OpenCPI
Matchstiq-Z1 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

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`
Overview	OpenCPI	`https://goo.gl/RskxiV`

Table 1: References

2 Overview

This purpose of this document is to allow the user to install OpenCPI on the Matchstiq-Z1 SDR. The software platform on the Matchstiq is the same as the software platform for the ZedBoard. This means several of the files that are used to create the Matchstiq-Z1 SD card come from the ZedBoard SD card.

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 SDR that is expected to be used is the Epiq Matchstiq-Z1 (ensure you are using the Z1 version). A USB to Ethernet adapter will need to be plugged in to the front panel micro-USB port. The SDR gets an IP Address by using DHCP. It is a requirement if operating in network mode (discussed later) that the USB to Ethernet adapter is connected to a network that uses DHCP[1].

There is a micro-USB serial port on the back of the SDR; to expose this port, the two screws in the back plate need to be removed. A Male micro-USB cable will need to be plugged into this port to access the serial connection. On some versions of the Matchstiq-Z1, *this connector can be very fragile*, **so be careful**.

On the front panel of the SDR, there are three labeled SMB (50 Ohm) connectors. One each for Receive (RX), Transmit(TX), and GPS. The SDR comes with 2 SMB to SMA adapters. This means that any RF cabling should use SMA or SMB connectors and be rated up to at least 3GHz which is the maximum TX/RX frequency of the radio.

4 Script Setup

There are two modes for running applications on any embedded radio: network mode and standalone mode. Network mode is when the development system hosts the OpenCPI tree as an NFS server to the Matchstiq-Z1 as an NFS

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

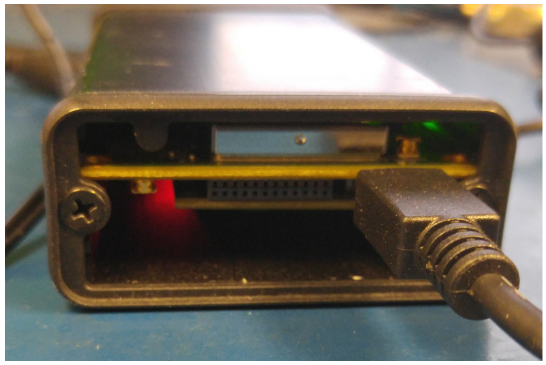

Figure 1: Connected Back Panel

Figure 2: Connected Front Panel

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 SDR'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 SDR. 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/zed/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.

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

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.

For Standalone Mode, all OpenCPI artifacts required to run any applications on the SDR 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 into the ATLAS partition of the SD card.

5 Hardware and SD Card Setup

The Matchstiq-Z1 SDR is equipped with two SD card slots: one internal and one accessible via the front panel. The SDRs are shipped from Epiq Solutions with an SD card installed in the internal slot that is loaded with their embedded environment. However, when an SD card is installed in the front panel SD slot, the SDR will choose to operate from this SD card rather than the internal SD card. Therefore, a user can easily switch the SDR between operating in the Epiq or AV environments.

Since the internal SD card is already formatted correctly, this guide assumes it has been removed for use in the front panel slot. If you wish to use your own SD card, you must ensure that it is formatted correctly. To gain access to the internal SD card slot, remove the screws from the front and back plates of the SDR and slide the board assembly out of the enclosure. Flip the SD card slot open and lift the card out.

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

- 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.)

Change boot files on the "ATLAS" partition

All files except `u-boot.bin` can be ignored or deleted. Any files/directories copied to the "ATLAS" partition will appear at `/mnt/card` on the Matchstiq-Z1.

