OpenCPI
ZCU111 Getting Started Guide

Version 1.4 - Geontech - Zynq UltraScale+ MPSoC/RFSoC Release

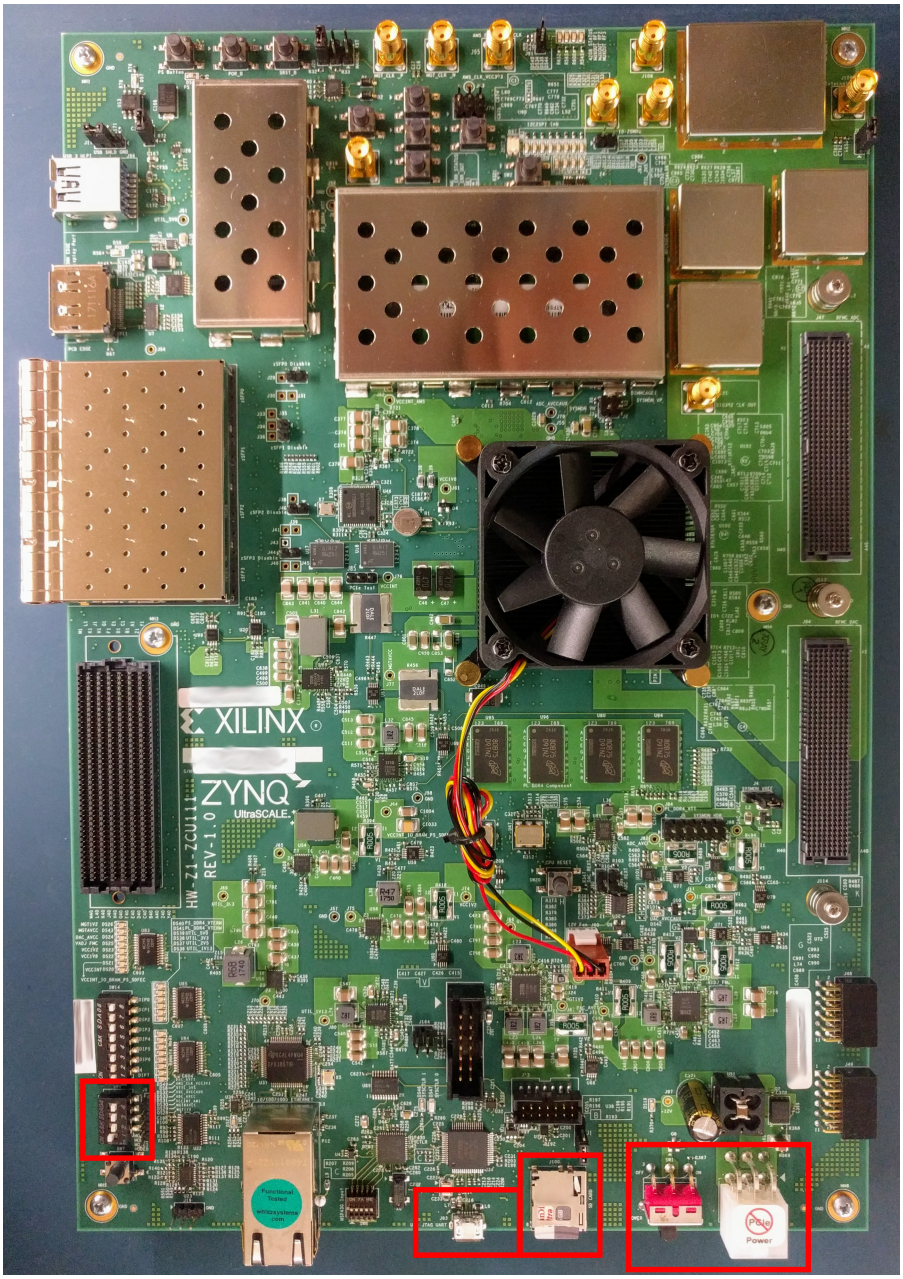

Figure 1: Top View (ZCU111)

Revision History

Revision	Description of Change	Date
release_1.4_zynq_ultra (Geontech custom release)	Initial Release of Zynq UltraScale+ Support in branch off of OpenCPI's release_1.4	1/2019

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	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	`http://opencpi.github.io/Overview.pdf`

Table 1: References

2 Overview

This document provides steps for configuring a factory provided Xilinx ZCU111 Evaluation Board with the OpenCPI runtime environment for executing applications, configuring a development system to build OpenCPI bitstreams targeting the *zcu111* platform, and examples of executing applications on the OpenCPI configured ZCU111.

