ZedBoard Getting Started Guide

Version 1.3

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

Table of Contents

1 References **4**

2 Overview **5**

3 Prerequisites **5**

4 Script Setup **6**
4.1 Setup for Either Mode . . . 7
4.2 Setup for Network Mode . . . 7
4.3 Setup for Standalone Mode . . . 7
4.4 Multiple ZedBoards on the same network . . . 7

5 Hardware and SD Card Setup **8**

6 Software Setup **10**
6.1 Network Mounting Mode . . . 10
6.1.1 CentOS 6 . . . 11
6.1.2 CentOS 7 . . . 11
6.2 Standalone Mode . . . 14

7 Verification **15**
7.1 Building . . . 15
7.2 Running Network Mode . . . 17
7.3 Running Standalone Mode . . . 19

Appendices **20**

A Using ISE instead of Vivado with the ZedBoard **20**

B Driver Notes **21**

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	Published By	Link
Getting Started	ANGRYVIPER Team	`Getting_Started.pdf`
Installation Guide	ANGRYVIPER Team	`RPM_Installation_Guide.pdf`
Acronyms and Definitions	ANGRYVIPER Team	`Acronyms_and_Definitions.pdf`

Table 1: References

2 Overview

This purpose of this document is to allow the user to install OpenCPI on the ZedBoard Development Board.

3 Prerequisites

This guide assumes that 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`.
`opencpi-*.x86_64.rpm`	Base installation RPM includes the runtime portion of the Component Development Kit (CDK), scripts for creating the user's workspace, limited documentation, and a read-only `ocpi.core` Project 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-hw-platform-zed-*.noarch.rpm`	Additional files necessary to build the framework targeting specific hardware platforms. Automatically requires the zed sw-platform package.
`opencpi-project-assets*.noarch.rpm`	The `ocpi.assets` Project, which contains the remaining supported OpenCPI resources, e.g. additional Platform Support, Workers, Demo Applications, etc.
`opencpi-sw-platform-xilinx13_3-*.noarch.rpm`	Additional files necessary to build the framework targeting the xilinx13_3 software platform.

This guide assumes that the Core and Assets projects have user copies and have been registered. Creating user copies of projects is done by using the new_project_source script as described in the Getting Started Guide. registering projects is done by running the command `ocpidev register project` at the top level of the project.

The development board that is expected to be used is Digilent's ZedBoard. OpenCPI has been tested on ZedBoard Rev. C and Rev. D. Each has separate limitations (described in Myriad-RF_1_Zipper_Limitations.pdf). An Ethernet cable will need to be be plugged in to the Ethernet port on the board. The ZedBoard gets an IP Address by using DHCP. It is a requirement if operating in network mode (discussed later) that the Ethernet cable is connected to a network that uses DHCP[1].

There is a micro-USB serial port on the back of the ZedBoard labeled UART. A Male micro-USB cable will need to be plugged into this port to access the serial connection.

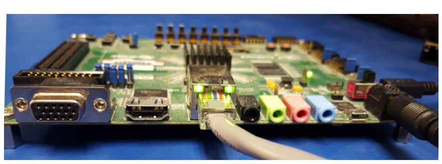


Figure 1: Connected Ethernet

[1]Static IP addresses are possible but have not been integrated with the OpenCPI startup procedure at this time.

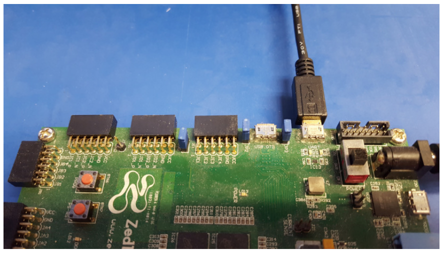


Figure 2: Connected Serial USB

On one side of the development board, there is a FMC LPC slot. Optionally, this can be used to connect plug-in modules. OpenCPI has been tested with Lime Microsystems' Zipper card with the MyriadRF-1, Analog devices FMCOMMS2, and Analog devices FMCOMMS3 connected to the ZedBoard.

Below the FMC LPC slot (underneath the board), is the SD card slot you will be using throughout this guide.

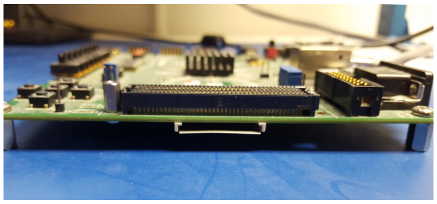


Figure 3: ZedBoard FMC Slot and SD card Slot

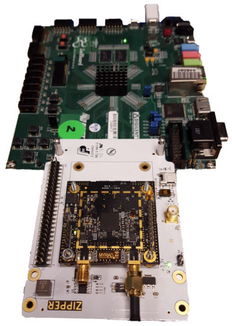


Figure 4: ZedBoard With Zipper and MyriadRF-1 Connected to the FMC Slot

4 Script Setup

There are two modes for running applications on any embedded radio or development board: network mode and standalone mode. Network mode is when the development system hosts the OpenCPI tree as an NFS server to the ZedBoard as an NFS client. This configuration provides easy and dynamic access to all of OpenCPI, and presumably any components and applications. Standalone mode is when all the artifacts are located on the board's local storage (*e.g.* SD card) and no network connection is required. This is a better *deployment* mode, and is better for situations where a network connection is not possible or practical. Network mode is generally preferred when it is possible because it makes the development process easier than standalone mode.