- Copy the following files/directories onto this partition:
  - The entire folder `/opt/opencpi/boot_support/zed/OpenCPI-SD-zed/opencpi`

    Remove from the SD card `opencpi/artifacts/*.bitz`
  - `/opt/opencpi/boot_support/zed/OpenCPI-SD-zed/uImage`
  - `/opt/opencpi/boot_support/zed/OpenCPI-SD-zed/uramdisk.image.gz`
  - `<AssetsProject>/hdl/platforms/matchstiq_z1/sd_card/iveia-atlas-i-z7e.dtb`[a]
- Copy the following files into the `opencpi` directory on this partition:
  - `ocpiassets/hdl/platforms/matchstiq_z1/sd_card/system.xml`

[a]This is the ZedBoard device tree with minor changes for Matchstiq-Z1.

Copy the files needed for Standalone Mode to the "ATLAS" partition

For Standalone Mode:

- Copy the following files into the `opencpi/xml` directory on this partition:
  - `<AssetsProject>/applications/bias.xml`
  - `<AseetsProject>/applications/test.input`
- Copy the following files into the `opencpi/artifacts` directory on this partition:
  - `<AssetsProject>/hdl/assemblies/testbias/container-testbias_matchstiq_z1_base/target-zynq/testbias_matchstiq_z1_base.bit.gz`

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

```
+-- iveia-atlas-i-z7e.dtb
+-- opencpi
|   +-- artifacts
|   |   +-- 001-bias_cc_s.so
        ...
|   |   +-- 020-testzc_s.so
|   |   \-- testbias_matchstiq_z1_base.bit.gz # only in Standalone Mode
|   +-- bin
|   |   +-- ocpibootstrap.sh
|   |   +-- ocpidriver
|   |   +-- ocpihdl
|   |   +-- ocpi_linux_driver
|   |   +-- ocpirun
|   |   +-- ocpiserve
|   |   +-- ocpixml
|   |   \-- ocpizynq
|   +-- 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
+-- u-boot.bin
+-- uImage
\-- uramdisk.image.gz
```

No changes required for “SDHOME” partition

All the files in this partition can be ignored. If space for files is required for your application, they can be deleted.

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 Assets Project `hdl/platforms/matchstiq_z1/97-matchstiq_z1.rules` to `/etc/udev/rules.d/`. This will cause the USB to Serial adapter to connect as `/dev/matchstiq_z1_0` with read and write permissions for all users. To connect to the serial port use the command “`screen /dev/matchstiq_z1_0 115200`”.

Update U-Boot variables

1. Remove power from the Matchstiq-Z1 unit.

2. Insert the SD card into the front panel SD card slot.

3. Connect a terminal to the rear micro-USB connector of the Matchstiq-Z1 with a baud rate of 115200.
   - per the previous section, "`screen /dev/matchstiq_z1_0 115200`" can be used to connect to the serial port.
4. Apply power to the Matchstiq-Z1 with the terminal still connected and stop the boot process by hitting any key to enter the U-Boot terminal.
5. Run the following commands to setup the environment variables:
   - `setenv bootcmd 'ivmmc; run ocpiboot'`
   - `setenv ocpiboot 'setenv bootargs console=ttyPS0,115200n8 root=/dev/ram rw earlyprintk; setenv fdt_high ffffffff; setenv initrd_high 0x1000000; fatload mmc ${iv_mmc} ${dtbaddr} ${dtbfile}; fatload mmc ${iv_mmc} ${loadaddr} ${bootfile}; fatload mmc ${iv_mmc} 0x2000000 uramdisk.image.gz; bootm ${loadaddr} 0x2000000 ${dtbaddr}'`

     *Note: This should be a one-line command. Make sure there are no newlines.
   - `saveenv`
6. These U-Boot environment variables are now saved to the second partition of the SD card