3 Prerequisites

WARNING: OpenCPI development for the ZCU111 has only been verified using a CentOS7 development host.

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

RPM Name	Description
`opencpi-*1.4_zynq_ultra.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-*1.4_zynq_ultra.x86_64.rpm`	Additional header files and scripts for developing new assets as HDL and/or RCC.
`opencpi-sw-platform-xilinx18_2-*1.4_zynq_ultra.noarch.rpm`	Additional files necessary to build the framework targeting specific RCC/software platforms, independent of the final deployed hardware.
`opencpi-hw-platform-zcu111-*1.4_zynq_ultra.noarch.rpm`	Additional files necessary to build the framework targeting specific hard-ware platform "X" when running RCC platform "Y" ("Y" can be "no sw"). This RPM also includes hardware-specific SD Card images when applicable.

There is an IDE developed by the ANGRYVIPER team which is generally used for a more graphical development process in OpenCPI, but **the IDE has never been tested with the ZCU111 BSP**. [1]

Initial support for the Zynq UltraScale+ MPSoC/RFSoC was done in a fork of OpenCPI within a branch off of OpenCPI's `release_1.4`. So, the RPMs above must be acquired from Geon Technologies, LLC, and should have the label '`release_1.4_zynq_ultra`' to denote the branch they were built from.

Note that the RFSoC is an architecture that builds on MPSoC by adding RF functionality. This OpenCPI release supports the Zynq UltraScale+ (on both the MPSoC and RFSoC devices), but **none of the RF capabilities of the RFSoC**.

Appendix A details the process for building from source and generating RPMs from source. Building RPMs from source is useful for users that do not have access to the RPM files. If the framework is built and installed from source, it is possible for experienced users to use this source installation *instead* of an RPM installation. This is particularly useful for users who will need to make changes to the OpenCPI framework itself. If choosing to use a source-build installation, Section 3.1 can be skipped, and Appendix A can be referenced for project creation/cloning and registration.

3.1 Installation of required projects: *core*, *assets*, *bsp_zcu1xx* and *sw_xilinx18_2*

If using a source-build of the OpenCPI framework, follow the instructions in Appendix A.2 and skip this section.

If not completed already, the user must execute *ocpi-copy-projects*, accepting the default settings, to copy and register the *core* and *assets* projects from the `/opt/opencpi/projects` to the user's workspace. Reference

[1] Users wishing to use the IDE can following the instructions in Appendix B of ANGRYVIPER's *RPM_Installation_Guide* revision 1.4.

ANGRYVIPER's Getting Started Guide for details on *ocpi-copy-projects*. Although the projects are registered by *ocpi-copy-projects*, changes to the registry can be made via `ocpidev un/register project`. An example of *ocpi-copy-projects*' usage is below:

```
$ ocpi-copy-projects
...
$ ls ~/ocpi_projects
assets bsp_zcu1xx core sw_xilinx18_2
$ ocpidev show registry
Project registry is located at: /opt/opencpi/cdk/../project-registry
-------------------------------------------------------------------------------------------
| Project Package-ID         | Path to Project                                | Valid/Exists |
| ------------------         | ---------------                                | ------------ |
| ocpi.assets                | /home/<user>/ocpi_projects/assets              | True         |
| ocpi.core                  | /home/<user>/ocpi_projects/core                | True         |
-------------------------------------------------------------------------------------------
```