For each mode, there are separate startup scripts that are run on the board. These scripts will need to be modified for each user's specific setup and file structure. There are starting points for these scripts that are provided by the framework. For networked mode the script is located at `/opt/opencpi/boot_support/zed/OpenCPI-SD-zed/opencpi/default_mynetsetup.sh`. For standalone mode, the script is located at `/opt/opencpi/boot_support/OpenCPI-SD-zed/opencpi/default_mysetup.sh`. These scripts need to be renamed to `mynetsetup.sh` and `mysetup.sh` before they are copied over to the SD card in Section 5.

4.1 Setup for Either Mode

`If Linux system time is not required to be accurate, you can skip this step.`

For either usage mode, the following settings that are passed by `mynetsetup.sh/mysetup.sh` to the `zynq_net_setup.sh/zynq_setup.sh` scripts *might* need to be modified:

- The system to be used as a time server. Defaulted to "time.nist.gov". Change this if you have a local time server that supports RFC-868.
- The current timezone description. Defaulted to "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 you do not have a time server, or cannot connect to a time server, you will need to manually set the time at start up. Use the `date` command to manually set the Linux system time. See `man date` on the host PC for more information.

4.2 Setup for Network Mode

When using Network Mode, the following modifications are required:

1. In `mynetsetup.sh`, find the following lines which are necessary for mounting Assets Project and the Core Project:

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

2. Edit `/home/user/ocpiCore` and `/home/user/ocpiAssets` to reflect the paths to the Core Project and Assets Project on the host.

4.3 Setup for Standalone Mode

For Standalone Mode, all OpenCPI artifacts required to run any applications on the ZedBoard need to be copied onto the SD card. So, you must build these artifacts earlier when operating in Standalone Mode. Perform the steps in Section 7.1 now. The artifacts created will be copied over to the SD card in Section 5. In general, you will want to copy any required `.so` (RCC workers), `.bit.gz` (hdl assemblies), and application XMLs or executables onto the SD card.

4.4 Multiple ZedBoards on the same network

If you plan to have multiple ZedBoards on one network, you will need to make a change to the zynq startup scripts. This is necessary because by default the ZedBoards will all have the same MAC address. To resolve this, uncomment the following lines in the zynq_net_setup.sh and zynq_setup.sh scripts:

```
# ifconfig eth0 down
# ifconfig eth0 hw ether 00:0a:35:00:01:23
# ifconfig eth0 up
# udhcpc
```

5 Hardware and SD Card Setup

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

This optional section provides the steps for creating an SD card backup image. The following sections assume the SD card is empty.

- Determine the device file name for the SD card. It will likely be something like `/dev/sdb` or `/dev/mmcblk0`. To do this, you can take a look at the end of dmesg: "`dmesg | tail -n 15`".
- Run the command "`dd if=DEVICENAME of=backup.image`" where DEVICENAME was determined above. This step should take ∼ 15 minutes depending on the card size.

To restore the card back to the original contents, run the command "`dd of=DEVICENAME if=backup.image`" (Do not do this step unless you want the original contents back on the SD card.)

Formatting and populating the SD card

- Format the SD card with a single FAT32 partition.
- Copy the all of the contents of `/opt/opencpi/boot_support/OpenCPI-SD-zed/` to this partition.

Copy the files needed for Standalone Mode to SD card

For Standalone Mode:

- Copy the following files into the `opencpi/xml` directory on this partition:
  - `<Assets_Project>/applications/bias.xml`
  - `<Assets_Project>/applications/test.input`
- Copy the following files into the `opencpi/artifacts` directory on this partition:
  - `<Assets_Project>/hdl/assemblies/testbias/container-testbias_zed_base/target-zynq/testbias_zed_base.bit.gz`

The SD card file structure should be as follows when these steps have been completed (ellipsis used to denote repetitive files):

```
+-- boot.bin
+-- devicetree.dtb
+-- opencpi
|   +-- artifacts
|   |   +-- 001-bias_cc_s.so
       ...
|   |   +-- 020-testzc_s.so
|   |   \-- testbias_zed_base.bit.gz # only in Standalone Mode
|   +-- bin
|   |   +-- ocpibootstrap.sh
|   |   +-- ocpidriver
|   |   +-- ocpihdl
|   |   +-- ocpi_linux_driver
|   |   +-- ocpirun
|   |   +-- ocpiserve
|   |   +-- ocpixml
|   |   \-- ocpizynq
|   +-- default_mynetsetup.sh
|   +-- default_mysetup.sh
|   +-- lib
|   |   \-- linux-x13_3-arm
|   |       +-- libocpi_application_s.so
           ...
|   |       +-- libocpi_util_s.so
|   |       +-- mdev-opencpi.rules
|   |       \-- opencpi-3.10.0-xilinx-dirty-v14.7.ko
|   +-- mynetsetup.sh
|   +-- mysetup.sh
|   +-- system.xml
|   +-- xml
|   |   +-- bias.xml
       ...
|   |   +-- testbias.xml
|   |   +-- test.input # only in Standalone Mode
|   |   \-- time_test.xml
|   +-- zynq_net_setup.sh
|   \-- zynq_setup.sh
+-- uImage
\-- uramdisk.image.gz
```

Serial setup

By default, the USB to serial adapter will connect as read-only unless `udev` rules are added to have the device connect as read write. Copy the file from the rpm install `/opt/opencpi/boot_support/zed/udev.rules/98-zedboard.rules` to `/etc/udev/rules.d/`. This will cause the USB to Serial adapter to connect as `/dev/zed0` with read and write permissions for all users. To connect to the serial port, use the command "`screen /dev/zed0 115200`".

