

```
$ ocpi-copy-projects
...
$ ls ~/ocpi_projects
assets core
```

```
$ ocpidev show registry
Project registry is located at: /opt/opencpi/cdk/../project-registry
-----------------------------------------------------------------------------------------
| Project Package-ID  | Path to Project                                 | Valid/Exists |
| ------------------  | ---------------                                 | ------------ |
| ocpi.core           | /home/user/ocpi_projects/core                   | True         |
| ocpi.assets         | /home/user/ocpi_projects/assets                 | True         |
-----------------------------------------------------------------------------------------
```

3.4 Vendor Software Setup

The platform that is expected to be used is the Digilent Zedboard (*e.g.* zed). This OpenCPI-enabled platform provides the capability of deploying hardware and software workers while using Xilinx's 13.3 distribution of Linux.

The synthesizers and cross-compilers required to build HDL and RCC Workers for this platform are installed by following the instructions found in the *OpenCPI FPGA Vendor Tools Installation Guide*. This document assumes that the user has installed the appropriate versions of Vivado and the Xilinx SDK.

3.5 Building OpenCPI projects: *core* and *assets*

The *core* and *assets* projects must be built *in a specific order* for this platform. This section outlines how to build the relevant projects and provides the commands to do so.

For this document, the projects should be built as follows:

1. Build `core` for the `xilinx13_3` RCC Platform and the `zed` HDL Platform (approx 30 min)
2. Build `assets` for the `xilinx13_3` RCC Platform and the `zed` HDL Platform, but omit assemblies (approx 45 min)
3. Build the `testbias` assembly from the `assets` project. This will be used later in this guide. (approx 10 min)

```
$ cd /home/<user>/ocpi_projects/
$ ocpidev build -d core --rcc-platform xilinx13_3 --hdl-platform zed
$ ocpidev build -d assets --rcc-platform xilinx13_3 --hdl-platform zed --no-assemblies
$ ocpidev build -d assets hdl assembly testbias --hdl-platform zed
```

Note: replace "`<user>`" with your username in the commands above.

Each of these build commands can also be performed via the ANGRYVIPER IDE as follows:

> *To perform this operation within the IDE:*
>
> 1. Open the ANGRYVIPER Perspective
> 2. Select the asset from OpenCPI Project View
> 3. Import to ANGRYVIPER Operations Panel using ">" button
> 4. Select the RCC and/or HDL platforms for the build (use Ctrl for multiple selection)
> 5. Click "Build"

See the ANGRYVIPER Team's Getting Started Guide for additional information concerning the use of `ocpidev` and the ANGRYVIPER IDE to build OpenCPI assets.

3.6 Hardware Setup

- **Digilent Zedboard**

  It is expected that this evaluation board includes a power supply, micro-USB to USB cable, micro-USB to female-USB adapter and standard SD card (4GB).

OpenCPI has been tested on revisions C and D of the Zedboard.

The micro-USB serial port located on the top-side of the ZedBoard labeled UART, can be used to access the serial connection with the processor.

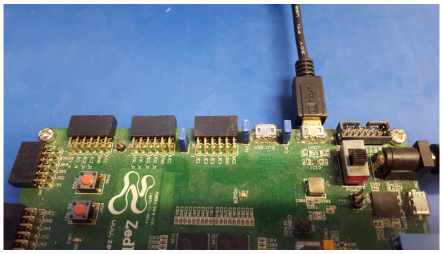


Figure 1: Connected Serial USB

Below the FMC LPC slot (bottom-side of the Zedboard), is the SD card slot which will be used throughout this guide.

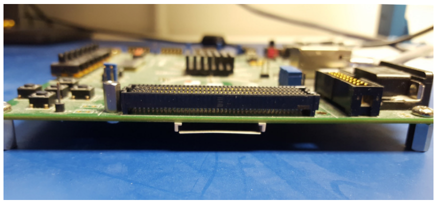


Figure 2: ZedBoard FMC Slot and SD card Slot

- **Ethernet cable**: An Ethernet port is available on the Zedboard and is required when the Network mode (discussed later) environment is used. The OpenCPI BSP for the ZedBoard is configured for DHCP.

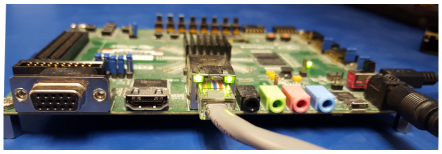


Figure 3: Connected Ethernet

- **OpenCPI Zedboard BSP supported daughter cards (OPTIONAL)**
  The ZedBoard has a FMC LPC slot that is used to connect plug-in modules or daughter cards. Currently, OpenCPI supports two FMC daughter cards, which may be installed on the Zedboard:
  - Analog Devices FMCOMMS2
  - Analog Devices FMCOMMS3
- **Access to a network which supports DHCP. (Network Mode)**
- **SD card reader**

4 SD Card Setup

4.1 Make a backup image of factory SD card (assumes Linux host)

This section provides the steps for creating an SD card backup image. It is optional, because the factory provided SD card does not have special formatting or content that must be preserved, unlike other systems (Epiq Solutions Matchstiq-Z1) that have been enabled for OpenCPI. The subsequent subsections assume the SD card is empty.

- Determine the device file name for the SD card by executing dmesg command below. It will likely be something like `/dev/sdb` or `/dev/mmcblk0`.
  ```
  $ dmesg | tail -n 15
  ```

- Run the following dd command to make a backup image, where DEVICENAME was determined above. This step should take ∼ 15 minutes depending on the card size.
  ```
  $ dd if=DEVICENAME of=backup.image
  ```

To restore the card back to the original contents, run the command “`dd of=DEVICENAME if=backup.image`”

4.2 Format the SD card

- If the user requires the SD card to be formatted, use a single FAT32 partition.

4.3 Copy embedded OS and boot files to SD card

WARNING: The user must ensure that the contents of the SD, match the version of the OpenCPI release that the artifacts were built against.

When using the factory SD card (with the proper formatting), all files can be ignored or deleted. Any files/directories copied to the SD card will appear at `/mnt/card` on the Zed.

Copy the following files/directories onto the SD card:

```
$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/boot.bin /run/media/<user>/<partition>/

$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/devicetree.dtb /run/media/<user>/<partition>/

$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/uImage /run/media/<user>/<partition>/

$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/uramdisk.image.gz /run/media/<user>/<partition>/
```

4.4 Copy files to SD card for desired Mode(s)

As previously discussed, Standalone and Network modes offer trade-offs for configuring the run-time environment of the platform. The following sections provide instructions for copying specific files/directories to the SD card in support of these modes. For maximum flexibility and completion of this getting started guide, it is recommended that the SD card be configured to support both modes, as covered in the next sub-section. However, instructions for configuring the SD card for each mode separately, have also been provided.