To build bitstreams and RCC workers for the ZCU111, two other projects will need to be cloned and registered as well:

```
$ cd ~/ocpi_projects;
$ git clone https://github.com/Geontech/sw_xilinx18_2.git --branch release_1.4_zynq_ultra;
$ ocpidev register project sw_xilinx18_2;
$ git clone https://github.com/Geontech/bsp_zcu1xx.git    --branch release_1.4_zynq_ultra;
$ ocpidev register project bsp_zcu1xx;
$ ocpidev show registry
Project registry is located at: /opt/opencpi/cdk/../project-registry
-------------------------------------------------------------------------------------------
| Project Package-ID         | Path to Project                                | Valid/Exists |
| ------------------         | ---------------                                | ------------ |
| ocpi.assets                | /home/<user>/ocpi_projects/assets              | True         |
| ocpi.core                  | /home/<user>/ocpi_projects/core                | True         |
| com.geontech.bsp.zcu1xx    | /home/<user>/ocpi_projects/bsp_zcu1xx          | True         |
| com.geontech.sw.xilinx18_2 | /home/<user>/ocpi_projects/sw_xilinx18_2       | True         |
-------------------------------------------------------------------------------------------
```

3.2 Vendor Software Setup

The platform that is expected to be used is Xilinx's ZCU111 Evaluation Board (*e.g.* `zcu111`). This OpenCPI-enabled platform provides the capability of deploying hardware and software workers while using Xilinx's 2018.2 distribution of Linux.

IMPORTANT: Use of OpenCPI on the ZCU111 requires Xilinx Vivado 2018.2 (including its SDK).

The synthesizers and cross-compilers required to build HDL and RCC Workers for the ZCU111 are present in Xilinx Vivado 2018.2. The instructions found in the *OpenCPI FPGA Vendor Tools Installation Guide* detail the process for installing older versions of Vivado, but can be still be useful as they include details for installation options and processes. This document assumes that the user has installed the appropriate versions of Vivado and the Xilinx SDK.

3.3 Building Required Projects

The *core*, *assets* and *bsp_zcu1xx* 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 `xilinx18_2` RCC Platform and the `zcu111` HDL Platform, but omit assemblies
2. Build `assets` for the `xilinx18_2` RCC Platform and the `zcu111` HDL Platform, but omit assemblies

3. Build the `bsp_zcu1xx` project for these same platforms
4. Build the `testbias` assembly from the `assets` project. This will be used later in this guide.

Once the HDL Platform is built in the BSP project, assemblies can be built for that HDL platform

First, enter the directory containing the projects:

When using an RPM installation:
```
$ cd /home/<user>/ocpi_projects/;
```
Note: replace "`<user>`" with your username in the command above.

When using a source installation:
```
$ cd <path-to-opencpi>/projects/;
```
Next, perform steps 1 through 4 to build the projects for `xilinx18_2` and `zcu111`:
```
$ ocpidev build -d core       --rcc-platform xilinx18_2 --hdl-platform zcu111 --no-assemblies;
$ ocpidev build -d assets     --rcc-platform xilinx18_2 --hdl-platform zcu111 --no-assemblies;
$ ocpidev build -d bsp_zcu1xx --rcc-platform xilinx18_2 --hdl-platform zcu111;
$ ocpidev build -d assets hdl assembly testbias         --hdl-platform zcu111;
```
Note: when using a source OpenCPI installation, the directory option "`-d bsp_zcu1xx`" will need to be changed to "`-d bsps/bsp_zcu1xx`" since the project is cloned one directory deeper as per Appendix A.2.

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

3.4 Hardware Setup and Requirements

- **Xilinx ZCU111 Evaluation Board**

  It is expected that this evaluation kit includes a power supply, micro-USB to USB cable and a micro SD card (4GB or larger).

  The micro-USB serial port on the ZCU111 labeled JTAG UART(Figure 1) can be used to access the serial connection with the processor.

- **Board Switch Settings**

  As mentioned in Table 2-4 of Xilinx's UG1271, set SW6 to 1110 for SD card boot mode. This means that the SW6's switch labeled 1 is 'ON' and 2, 3, and 4 are 'OFF'. See Figure 1.

- **Ethernet cable**: An Ethernet port is available on the ZCU111 (Figure 1) and is required when the Network mode (discussed later) environment is used. The OpenCPI BSP for the ZCU111 is configured for DHCP.

- **Access to a network which supports DHCP. (Network Mode)**

- **SD card**: As mentioned earlier, a 4GB or larger micro SD card should come with the board. The bootable SD card slot is located on the front of the unit (Figure 1) and ejects by gently pushing it in and releasing.