Boot the ZedBoard from the SD card

1. Remove power from the ZedBoard unit.
2. Ensure jumpers are configured correctly

   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.

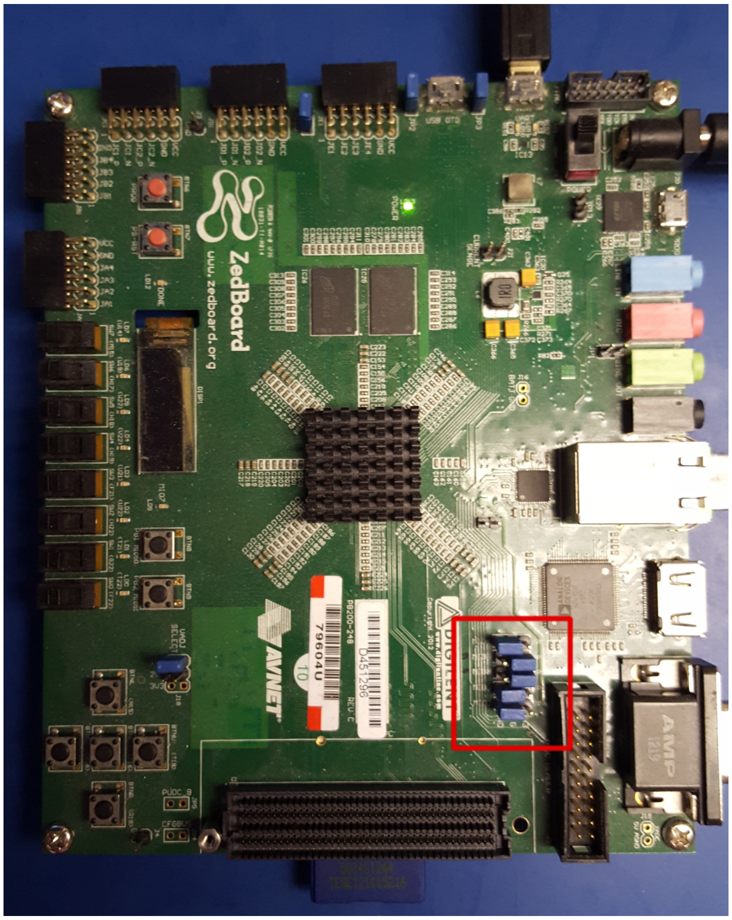


Figure 5: Top View of the ZedBoard with J10, J9, J8 Set

3. Insert the SD card into the SD card slot.
4. Connect a terminal to the micro-USB UART connector of the ZedBoard with a baud rate of 115200.
   - 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.

6 Software Setup

6.1 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.

6.1.1 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`).

```
% sudo vi /etc/exports

/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
/home/user/coreProj XX.XX.XX.XX/MM(rw,sync,no_root_squash,no_subtree_check)
/home/user/assetsProj 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
```

6.1.2 CentOS 7

From the host, install the necessary tools using yum:

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

[a] 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 installed

From the host, allow NFS past SELINUX:

```
% 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`).

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

/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
/home/user/coreProj XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
/home/user/assetsProj 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)
/home/user/coreProj XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=34)
/home/user/assetsProj XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=35)
```

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.

Power cycle the ZedBoard

Make sure the USB cable is plugged into the UART micro-USB port and connected to the network if applicable . The ZedBoard should now boot a valid OpenCPI environment. The user name and password for the development board are both "root". A successful boot up screen will look as follows:

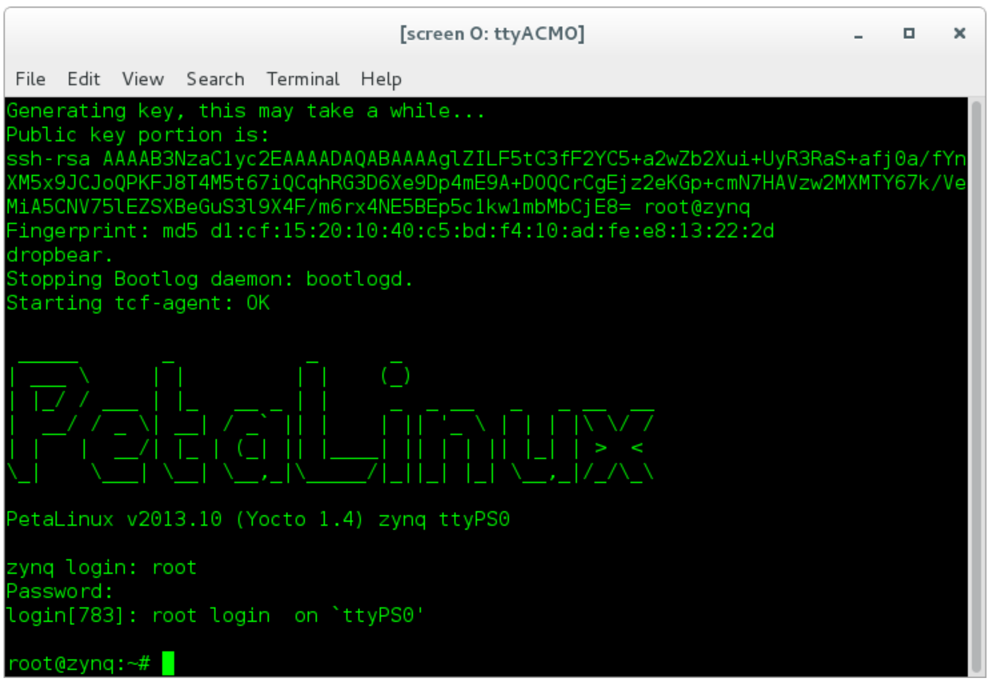


Figure 6: Successful Boot

Using the ZedBoard in Network Mode

Every time the development board is rebooted, the user is required to run the **mynetsetup.sh**, which configures the system for OpenCPI with support for network/NFS mode[2]. The user passes the network address of the development system in as the only argument to **mynetsetup.sh**.

[2]This script calls the **zynq_net_setup.sh** script, which is not user-modifiable.

After the development board is booted, the first thing that needs to be done is to source the `mynetsetup.sh` script with

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

and changing XX.XX.XX.XX to the IP address of the NFS host (your development machine, *e.g.* 192.168.1.10). A successful run is shown in Figure 7.

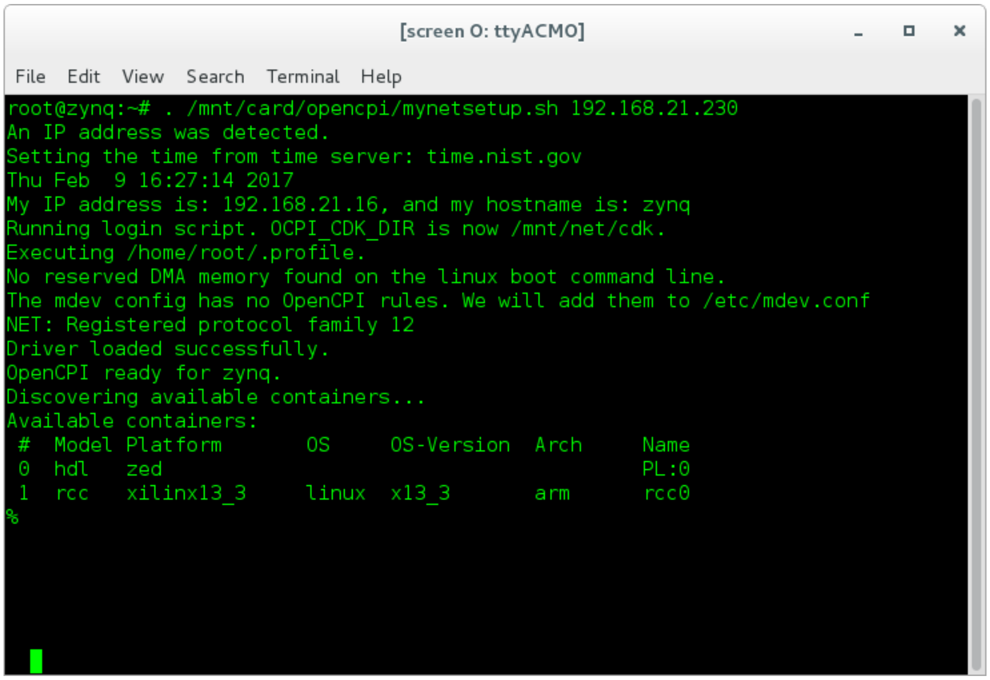


Figure 7: Successful Network Mode Setup

6.2 Standalone Mode

All artifacts for any applications or tests that need to be located on the SD card must be in the `opencpi/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.

Power cycle the ZedBoard

The ZedBoard should now boot a valid OpenCPI environment. The user name and password for the development board are both "root". A successful boot up screen will look as follows:

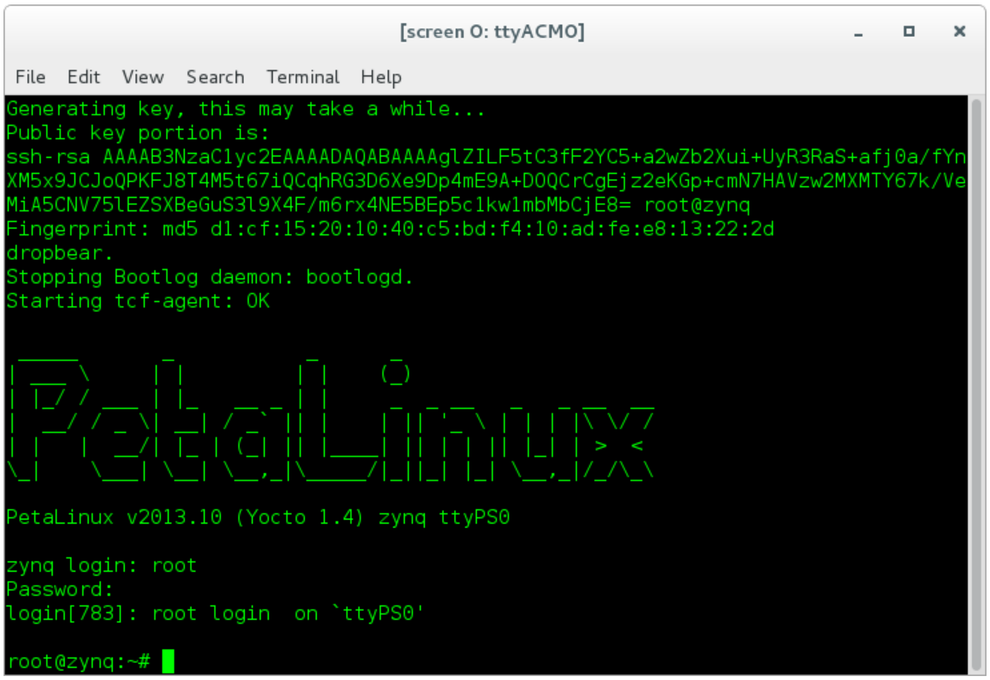


Figure 8: Successful Boot

Using the ZedBoard in Standalone Mode

Every time the development board is rebooted, the user is required to run the `mysetup.sh`, which configures the system for OpenCPI[3]. Any time that a new version of OpenCPI is released, the SD card will need to be recreated in order to update the artifacts and the executables that are stored on the SD card.

After the development board is booted, the first thing that needs to be done is to source the `mysetup.sh` script with

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

A successful run of this will be as follows:

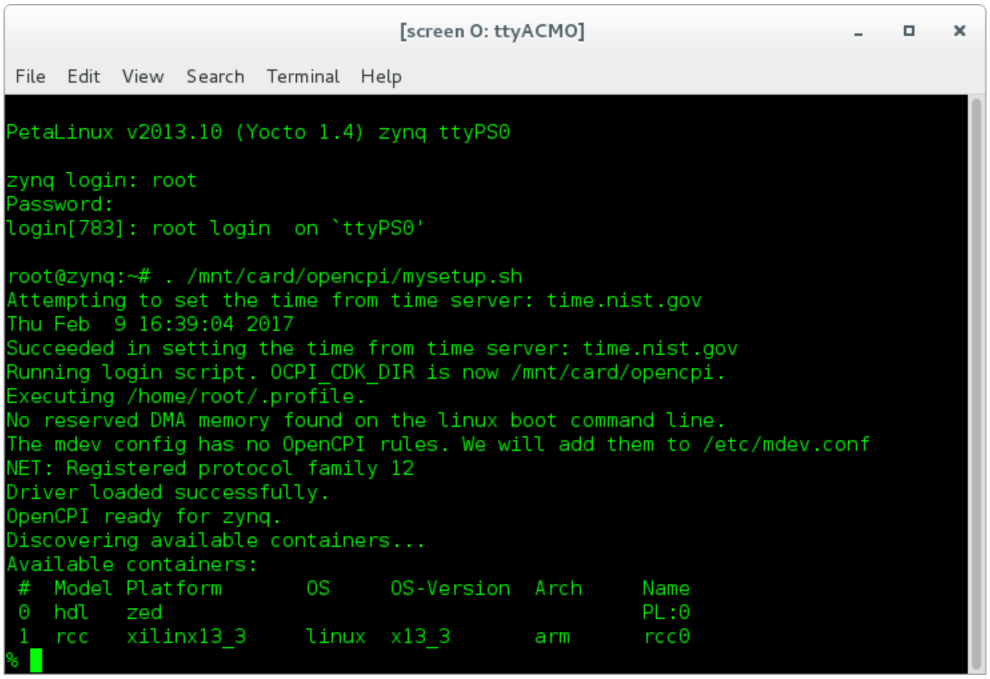


Figure 9: Successful Standalone Mode Setup

7 Verification

The installation can be verified by running an application that uses both RCC and HDL workers. There is an application that uses two RCC and one HDL worker located in `<Assets Project>/applications/bias.xml`. The two RCC workers are provided pre-built on the SD card or mounted CDK directory. The HDL worker needs to be built into an assembly before the application can be executed.

7.1 Building

If operating in Standalone Mode, these steps should have been performed earlier. (See 4.2.)

First the Core project needs to be built for the zed platform. This is performed at the top level of the Core project running the command `ocpidev build --hdl-platforms zed`. This will take about 45 minutes to complete.

Then the Zed platform needs to be built this is located in the Assets project. To build this at the top level of the Assets project run the command `ocpidev build hdl platform zed`. This will take about 45 minutes to complete.

There is an existing assembly in the assets project that is the bias worker with its inputs and outputs connected to software. This assembly needs to be built for "zed" platform. It is located in `<Assets_Project>/hdl/assemblies/`

[3]This script calls the `zynq_setup.sh` script, which is not user-modifiable.

testbias/. At the top level of the project run the following command: "ocpidev build hdl assembly testbias --hdl-platform zed". This should take about 15 minutes to complete. You can confirm that it succeeded by locating the *.bit.gz file in container-testbias_zed_base/target-zed/.