4.4.1 Standalone and Network Modes

The SD can be setup to support both modes, as there is no conflict between the files/directories for either mode. To setup the SD to support both modes:

After performing the steps from 4.3, copy the entire *opencpi* directory to the SD card.

```
$ cp -rL /opt/opencpi/cdk/zed/sdcard-xilinx13_3/opencpi /run/media/<user>/<partition>/

$ cp /home/<user>/ocpi_projects/assets/hdl/assemblies/testbias/container-testbias_zed_base/\
target-zynq/testbias_zed_base.bit.gz /run/media/<user>/<partition>/opencpi/xilinx13_3/artifacts/
```

4.4.2 Standalone Mode

After performing the steps from 4.3, copy the entire *opencpi* directory to the SD card, then copy the relevant bitstreams, artifacts into the *artifacts* directory and application XMLs into the *applications* directory. For this getting started guide, only one bitstream is required to be copied onto the SD cards, where as the required artifacts and application XML where copied to the SD along with the entire *opencpi* directory.

```
$ cp -rL /opt/opencpi/cdk/zed/sdcard-xilinx13_3/opencpi /run/media/<user>/<partition>/

$ cp /home/<user>/ocpi_projects/assets/hdl/assemblies/testbias/container-testbias_zed_base/\
target-zynq/testbias_zed_base.bit.gz /run/media/<user>/<partition>/opencpi/xilinx13_3/artifacts/
```

4.4.3 Network Mode

After performing the steps from 4.3, create a directory on the partition named `opencpi` and copy the following files into the this directory:

```
$ mkdir /run/media/<user>/<partition>/opencpi

$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/opencpi/default_mynetsetup.sh \
/run/media/<user>/<partition>/opencpi/

$ cp /opt/opencpi/cdk/zed/sdcard-xilinx13_3/opencpi/zynq_net_setup.sh \
/run/media/<user>/<partition>/opencpi/
```

4.5 SD Card Source

The final SD Card artifacts are distributed in `/opt/opencpi/cdk/zed/` via RPM as noted previously. The end user is not required nor expected to generate the files.

5 Script Setup

There are two type of setups or modes for running applications on any embedded radio: Network and Standalone. In Network mode, a development system hosts the OpenCPI tree as an NFS server to the Zedboard which is an NFS client. This configuration provides quick and dynamic access to all of OpenCPI, and presumably any applications, components and bitstreams. In Standalone mode, all the artifacts are located on the SDR's local storage (*e.g.* SD card) and no network connection is required. This may be more suited for *deployment* scenarios in which network connection is not possible or practical. Network mode is generally preferred during the development process.

5.1 Setting up the Network and Standalone Mode scripts

For each mode, a startup script is used to configure the environment of the embedded system. The OpenCPI framework provides a default script for each mode. The default scripts are to be copied and modified per the user's requirements.

5.1.1 Network Mode

1) Make a copy of the default script for editing.

```
$ cp /run/media/<user>/<partition>/opencpi/default_mynetsetup.sh \
/run/media/<user>/<partition>/opencpi/mynetsetup.sh
```

2) Edit the copy

1. In `mynetsetup.sh`, uncomment the following lines which are necessary for mounting *core* and *assets* project:

   ```
   mkdir -p /mnt/ocpi_core
   mount -t nfs -o udp,nolock,soft,intr $1:/home/user/ocpi_projects/core /mnt/ocpi_core
   mkdir -p /mnt/ocpi_assets
   mount -t nfs -o udp,nolock,soft,intr $1:/home/user/ocpi_projects/assets /mnt/ocpi_assets
   ```

2. Edit `/home/user/ocpi_projects/core` and `/home/user/ocpi_projects/assets` to reflect the paths to the *core* and *assets* project on the host, e.g.:

   ```
   mkdir -p /mnt/ocpi_core
   mount -t nfs -o udp,nolock,soft,intr $1:/home/johndoe/ocpi_projects/core /mnt/ocpi_core
   mkdir -p /mnt/ocpi_assets
   mount -t nfs -o udp,nolock,soft,intr $1:/home/johndoe/ocpi_projects/assets /mnt/ocpi_assets
   ```

5.1.2 Standalone Mode

In this mode, all OpenCPI artifacts that are required to run any application on the Zedboard must be copied onto the SD card. Building the provided projects to obtain such artifacts is discussed in Section 3.5. Once the artifacts have been created, they must be copied to the SD card in Section 4. In general, any required `.so` (RCC workers), `.bit.gz` (hdl assemblies), and application XMLs or executables must be copied to the ATLAS partition of the SD card.

1) Make a copy of the default script for editing

```
$ cp /run/media/<user>/<partition>/opencpi/default_mynetsetup.sh \
/run/media/<user>/<partition>/opencpi/mynetsetup.sh
```

2) Edit the copy

Unlike Network mode, there is no required modifications to this script.

3) Copy any additional artifacts to SD card's `opencpi/xilinx13_3/artifacts/` directory

5.2 Setup system time reference

If Linux system time is not required to be accurate, this step may be skipped.

For either Network or Standalone mode, the following settings that are passed by `my[net]setup.sh` to the `zynq_[net_]setup.sh` scripts *may* require modification:

- Identify the system that is to be used as a time server, where the default is "time.nist.gov" and is set in `/mnt/card/opencpi/ntp.conf`. A valid time server must support NTP.
- Identify the current timezone description, where the default is "EST5EDT,M3.2.0,M11.1.0". Change this if required for the local timezone. See `man tzset` on the host PC for more information.
- If a time server is not required, or cannot connect to a time server, the user is required to manually set the time at radio start-up. Use the `date` command to manually set the Linux system time. See `man date` on the host PC for more information.

5.3 "rsync" provided binary

An ARM-compiled version of "`rsync`" is provided in the included SD card image for xilinx13_3. This tool allows the use of *standalone mode* while shortening the required developer time to synchronize the artifacts being developed. For command-line usage, see the rsync home page. The easiest usage is to have the radio "pull" from the developer's workstation; this does not need any additional command-line arguments.

Implementation Details

Unfortunately, the "`rsync`" executable is not in the default path because when called remotely, it requests a non-interactive shell. For this reason, a "pull" approach is recommended. If a user for some reason requires a "push" from the workstation to the radio, the local "`rsync`" executable must be told the *remote location* of the `rsync` executable to call, *e.g.* `rsync -rsync-path=/mnt/card/opencpi/xilinx13_3/bin/rsync`

5.4 Multiple ZedBoards on the same network

If it is required that multiple ZedBoards are to be on the same network, the following change to the zynq startup scripts is required. This is necessary because by default the ZedBoards have the same MAC address from the factory. To resolve this, uncomment the following lines in the mynetsetup.sh and/or mysetup.sh scripts and modify the Ethernet address to be unique:

```
# ifconfig eth0 down
# ifconfig eth0 hw ether <unique MAC address> # e.g. ifconfig eth0 hw ether 00:0a:35:00:01:24
# ifconfig eth0 up
# udhcpc
```

6 Hardware Setup

6.1 Establish a Serial Connection

By default, the USB to Serial adapter connects as read-only, which requires sudo privileges for establishing a serial connection. OpenCPI recognizes that sudo may not be available and has provided an alternative for configuring the device thereby allowing all users access to the device. Specifically, this is accomplished by adding `udev` rules to instruct the device connection to have read and write permissions for all users.