- **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. 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`" (Do not do this step unless you want the original contents back on the SD card.)

4.2 Generate the SD card image

This section describes how to use Yocto to generate a Xilinx 18.2 SD card image for OpenCPI. If you already have an OpenCPI SD card image (`opencpi-runtime-image-zcu111-zynqmp.wic`) for the ZCU111, you can move on to the next section.

1. Follow instructions at `https://www.yoctoproject.org/docs/2.0/yocto-project-qs/yocto-project-qs.html` to install Yocto's prerequisite RPMs
2. Follow the "Downloading", "Setup" and "Build" instructions at `https://github.com/Geontech/opencpi-manifest.git` to perform the following:
   (a) Downloading: clone all of the required Yocto layers and projects for use of Xilinx's 2018.2 distribution Linux with OpenCPI
   (b) Setup: setup the environment
      i. Copy OpenCPI's ZCU111 SD card files over to the `meta-opencpi` layer (from OpenCPI installation - details in Appendix A)
      ii. Setup the environment for `bitbake`
   (c) Building: Build the `opencpi-runtime-image` using `bitbake`

You should now be in the `build/` directory, and from there the SD card image can be found at:
`tmp/deploy/images/zcu111-zynqmp/opencpi-runtime-image-zcu111-zynqmp.wic`

4.3 Write image to SD card

This section is also covered in the `opencpi-manifest` mentioned above at `https://github.com/Geontech/opencpi-manifest.git`, but it is summarized here for convenience.

- 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`

- Write `opencpi-runtime-image-zcu111-zynqmp.wic` to the SD card, replacing $< SDcarddevicefilename >$ with the device file name determined in the previous step and ensuring that the path to the `wic` file is valid:

```
$ sudo dd if=<path-to>/opencpi-runtime-image-zcu111-zynqmp.wic \
of=<SD-card-device-file-name> bs=1M && sync;
```

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 ZCU111 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 board'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.

For both Network and Standalone mode, the following step is necessary:

1) Unplug the SD card from the host and plug it back in so its contents are mounted

Note: any commands creating or editing files on the SD card from the host will require use of "`sudo`". As an alternative, many of these file changes can actually be made on the ZCU111 itself after booting the board.

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 then modified per the user's requirements.

5.1.1 Network Mode

1) Make a copy of the default script for editing (the name of the script "`mynetsetup.sh`" is important for proper execution):

```
$ cp /run/media/<user>/root/home/root/opencpi/default_mynetsetup.sh \
     /run/media/<user>/root/home/root/opencpi/mynetsetup.sh;
```

2) Edit the copy

1. In `mynetsetup.sh`, uncomment the following lines which are necessary for mounting *core*, *assets*, and *bsp_ zcu1xx* projects:

   ```
   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
   mkdir -p /mnt/bsp_zcu1xx
   mount -t nfs -o udp,nolock,soft,intr $1:/home/<user>/ocpi_projects/bsp_zcu1xx /mnt/bsp_zcu1xx
   ```

2. Edit `/home/<user>/ocpi_projects/core` and `/home/<user>/ocpi_projects/assets` to reflect the paths to the *core*, *assets*, and *bsp_ zcu1xx* projects on the host, for example:

   ```
   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
   mkdir -p /mnt/bsp_zcu1xx
   mount -t nfs -o udp,nolock,soft,intr $1:/home/johndoe/ocpi_projects/bsp_zcu1xx /mnt/bsp_zcu1xx
   ```

5.1.2 Standalone Mode

In this mode, all OpenCPI artifacts that are required to run any application on the ZCU111 must be copied onto the SD card. Building the provided projects to obtain such artifacts is discussed in Section 3.3. Once the artifacts have been created, they must be copied to the SD card (*e.g.* `/run/media/<user>/root/home/root/opencpi/` or a new subdirectory there called `artifacts`) In general, any required `.so` (RCC workers), `.bin.gz` (hdl assemblies), and application XMLs or executables must be copied to the SD card.