7.2 Running Network Mode

The default setup script should already set the `OCPI_LIBRARY_PATH` variable to point to the RCC workers that are required to execute the application, but it needs to be updated to point to the assembly that was built. After running the `mynetsetup.sh` script, navigate to `/mnt/ocpi_core/applications`. Update the `OCPI_LIBRARY_PATH` variable using the following command:
`export OCPI_LIBRARY_PATH=$OCPI_LIBRARY_PATH:/mnt/ocpi_assets/hdl/assemblies/`

In order to run the application, use the following command: "`ocpirun -v -t 1 -d -m bias=hdl bias.xml`" The output should be as follows:

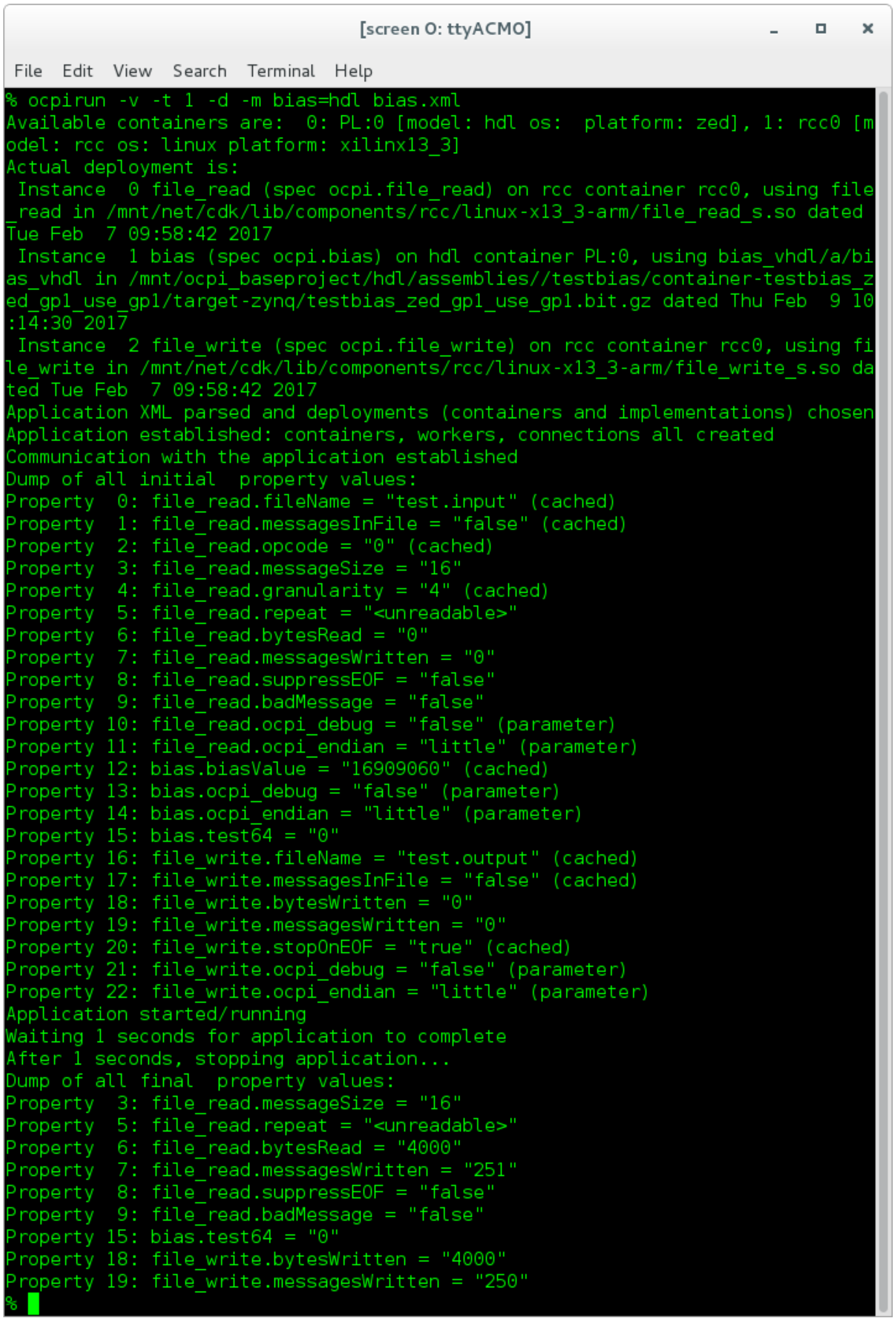


Figure 10: Successful Network Mode Execution

To view the input file, run the following command: "`hexdump test.input | less`" and the file should look like Figure 11:

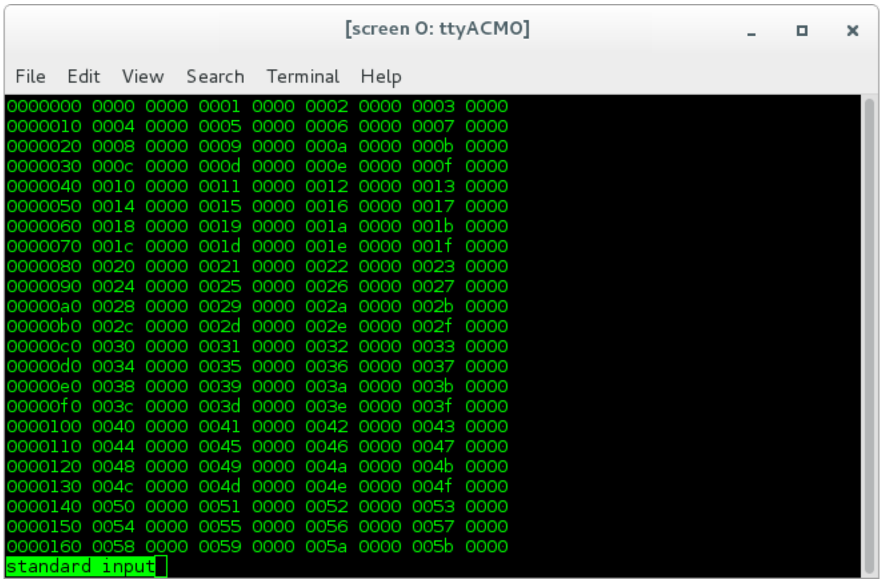


Figure 11: Expected Input

To view the output file, run the following command: "`hexdump test.output | less`" and the file should look like Figure 12:

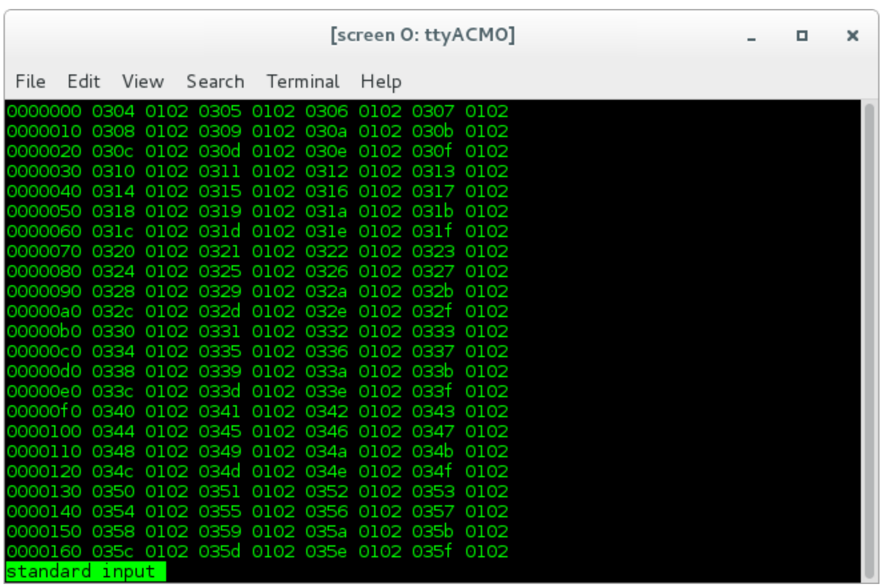
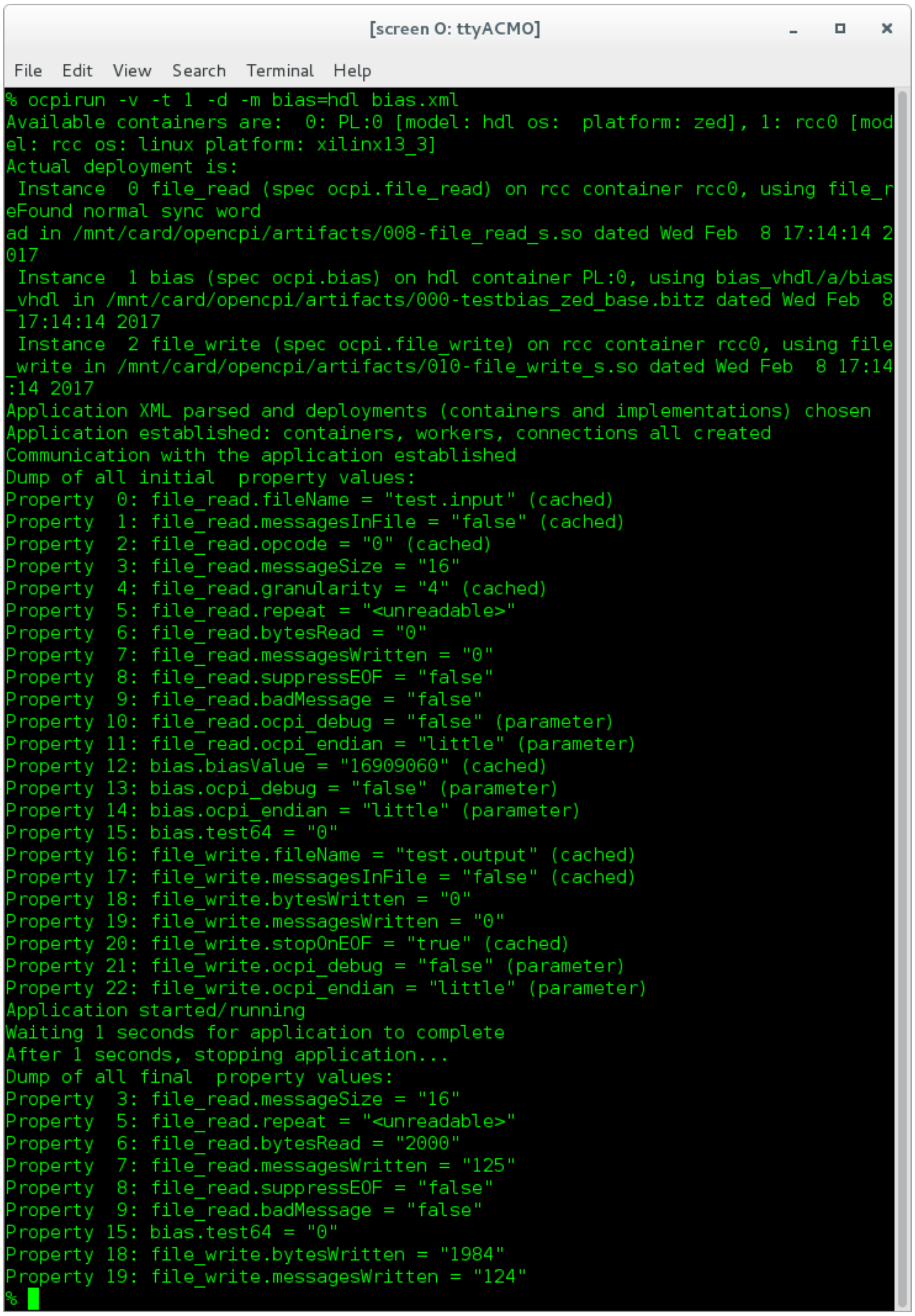