- If OpenCPI was installed via RPMs, the `udev` rules are automatically setup for the user.
- If OpenCPI was installed from source, then the user must manually add the `udev` rules by copying the file from the host machine's installation directory to the host machine's `/etc/udev/rules.d/`. The following command can be used as a guide:

```
$ cd /etc/udev/rules.d/
$ sudo ln -s /<install-path>/opencpi/cdk/zed/host-udev-rules/98-zedboard.rules 98-zedboard.rules
```

- Whether installed via RPMs or source (and manually creating the symbolic link), the USB to Serial adapter will be connected as `/dev/zed0` with read and write permissions for all users.

Once the Zedboard is powered on and micro-USB cable is connected UART to the development host, use the following command to connect to the serial port:

```
$ screen /dev/zed0 115200
```

6.2 Booting the ZedBoard from the SD card

1. Remove power from the ZedBoard unit.
2. Ensure jumpers are configured correctly
   (a) To boot from the SD card, jumpers JP10, JP9, and JP8 need to be set to 3.3V, 3.3V, and GND respectively as shown below.
   (b) The only supported FMC voltage for OpenCPI Zedboard FPGA bitstreams is 2.5 V. To ensure property FMC configuration, the VADJ SELECT (J18) jumper must be set to 2V5.
3. Insert the SD card into the SD card slot.
4. Connect a terminal to the micro-USB connector labelled 'UART' on the ZedBoard. The baud rate should be 115200 baud.
   - per the previous section, "`screen /dev/zed0 115200`" can be used to connect to the serial port.
5. Apply power to the ZedBoard with the terminal still connected.

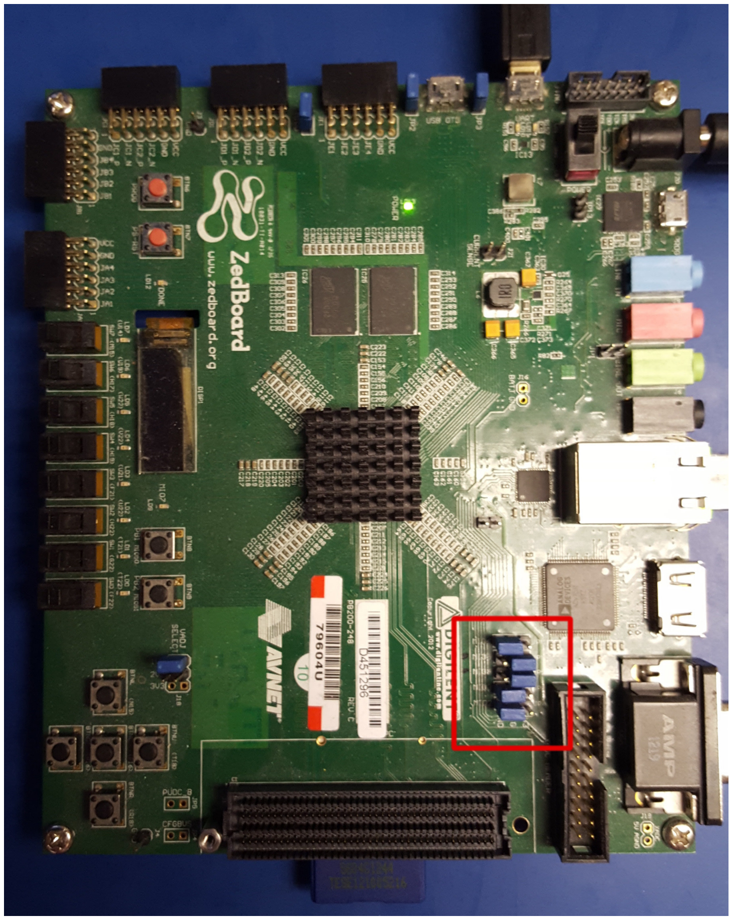


Figure 4: Top View of the ZedBoard with J10, J9, J8 Set

7 Development Host Setup - Network Mode ONLY

7.1 Network Mounting Mode

The NFS server needs to be enabled on the host in order to run the SDR in Network Mode. The following sections are directions on how to do this for both CentOS 6 and CentOS 7 host operating systems.

7.1.1 CentOS 6

From the host, install the necessary tools using yum:

```
% sudo yum install nfs-utils nfs-utils-lib
% sudo chkconfig nfs on
% sudo service rpcbind start
% sudo service nfs start
```

From the host, add the following lines to the bottom of `/etc/exports` and change "XX.XX.XX.XX/MM" to a valid netmask for the DHCP range that the SDR will be set to for your network (*e.g.* `192.168.0.0/16`). This should be as "tight" as possible for security reasons. **Do *not* share out your top-level directory! This would allow theft of your private "ssh" keys, etc!**

```
% sudo vi /etc/exports
```

```
/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
<host core project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
<host assets project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
<host assets_ts project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
```

```
% sudo exportfs -av
```

From the host, restart the services that have modified for the changes to take effect:

```
% sudo service nfs start
```

7.1.2 CentOS 7

From the host, install the necessary tools using yum:

```
% sudo yum install nfs-utils
```
[1]

From the host, allow NFS past SELinux[2]:

```
% sudo setsebool -P nfs_export_all_rw 1
% sudo setsebool -P use_nfs_home_dirs 1
```

From the host, allow NFS past the firewall:

```
% sudo firewall-cmd --permanent --zone=public --add-service=nfs
% sudo firewall-cmd --permanent --zone=public --add-port=2049/udp
% sudo firewall-cmd --permanent --zone=public --add-service=mountd
% sudo firewall-cmd --permanent --zone=public --add-service=rpc-bind
% sudo firewall-cmd --reload
```

Define the export by creating a new file that has the extension "`exports`". If it does not have that extension, it will be ignored. Add the following lines to that file and replace "XX.XX.XX.XX/MM" with a valid netmask for the DHCP range that the SDR will be set to for your network (*e.g.* `192.168.0.0/16`). This should be as "tight" as possible for security reasons. **Do *not* share out your top-level directory! This would allow theft of your private "ssh" keys, etc!**

```
% sudo vi /etc/exports.d/user_ocpi.exports

/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
<host core project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
<host assets project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
<host assets_ts project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
```

If the file system that you are mounting is XFS, then each mount needs to have a unique `fsid` defined. Instead, use:

```
% sudo vi /etc/exports.d/user_ocpi.exports

/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=33)
<host core project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=34)
<host assets project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=35)
<host assets_ts project location> XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=36)
```

Restart the services that have modified for the changes to take effect:

```
% sudo systemctl enable rpcbind
% sudo systemctl enable nfs-server
% sudo systemctl enable nfs-lock
% sudo systemctl enable nfs-idmap
% sudo systemctl restart rpcbind
% sudo systemctl restart nfs-server
% sudo systemctl restart nfs-lock
% sudo systemctl restart nfs-idmap
```

* Note: Some of the "enable" commands may fail based on your package selection, but should not cause any problems.