1) Make a copy of the default script for editing (the name of the script "`mysetup.sh`" is important for proper execution):

```
$ cp /run/media/<user>/root/home/root/opencpi/default_mysetup.sh \
     /run/media/<user>/root/home/root/opencpi/mysetup.sh;
```

2) Edit the copy

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

3) Copy any additional artifacts (*i.e.* `*bin.gz` or `*.so` files) to SD card's `opencpi/artifacts/` directory

6 Development Host Setup - for Network Mode ONLY

WARNING: The ZCU111 in OpenCPI network mode has only been tested using a CentOS7 development host.

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)
<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)
<optional - host bsp 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
```

6.1.2 CentOS 7

From the host, install the necessary tools using yum:

```
% sudo yum install nfs-utils [2]
```

From the host, allow NFS past SELinux[3]:

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

From the host, if `firewalld` is enabled and running, 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`).

[2] `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

[3] 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.

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

/opt/opencpi XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
/home/<user>/ocpi_projects/core XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
/home/<user>>/ocpi_projects/assets XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt)
# If there is a BSP project for this platform:
/home/<user>/ocpi_projects/bsp_XXXX 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>/ocpi_projects/core XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=34)
/home/<user>/ocpi_projects/assets XX.XX.XX.XX/MM(rw,sync,no_root_squash,crossmnt,fsid=35)
# If there is a BSP project for this platform:
/home/<user>/ocpi_projects/bsp_XXXX 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 "sudo systemctl enable" commands may fail based on your package selection, but should not cause any problems.

Note: You will need to add the `com.geontech.bsp.zcu1xx` (directory name `bsp_zcu1xx`) project to your list of exports (in `/etc/exports.d/user_ocpi.exports`).

7 Configuring the runtime environment on the platform

This section details the runtime environment configuration steps for Network and Standalone modes on the ZCU111. Note that if you wish to switch from one mode to the other, run the following command on the ZCU111 and reboot:

```
$ rm /home/root/.profile;
```

7.1 Network Mode

1. Plug in an Ethernet cable to a network configured for DHCP
2. Ensure a micro-USB to USB cable is connected between the ZCU111's serial port and development host
3. Apply power to the ZCU111
4. Use a serial terminal application to establish a serial connection, for example:

   ```
   $ sudo screen /dev/ttyUSB1 115200
   ```

   Note: connecting the ZCU111 to a host via USB-UART will result in 4 `ttyUSB*` files in `/dev/`. The 0th one is NOT the one of interest here. Use the 1st one instead.
5. After a successful boot to PetaLinux, login to the system, using "**root**" for user name and password
6. Setup the OpenCPI environment on remote system

   Each time the board 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.[4]. The user must provide the network address of the development system to the script as its only argument:

   ```
   $ . /home/root/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 should output the following:

   ```
   An IP address was detected.
   My IP address is: 10.3.1.103, and my hostname is: zcu<xxx>-zynqmp
   Running login script. OCPI_CDK_DIR is now /mnt/net/cdk.
   Executing /etc/profile.d/opencpi-persist.sh
   No /etc/opencpi-release - assuming xilinx18_2 hardware
   No reserved DMA memory found on the linux boot command line.
   [   80.700634] opencpi: loading out-of-tree module taints kernel.
   [   80.707133] opencpi: dmam_alloc_coherent failed
   [   80.711608] opencpi: get_dma_memory failed in opencpi_init, trying fallback
   [   80.718552] opencpi: dmam_alloc_coherent failed
   [   80.723038] opencpi: get_dma_memory in request_memory failed, trying fallback
   [   80.730156] opencpi: if allocation failure occurs, see README for memmap configuration
   [   80.738074] NET: Registered protocol family 12
   Driver loaded successfully.
   OpenCPI ready for zynq_ultra.
   Discovering available containers...
   OCPI( 2:527.0378): HDL Device 'PL:0' responds, but the OCCP signature: magic: 0xffffffff00000000 (sb 0
   Available containers:
    #  Model Platform       OS     OS-Version  Arch     Name
    0  hdl   zcu<xxx>                                   PL:0
    1  rcc   xilinx18_2     linux  x18_2       arm64    rcc0
   ```

   Note: The line in this output regarding the OCCP signature is printed because the default bitstream on the board is not an OpenCPI one. Once an OpenCPI bitstream is loaded on the board, `ocpirun -C` will not include this output.