Figure 12: Expected Output

7.3 Running Standalone Mode

The default setup script should already set the `OCPI_LIBRARY_PATH` variable to all the required locations for the framework to execute the application. All three of the artifacts that are located on the SD card are mounted at `/mnt/card/opencpi/artifacts`. After running `mysetup.sh`, navigate to `/mnt/card/opencpi/xml`.
In order to run the application, use the following command: "`ocpirun -v -t 1 -d -m bias=hdl bias.xml`" The output should be as follows:




```
% 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 13: Successful Standalone Mode Execution

To view the input file, run the following command: “`hexdump test.input | less`” and the file should look like Figure 14:

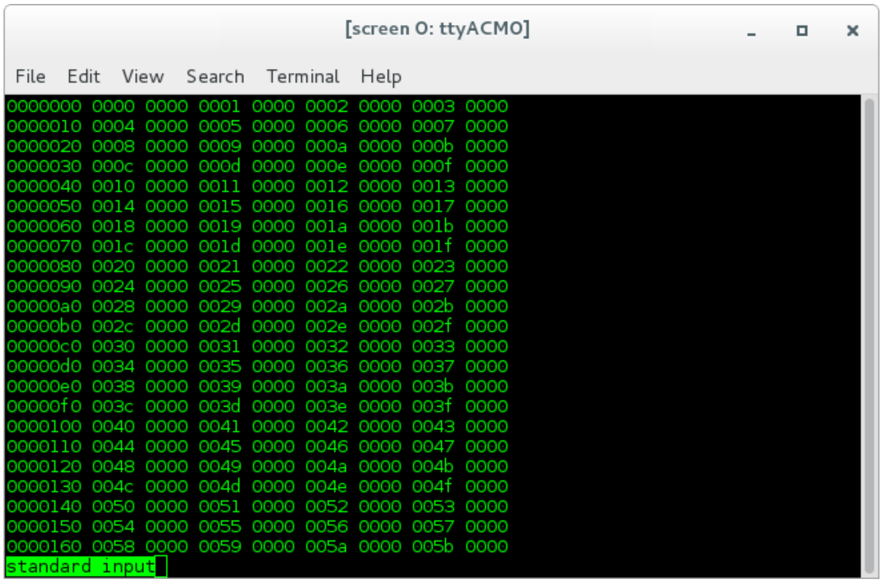


Figure 14: Expected Input

To view the output file, run the following command: “`hexdump test.output | less`” and the file should look like Figure 15:

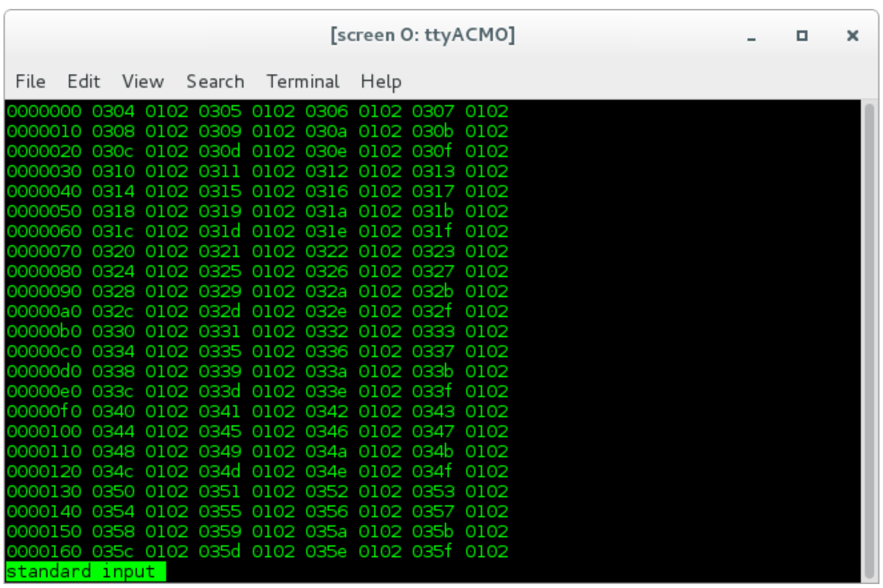


Figure 15: Expected Output

Appendices

A Using ISE instead of Vivado with the ZedBoard

It is recommended that you use the default toolset (Xilinx Vivado) to build ZedBoard bitstreams with OpenCPI. However, if you wish to use ISE instead, you can use the `zed_ise` platform and `zynq_ise` target. So, instead of "`ocpidev build --hdl-platform zed`", run "`ocpidev build --hdl-platform zed_ise`". Additionally, you will have to copy the `system.xml` file located in <Assets_Project>/hdl/platforms/zed_ise/sd_card/ to the `opencpi` directory of the SD card.

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. An actual allocation failure will generate unambiguous error messages.