Verify that the changes are correct by running the command "`env p`" and comparing to:

```
baudrate=115200
bootcmd=ivmmc;run ocpiboot
bootdelay=3
bootfile=uImage
defargs=setenv bootargs console=ttyPS0,115200n8 mem=240M iv_mb=${iv_mb} iv_io=${iv_io}
iv_bp=${iv_bp} iv_mmc=${iv_mmc} ${otherargs}
dtbaddr=0x02a00000
dtbfile=iveia-atlas-i-z7e.dtb
iv_io=205-00034-00-A0,,Atlas-II_GF_Carrier
iv_io_default=205-00034-00-A0,,Atlas-II_GF_Carrier
iv_io_ord=00034
iv_mb=205-00049-00-B1,A2WT9,Atlas-I-Z7e
iv_mb_ord=00049
iv_mmc=0
loadaddr=0x03000000
mmcdtload=fatload mmc ${iv_mmc} ${dtbaddr} ${dtbfile};fdt addr ${dtbaddr};fdt set
/chosen bootargs "${bootargs}";fdt ivclean ${iv_mb_ord}
mmcxload=axi_reset 1; fatload mmc ${iv_mmc} ${loadaddr} ${xloadfile};xload ${loadaddr}
${filesize}; axi_reset 0;
ocpiboot=setenv bootargs console=ttyPS0,115200n8 mem=240M root=/dev/ram rw earlyprintk;
setenv fdt_high ffffffff;  setenv initrd_high 0x1000000; fatload mmc ${iv_mmc} ${dtbaddr}
${dtbfile}; fatload mmc ${iv_mmc} ${loadaddr} ${bootfile};  fatload mmc ${iv_mmc} 0x2000000
uramdisk.image.gz; bootm ${loadaddr} 0x2000000 ${dtbaddr}
sdboot=run mmcxload;run defargs;fatload mmc ${iv_mmc} ${loadaddr} ${bootfile};run
mmcdtload;setenv fdt_high ffffffff;bootm ${loadaddr} - ${dtbaddr}
stderr=serial
stdin=serial
stdout=serial
xloadfile=xilinx.bit

Environment size: 1283/131068 bytes
```

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 Matchstiq-Z1

Make sure the USB to Ethernet adapter is plugged into the front panel. The SDR should now boot a valid OpenCPI environment. The user name and password for the SDR are both "root". A successful boot up screen will look as follows:

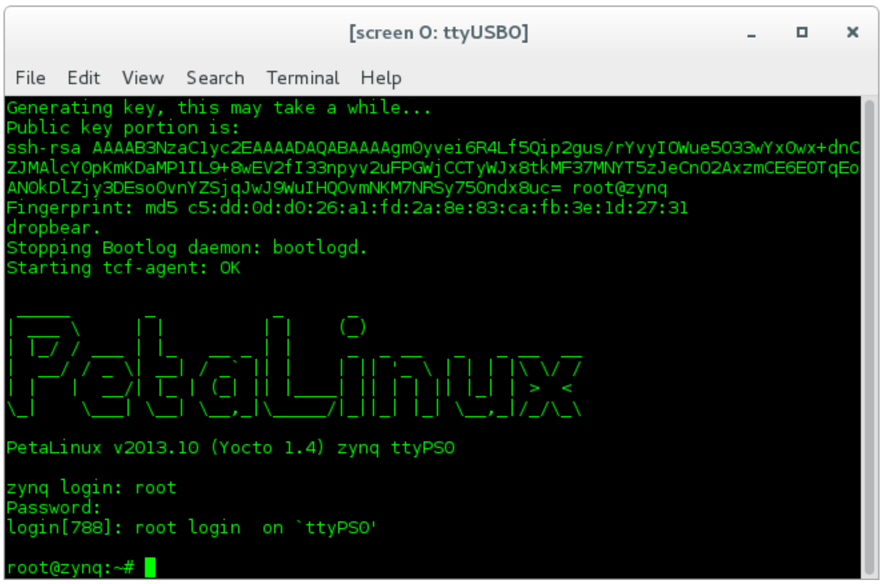

Figure 3: Successful Boot

Using the Matchstiq-Z1 in Network Mode

Every time the SDR is rebooted, the user is required to run the `mynetsetup.sh` script, 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`.