[1] `nfs-utils-lib` was rolled into `nfs-utils` starting with CentOS 7.2, if using eariler versions of CentOS 7, `nfs-utils-lib` will need to be explicitly installed

[2] You can use `getsebool` to see if these values are already set before attempting to set them. Some security tools may interpret the change attempt as a system attack.

8 Configuring the run-time environment on the platform

8.1 Network Mode

1. Ensure the Ethernet cable is plugged in and connected to a network configured for DHCP.
2. Ensure a micro-USB to USB cable is connected between the Zed's serial port and development host.
3. Apply power to the Zedboard
4. Use a serial terminal application to establish a serial connection, for example:

   ```
   $ screen /dev/zed0 115200
   ```

5. Typically, upon the initial power-on of the platform, the boot sequence will stop at the uboot configuration prompt. When this occurs, simply enter *boot* to allow the boot sequence to continue:

   ```
   $ zynq-uboot> boot
   ```

6. After a successful boot to PetaLinux, login to the system, using "**root**" for user name and password.

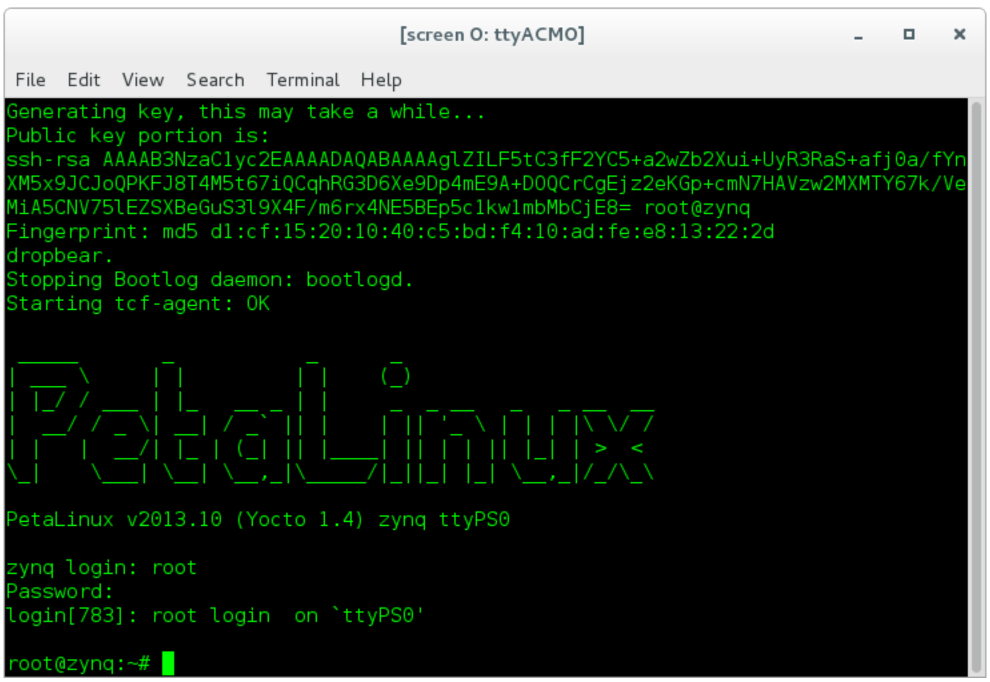


Figure 5: Successful Boot to PetaLinux

7. (a) When a **single** Zedboard is on the network, execute the following command to enable its Ethernet interface:

   ```
   $ ifconfig eth0 up
   ```

   (b) When **multiple** Zedboards are on the network, the mynsetsetup.sh script **MUST** be modified according to 5.4 prior to proceeding to the next step, in order to prevent network collisions due to multiple Zedboards having the same MAC address.

8. Setup the OpenCPI environment on remote system

   Each time the SDR is booted, the OpenCPI environment must be setup. By sourcing the `mynetsetup.sh` script, the remote system's environment is configured for OpenCPI and NFS directories are mounted for Network mode.[3]. The user must provide the network address of the development system to the script as its only argument:

[3]This script calls the `zynq_net_setup.sh` script, which should not be modifiable by the user.

```
$ source /mnt/card/opencpi/mynetsetup.sh XX.XX.XX.XX
```

where XX.XX.XX.XX is the IP address of the NFS host (i.e. that development host, *e.g.* 192.168.1.10). A successful run is shown in Figure 6.

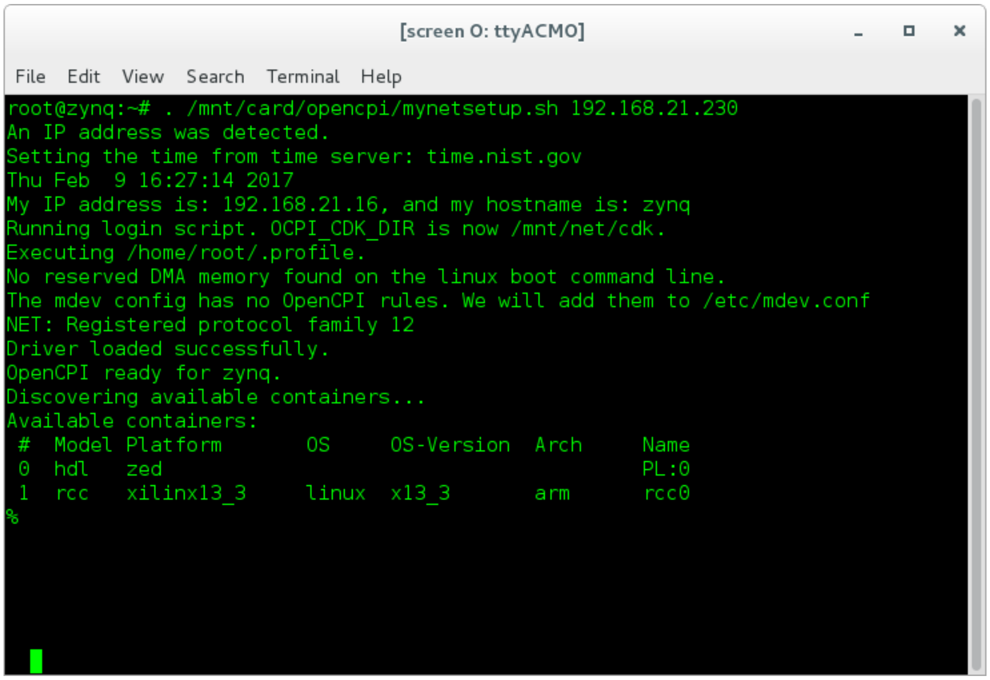


Figure 6: Successful Network Mode Setup

Note: If the output includes:

```
Attempting to set the time from time server
Alarm clock
```

`ntp` was unable to set time using servers in `ntp.conf`. For more information see Section 5.2

8.2 Standalone Mode

All artifacts (.so, .bit.gz) for any applications or tests that need to be located on the SD card must be in the `opencpi/xilinx13_3/artifacts` folder. All of the helper utilities such as `ocpirun` and `ocpihdl` are already located on the SD card and do not need to be copied over to the ZedBoard platform.

1. (Not required for OpenCPI in this mode) Plug in an Ethernet cable to a network configured for DHCP.
2. Ensure a micro-USB to USB cable is connected between the Zedboard's serial port and development host.
3. Apply power to the Zedboard
4. Use a serial terminal application to establish a serial connection, for example:

   ```
   $ screen /dev/zed0 115200
   ```

5. After a successful boot to PetaLinux, login to the system, using "**root**" for user name and password.

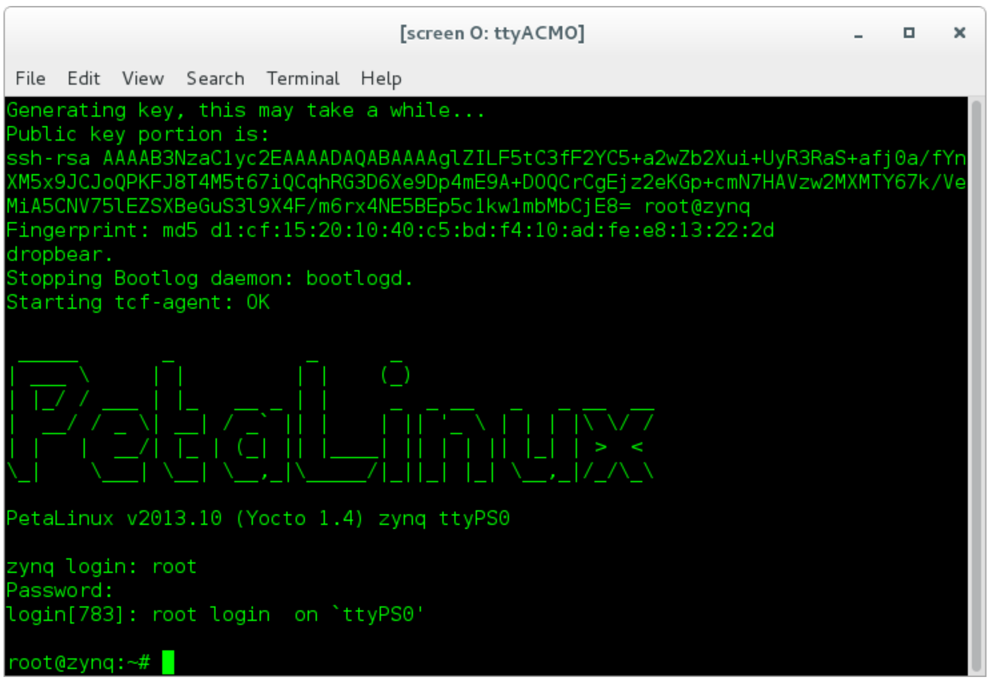


Figure 7: Successful Boot

6. WARNING: Applications (including XML-only ones) fail if there is not an IP address assigned to the platform, even when in "standalone mode." When the Ethernet port is not connected to a network configured with DHCP, a temporary IP address must be set:

   ```
   $ ifconfig eth0 192.168.244.244
   ```

7. Setup the OpenCPI environment on remote system

   Each time the SDR is booted, the OpenCPI environment must be setup. By sourcing the `mysetup.sh` script, the remote system's environment is configured for OpenCPI.[4]. There are no arguments required for this script.

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

[4] This script calls the `zynq_setup.sh` script, which should not be modifiable by the user.

A successful setup of the platform will look as follows:

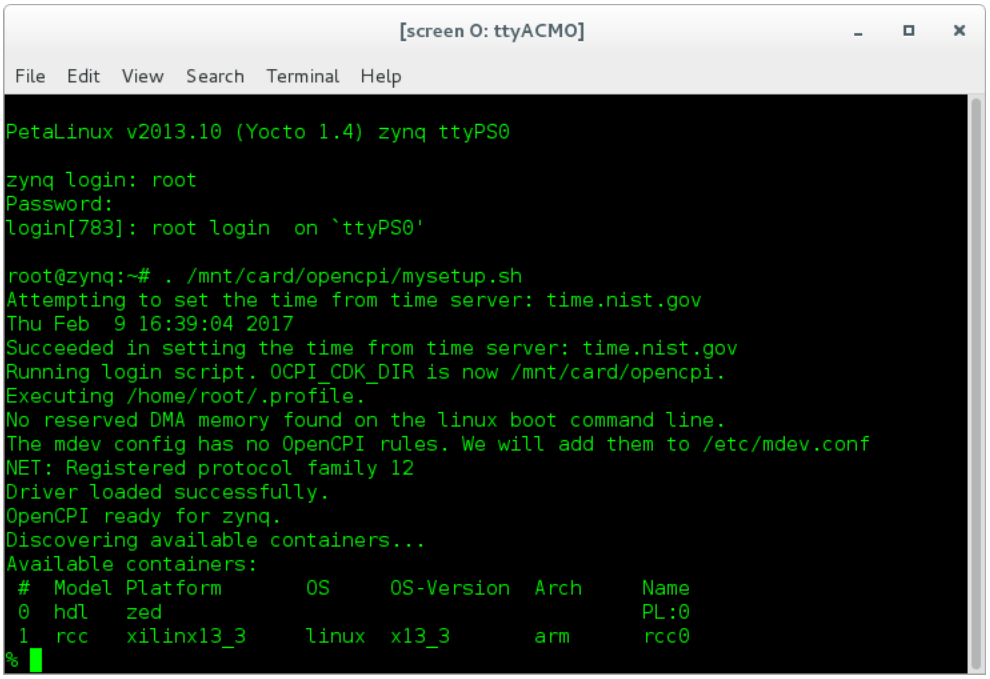


Figure 8: Successful Standalone Mode Setup

Note: If the output includes:

```
Attempting to set the time from time server
Alarm clock
```

`ntp` was unable to set time using servers in `ntp.conf`. For more information see Section 5.2

9 Build an Application

The setup of the platform can be verified by running an application that uses both RCC and HDL workers. A simple application that requires two RCC and one HDL worker is located in `assets/applications/bias.xml`, but only the RCC artifacts are provided with the installation of RPMs, and are availble on the SD card (Standard Mode) or mounted CDK directory (Network Mode). The remaining task is to build an assembly, or bitstream for loading the FPGA, which contains the HDL worker.