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

```
source /mnt/card/opencpi/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 4.

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

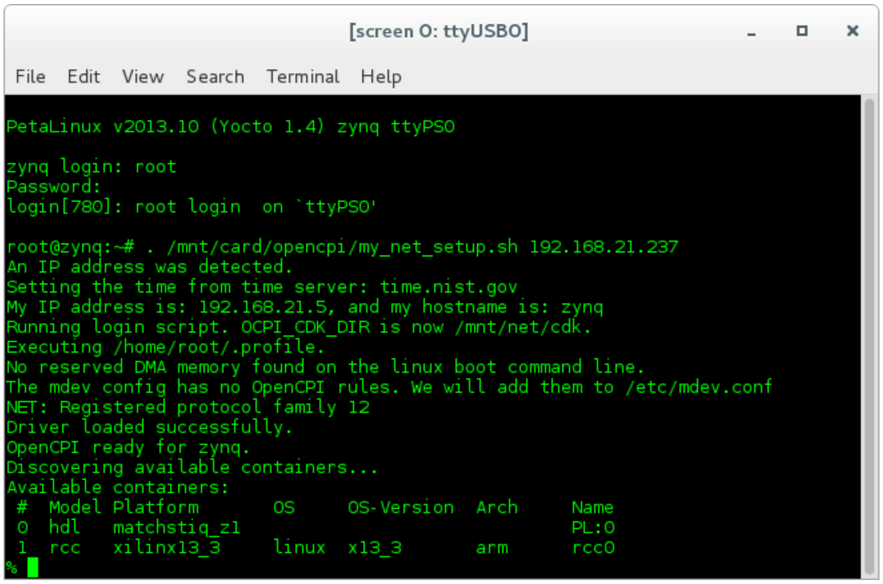

Figure 4: Successful Network Mode Setup

6.2 Standalone Mode

All artifacts for any applications or tests that need to be located on the SD card must be on the ATLAS partition 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 SDR platform.

Power cycle the Matchstiq-Z1

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

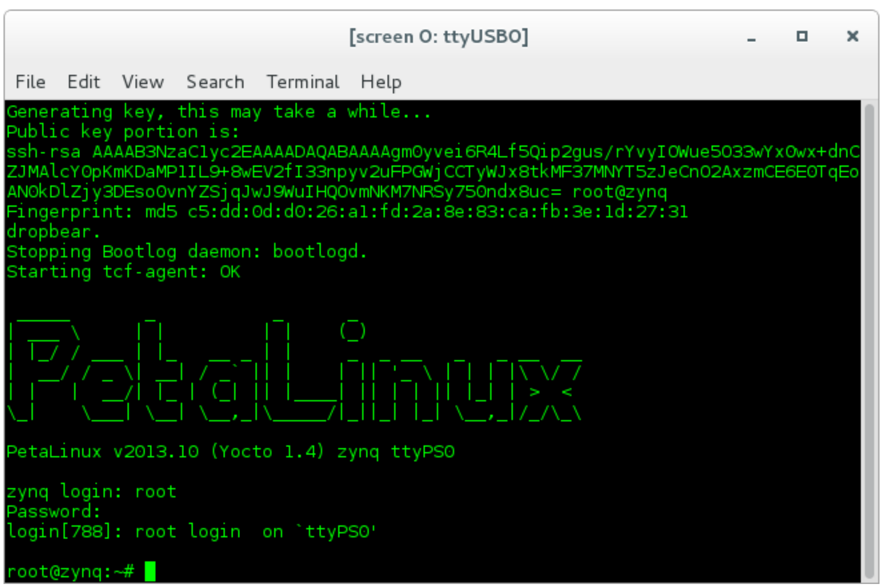

Figure 5: Successful Boot

Using the Matchstiq-Z1 in Standalone Mode

Every time the SDR 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 SDR 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:

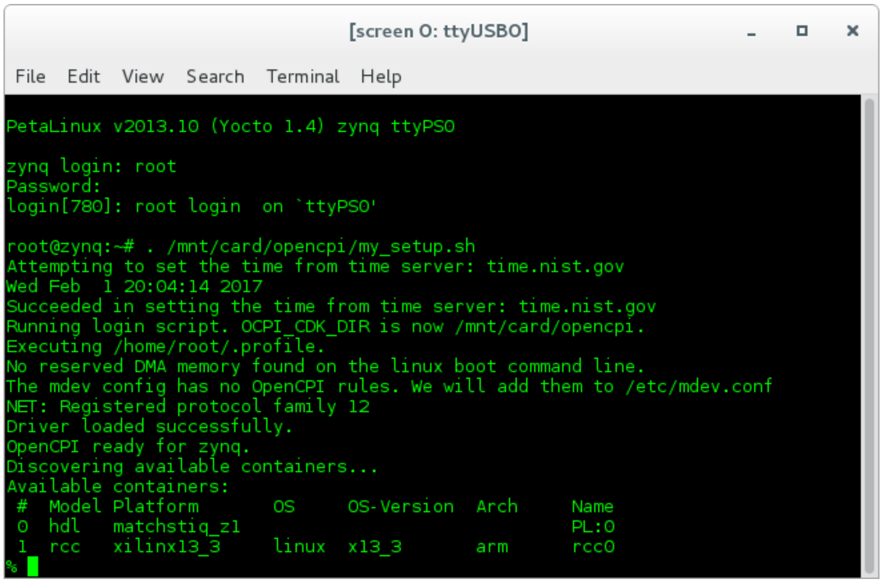

Figure 6: Successful Standalone Mode Setup

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.)

First the Core project needs to be built for the matchstiq_z1 platform. This is performed at the top level of the Core project running the command `ocpidev build --hdl-platform matchstiq_z1`. This will take about 45 minutes to complete.

Then the matchstiq_z1 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 platforms --hdl-platform matchstiq_z1`. 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 "matchstiq_z1" platform. It is located in `<Assets_Project>/hdl/assemblies/testbias/`. At the top level of the project run the following command: "`ocpidev build hdl assembly testbias --hdl-platform matchstiq_z1`". This should take about 15 minutes to complete. You can confirm that

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

it succeeded by locating the `*.bit.gz` file in `container-testbias_matchstiq_z1_base/target-matchstiq_z1/`.

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_assets/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 similar to Figure 7:

```
% ocpirun -v -t 1 -d -m bias=hdl bias.xml
Available containers are:  0: PL:0 [model: hdl os:  platform: matchstiq_z1], 1:
rcc0 [model: rcc os: linux platform: xilinx13_3]
Actual deployment is:
  Instance  0 file_read (spec ocpi.core.file_read) on rcc container 1: rcc0, usi
ng file_read in /mnt/net/cdk/../projects/core/exports/lib/components/rcc/linux-x
13_3-arm/file_read_s.so dated Thu Feb 15 10:24:06 2018
  Instance  1 bias (spec ocpi.core.bias) on hdl container 0: PL:0, using bias_vh
dl/a/bias_vhdl in /mnt/ocpi_assets/hdl/assemblies//testbias/container-testbias_m
atchstiq_z1_base/target-zynq/testbias_matchstiq_z1_base.bit.gz dated Thu Feb 15
21:25:28 2018
  Instance  2 file_write (spec ocpi.core.file_write) on rcc container 1: rcc0, u
sing file_write in /mnt/net/cdk/../projects/core/exports/lib/components/rcc/linu
x-x13_3-arm/file_write_s.so dated Thu Feb 15 10:24:06 2018
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 up to 1 seconds for application to finish
Application finished
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 7: Successful Network Mode Execution

To view the input file, run the following command: "`hexdump test.input | less`" and the file should look like Figure 8:

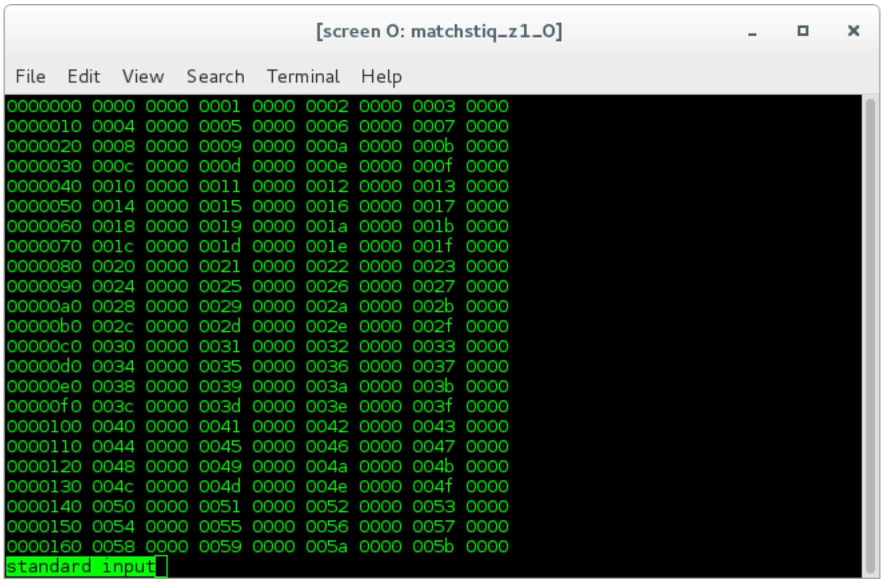

Figure 8: Expected Input

To view the output file, run the following command: "`hexdump test.output | less`" and the file should look like Figure 9:

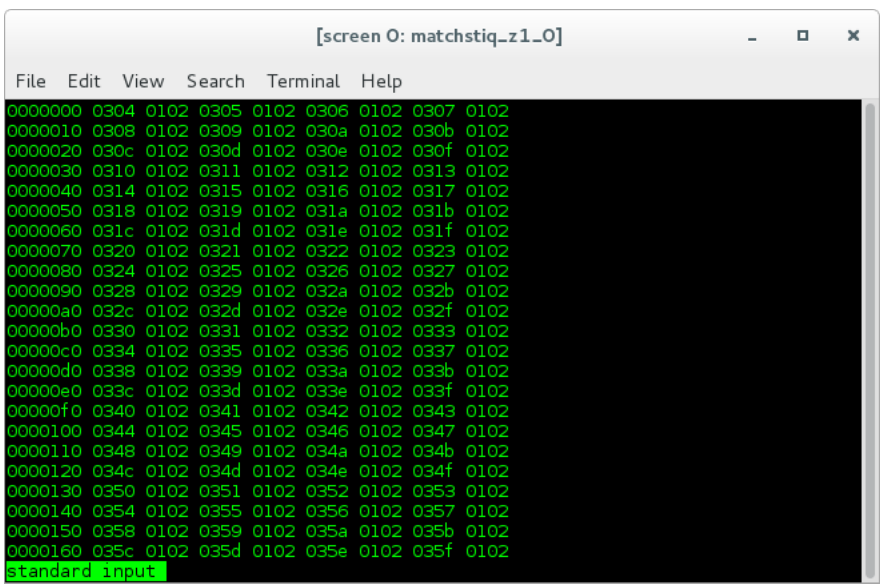

Figure 9: Expected Output

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 similar to Figure 10:

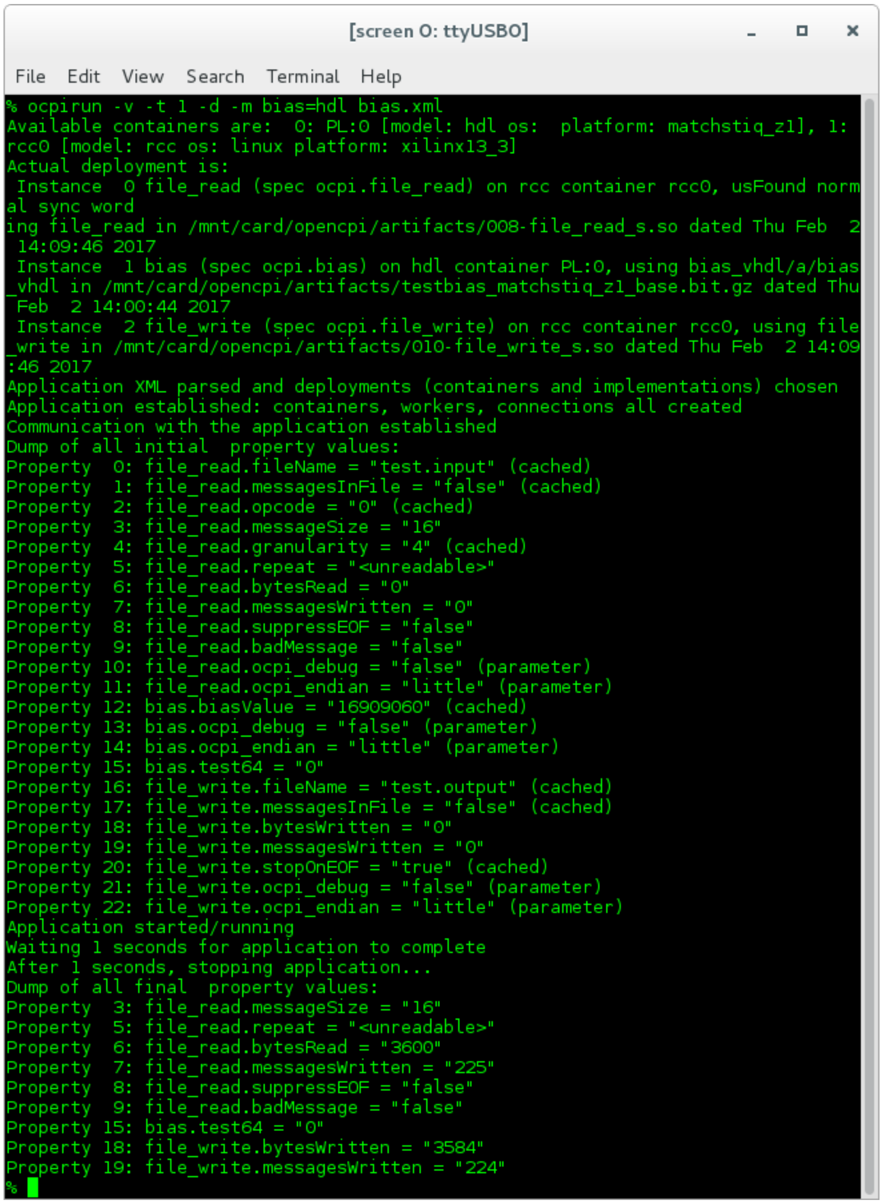

Figure 10: Successful Standalone Mode Execution

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

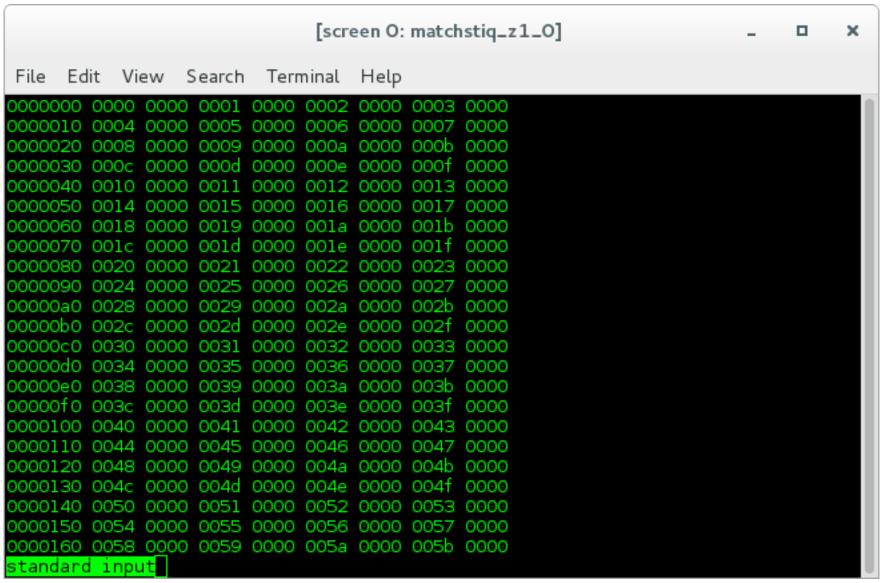

Figure 11: Expected Input

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

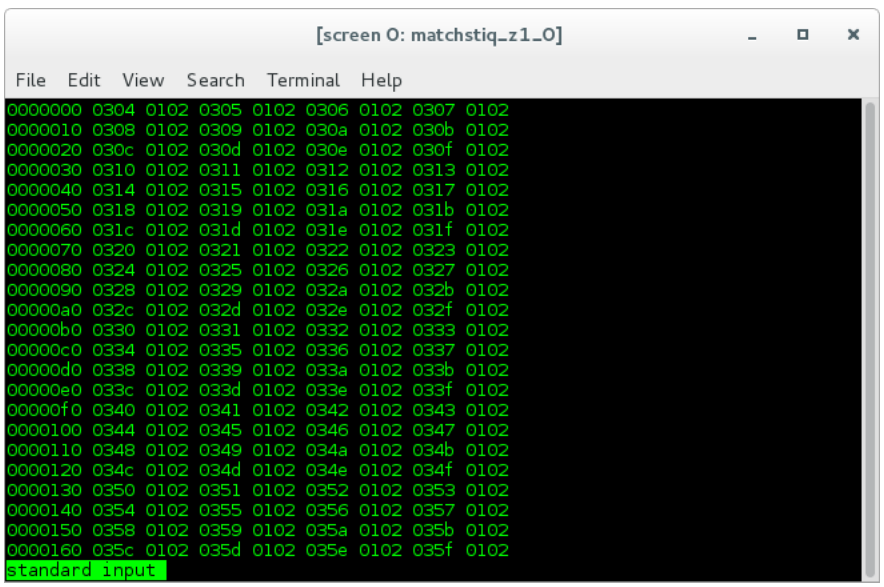

Figure 12: Expected Output

Appendices

A Intermittent Errors

Some tests have had "Segmentation Faults" or "Alignment Errors" in certain scenarios on the Z1. This seems to happen when both USB ports are used to simultaneously transmit a large amount of data, *e.g.* high log-level output to a USB serial console as well as NFS-mounted output files over a USB-to-Ethernet adapter. The default test setup avoids triggering this by limiting output that is fed to the user, but users should be aware of this issue if non-default test scenarios are attempted. If ssh is used to have all data routed through the USB-to-Ethernet adapter, this failure mode is avoided.

B Using ISE instead of Vivado with the Matchstiq-Z1

It is recommended that you use the default toolset (Xilinx Vivado) to build Matchstiq-Z1 bitstreams with OpenCPI. However, if you wish to use ISE instead, reference the README file in <ocpiassets>/hdl/platforms/matchstiq_z1/, and perform the following steps:

1. Modify the target part in <ocpiassets>/hdl/platforms/matchstiq_z1/matchstiq_z1.mk to use the ISE alias:

```
HdlPart_matchstiq_z1=xc7z_ise_alias_020-1-clg484
```

2. Export the ISE constraints files found in <ocpiassets/>hdl/platforms/matchstiq_z1/ise_constraints/ by modifying ExportFiles variable in <ocpiassets>/hdl/platforms/matchstiq_z1/Makefile:

```
ExportFiles=ise_constraints/matchstiq_z1.ucf ise_constraints/matchstiq_z1.ut matchstiq_z1.mk
```

C 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.