10 Run an Application

10.1 Network Mode

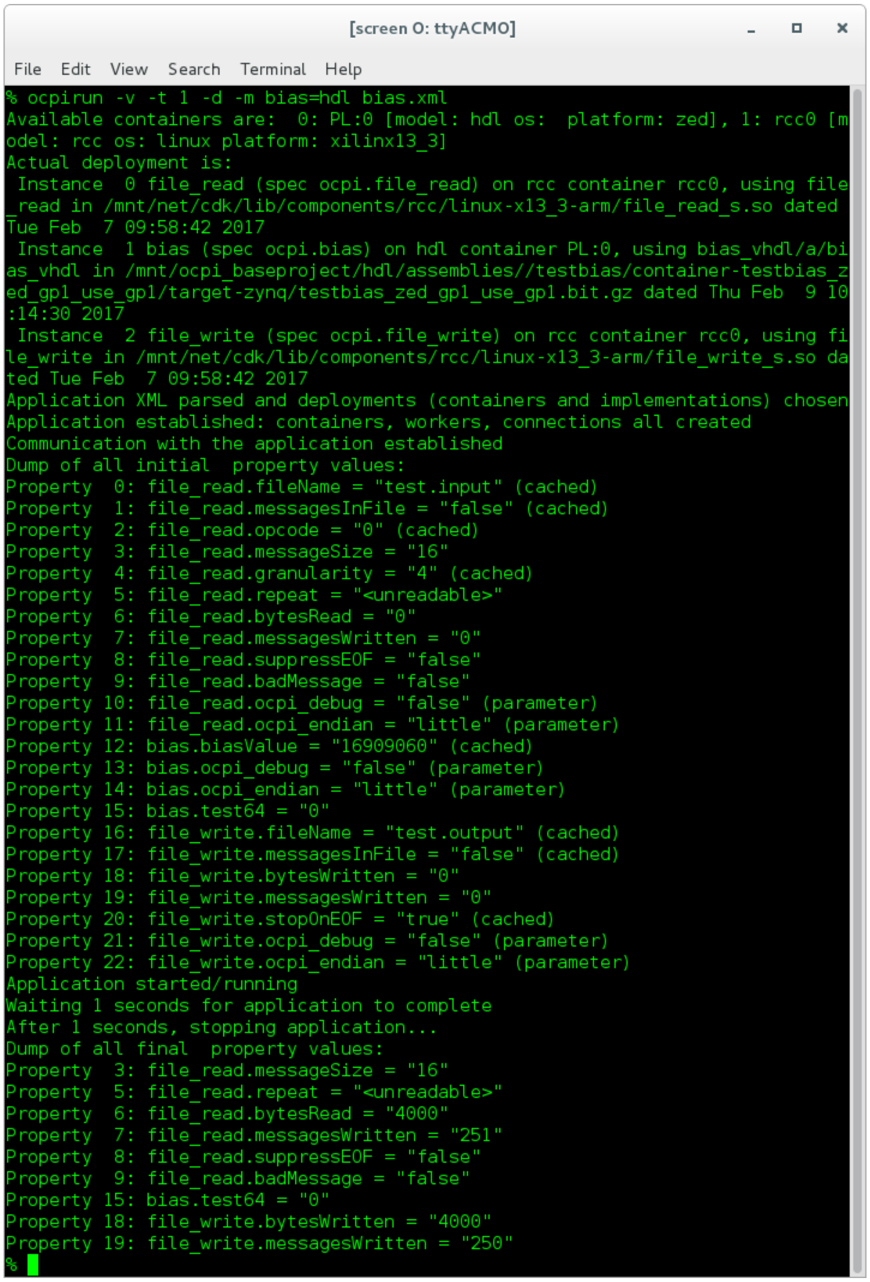
The default setup script sets the `OCPI_LIBRARY_PATH` variable to include the RCC workers that are required to execute the application, but it must be updated to include to the assembly bitstream that was built. After running the `mynetsetup.sh` script, navigate to `/mnt/ocpi_assets/applications`, then update the `OCPI_LIBRARY_PATH` variable:

```
$ cd /mnt/ocpi_assets/applications
$ export OCPI_LIBRARY_PATH=$OCPI_LIBRARY_PATH:/mnt/ocpi_assets/artifacts
```

Run the application using the following command:

```
$ ocpirun -v -t 1 -d -m bias=hdl bias.xml
```

The output should be similar to Figure 9:




```
% ocpirun -v -t 1 -d -m bias=hdl bias.xml
Available containers are:  0: PL:0 [model: hdl os:  platform: zed], 1: rcc0 [m
odel: rcc os: linux platform: xilinx13_3]
Actual deployment is:
 Instance  0 file_read (spec ocpi.file_read) on rcc container rcc0, using file
_read in /mnt/net/cdk/lib/components/rcc/linux-x13_3-arm/file_read_s.so dated
Tue Feb  7 09:58:42 2017
 Instance  1 bias (spec ocpi.bias) on hdl container PL:0, using bias_vhdl/a/bi
as_vhdl in /mnt/ocpi_baseproject/hdl/assemblies//testbias/container-testbias_z
ed_gp1_use_gp1/target-zynq/testbias_zed_gp1_use_gp1.bit.gz dated Thu Feb  9 10
:14:30 2017
 Instance  2 file_write (spec ocpi.file_write) on rcc container rcc0, using fi
le_write in /mnt/net/cdk/lib/components/rcc/linux-x13_3-arm/file_write_s.so da
ted Tue Feb  7 09:58:42 2017
Application XML parsed and deployments (containers and implementations) chosen
Application established: containers, workers, connections all created
Communication with the application established
Dump of all initial  property values:
Property  0: file_read.fileName = "test.input" (cached)
Property  1: file_read.messagesInFile = "false" (cached)
Property  2: file_read.opcode = "0" (cached)
Property  3: file_read.messageSize = "16"
Property  4: file_read.granularity = "4" (cached)
Property  5: file_read.repeat = "<unreadable>"
Property  6: file_read.bytesRead = "0"
Property  7: file_read.messagesWritten = "0"
Property  8: file_read.suppressEOF = "false"
Property  9: file_read.badMessage = "false"
Property 10: file_read.ocpi_debug = "false" (parameter)
Property 11: file_read.ocpi_endian = "little" (parameter)
Property 12: bias.biasValue = "16909060" (cached)
Property 13: bias.ocpi_debug = "false" (parameter)
Property 14: bias.ocpi_endian = "little" (parameter)
Property 15: bias.test64 = "0"
Property 16: file_write.fileName = "test.output" (cached)
Property 17: file_write.messagesInFile = "false" (cached)
Property 18: file_write.bytesWritten = "0"
Property 19: file_write.messagesWritten = "0"
Property 20: file_write.stopOnEOF = "true" (cached)
Property 21: file_write.ocpi_debug = "false" (parameter)
Property 22: file_write.ocpi_endian = "little" (parameter)
Application started/running
Waiting 1 seconds for application to complete
After 1 seconds, stopping application...
Dump of all final  property values:
Property  3: file_read.messageSize = "16"
Property  5: file_read.repeat = "<unreadable>"
Property  6: file_read.bytesRead = "4000"
Property  7: file_read.messagesWritten = "251"
Property  8: file_read.suppressEOF = "false"
Property  9: file_read.badMessage = "false"
Property 15: bias.test64 = "0"
Property 18: file_write.bytesWritten = "4000"
Property 19: file_write.messagesWritten = "250"
%
```

Figure 9: Successful Network Mode Execution

Run the following command to view the input. It should look like Figure 10:

```
$ hexdump test.input | less
```

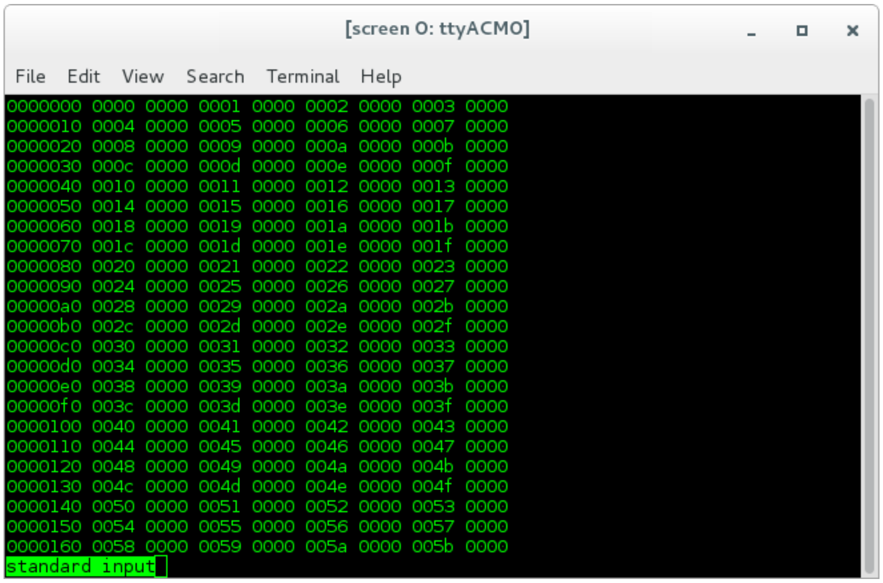


Figure 10: Expected Input

Run the following command to view the output. It should look like Figure 11:

```
$ hexdump test.output | less
```

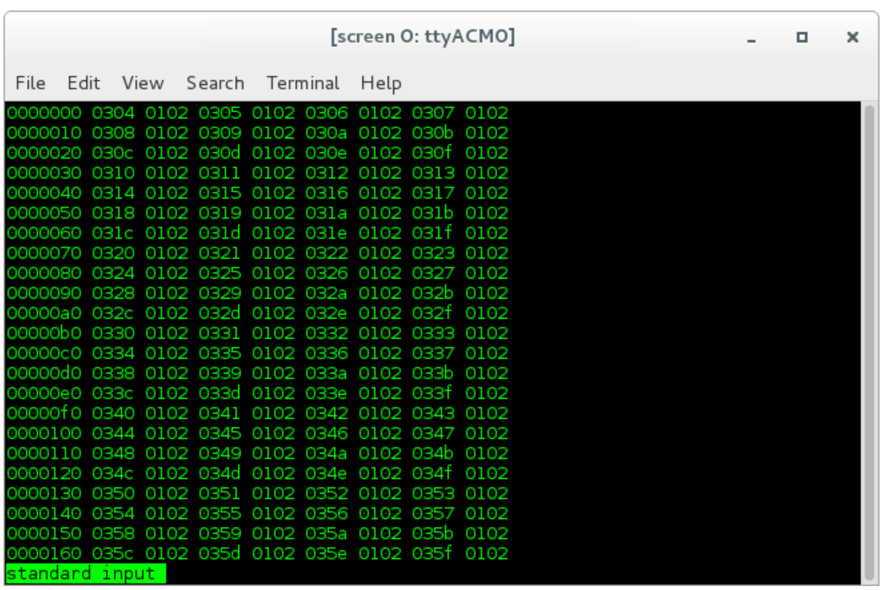


Figure 11: Expected Output

10.2 Standalone Mode

The default setup script sets the OCPI_LIBRARY_PATH variable to include the all of the artifacts that are required to execute the application. Specifically, all three of the artifacts that are located on the SD card are mounted at /mnt/card/opencpi/xilinx13_3/artifacts. After running mysetup.sh, navigate to /mnt/card/opencpi/applications and ensure the OCPI_LIBRARY_PATH variable is configure as shown below:

```
$ cd /mnt/card/opencpi/applications
$ export OCPI_LIBRARY_PATH=$OCPI_LIBRARY_PATH:/mnt/card/opencpi/xilinx13_3/artifacts
```

Run the application using the following command:

```
$ ocpirun -v -t 1 -d -m bias=hdl bias.xml
```

The output should be similar to Figure 12:

```
[screen 0: ttyACM0]
File Edit View Search Terminal Help
% ocpirun -v -t 1 -d -m bias=hdl bias.xml
Available containers are:  0: PL:0 [model: hdl os:  platform: zed], 1: rcc0 [mod
el: rcc os: linux platform: xilinx13_3]
Actual deployment is:
 Instance  0 file_read (spec ocpi.file_read) on rcc container rcc0, using file_r
eFound normal sync word
ad in /mnt/card/opencpi/artifacts/008-file_read_s.so dated Wed Feb  8 17:14:14 2
017
 Instance  1 bias (spec ocpi.bias) on hdl container PL:0, using bias_vhdl/a/bias
_vhdl in /mnt/card/opencpi/artifacts/000-testbias_zed_base.bitz dated Wed Feb  8
 17:14:14 2017
 Instance  2 file_write (spec ocpi.file_write) on rcc container rcc0, using file
_write in /mnt/card/opencpi/artifacts/010-file_write_s.so dated Wed Feb  8 17:14
:14 2017
Application XML parsed and deployments (containers and implementations) chosen
Application established: containers, workers, connections all created
Communication with the application established
Dump of all initial  property values:
Property  0: file_read.fileName = "test.input" (cached)
Property  1: file_read.messagesInFile = "false" (cached)
Property  2: file_read.opcode = "0" (cached)
Property  3: file_read.messageSize = "16"
Property  4: file_read.granularity = "4" (cached)
Property  5: file_read.repeat = "<unreadable>"
Property  6: file_read.bytesRead = "0"
Property  7: file_read.messagesWritten = "0"
Property  8: file_read.suppressEOF = "false"
Property  9: file_read.badMessage = "false"
Property 10: file_read.ocpi_debug = "false" (parameter)
Property 11: file_read.ocpi_endian = "little" (parameter)
Property 12: bias.biasValue = "16909060" (cached)
Property 13: bias.ocpi_debug = "false" (parameter)
Property 14: bias.ocpi_endian = "little" (parameter)
Property 15: bias.test64 = "0"
Property 16: file_write.fileName = "test.output" (cached)
Property 17: file_write.messagesInFile = "false" (cached)
Property 18: file_write.bytesWritten = "0"
Property 19: file_write.messagesWritten = "0"
Property 20: file_write.stopOnEOF = "true" (cached)
Property 21: file_write.ocpi_debug = "false" (parameter)
Property 22: file_write.ocpi_endian = "little" (parameter)
Application started/running
Waiting 1 seconds for application to complete
After 1 seconds, stopping application...
Dump of all final  property values:
Property  3: file_read.messageSize = "16"
Property  5: file_read.repeat = "<unreadable>"
Property  6: file_read.bytesRead = "2000"
Property  7: file_read.messagesWritten = "125"
Property  8: file_read.suppressEOF = "false"
Property  9: file_read.badMessage = "false"
Property 15: bias.test64 = "0"
Property 18: file_write.bytesWritten = "1984"
Property 19: file_write.messagesWritten = "124"
%
```

Figure 12: Successful Standalone Mode Execution

Run the following command to view the input. It should look like Figure 13:

```
$ hexdump test.input | less
```

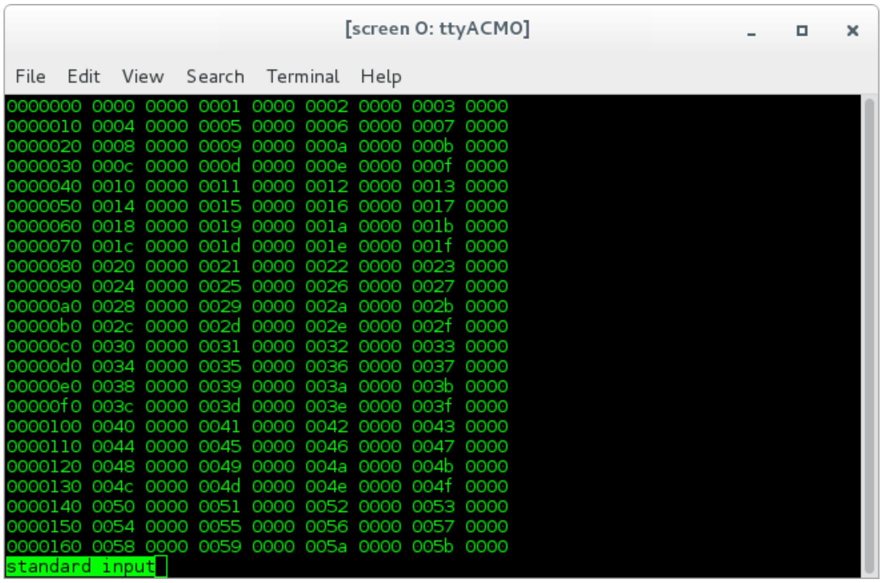


Figure 13: Expected Input

Run the following command to view the output. It should look like Figure 14:

```
$ hexdump test.output | less
```

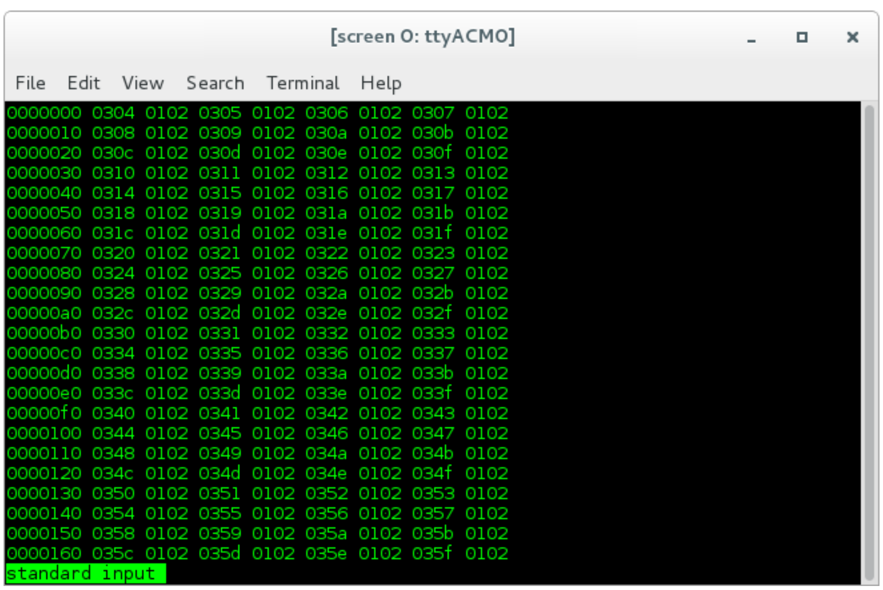


Figure 14: Expected Output

Appendices

A Using ISE instead of Vivado with the ZedBoard

If the user requires the use of the Xilinx ISE tools, rather than the Vivado (recommended), a different OpenCPI platform must be targeted for building bitstreams for the Zedboard. Specifically, the *zed_ise* (*zynq_ise* is the target) OpenCPI platform is built using ISE tools, where as the *zed* (*zynq* is the target) OpenCPI platform is built using Vivado tools.

Its critical to note that the entire *core* and *assets* projects must be built using ISE tools and that the *zed_ise* platform is located in the *assets* project.

After ensuring the proper environment variables are set in support of the ISE tools, use the following command to build from the top-level of a project:

```
$ ocpidev build --hdl-platform zed_ise
```

B Driver Notes

When available, the driver will attempt to make use of the CMA region for direct memory access. In use cases where many memory allocations are made, the user may receive the following kernel message:

```
alloc_contig_range␣test_pages_isolated([memory␣start],␣[memory␣end])␣failed
```

This is a kernel warning, but does not indicate that a memory allocation failure occurred, only that the CMA engine could not allocate memory in the first pass. Its default behavior is to make a second pass, and if that succeeded, the end user should not see any more error messages. This message cannot be suppressed, but can be safely ignored. An actual allocation failure will generate unambiguous error messages.

C Deprecated Zipper

Beginning with OpenCPI Version 1.5, support for Lime Microsystems' Zipper card is now deprecated, and the following note and figure have been removed from the main body of this Getting Started Guide:

OpenCPI has been tested on revisions C and D of the Zedboard. However, limitations have been observed for both revisions when used with the Zipper daughter-card, details are provided in Myriad-RF_1_Zipper_Limitations.pdf.

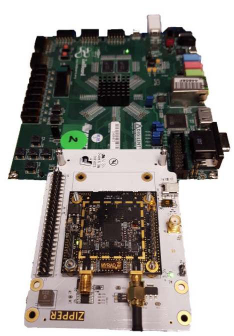


Figure 15: ZedBoard With Zipper and MyriadRF-1 Connected to the FMC Slot