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

7.2 Standalone Mode

1. (Not required for this mode - see Item 6) Plug in an Ethernet cable to a network configured for DHCP
2. Ensure a micro-USB to USB cable is connected between the ZCU111's serial port and development host
3. Apply power to the ZCU111
4. Use a serial terminal application to establish a serial connection, for example:

   ```
   $ sudo screen /dev/ttyUSB1 115200
   ```

   Note: connecting the ZCU111 to a host via USB-UART will result in 4 `ttyUSB*` files in `/dev/`. The 0th one is NOT the one of interest here. Use the 1st one instead.
5. After a successful boot to PetaLinux, login to the system, using "**root**" for user name and password
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 board is booted, the OpenCPI environment must be setup. By sourcing the `mysetup.sh` script, the remote system's environment is configured for OpenCPI [5]. There are no arguments for this script.

   ```
   $ . /home/root/opencpi/mysetup.sh
   ```

   A successful run should output the following:

   ```
   Running login script. OCPI_CDK_DIR is now /home/root/opencpi.
   Executing /home/root/.profile
   No reserved DMA memory found on the linux boot command line.
   The mdev config has no OpenCPI rules. We will add them to /etc/mdev.conf
   NET: Registered protocol family 12
   Driver loaded successfully.
   OpenCPI ready for zynq_ultra.
   Discovering available containers...
   Available containers:
    #  Model Platform       OS     OS-Version  Arch     Name
    0  hdl   zcu<xxx>                                   PL:0
    1  rcc   xilinx18_2     linux  x13_4       arm      rcc0
   ```

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

8 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.

9 Run an Application

9.1 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 using the following command:

```
$ export OCPI_LIBRARY_PATH=/mnt/ocpi_assets/artifacts:/mnt/ocpi_core/artifacts
```

Run the application using the following command:

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

The output should be similar to:

```
Available containers are: 0: PL:0 [model: hdl os: platform: zcu<xxx>], 1: rcc0 [model: rcc os:
    ↪ linux platform: xilinx18_2]
Actual deployment is:[ 980.856510] opencpi: dmam_alloc_coherent failed
  Instance 0 file_read (spec ocpi.core.file_read) on rcc container 1: rcc0, using file_read in /
      ↪ mnt/net/projects/core/artifacts/ocpi.core.file_read.rcc.0.xilinx18_2.so dated Fri Jan
      ↪ 18 16:22:34 2019
[ 980.862362] opencpi: get_dma_memory in request_memory failed, trying fallback
[ 980.875023] opencpi: if allocation failure occurs, see README for memmap configuration
  Instance 1 bias (spec ocpi.core.bias) on hdl container 0: PL:0, using bias_vhdl/a/bias_vhdl in
      ↪  /mnt/net/projects/assets/artifacts/ocpi.assets.testbias_zcu<xxx>_base.hdl.0.zcu<xxx>.
      ↪ gz dated Mon Jan 28 11:40:53 2019
  Instance 2 file_write (spec ocpi.core.file_write) on rcc container 1: rcc0, using file_write
      ↪ in /mnt/net/projects/core/artifacts/ocpi.core.file_write.rcc.0.xilinx18_2.so dated Fri
      ↪ Jan 18 16:22:39 2019
Application XML parsed and deployments (containers and artifacts) 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 for application to finish (no time limit)
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"
```

Run the following command to view the input:

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

The output should look like the following:

```
0000000 0000 0000 0001 0000 0002 0000 0003 0000
0000010 0004 0000 0005 0000 0006 0000 0007 0000
0000020 0008 0000 0009 0000 000a 0000 000b 0000
0000030 000c 0000 000d 0000 000e 0000 000f 0000
0000040 0010 0000 0011 0000 0012 0000 0013 0000
0000050 0014 0000 0015 0000 0016 0000 0017 0000
0000060 0018 0000 0019 0000 001a 0000 001b 0000
0000070 001c 0000 001d 0000 001e 0000 001f 0000
0000080 0020 0000 0021 0000 0022 0000 0023 0000
0000090 0024 0000 0025 0000 0026 0000 0027 0000
00000a0 0028 0000 0029 0000 002a 0000 002b 0000
00000b0 002c 0000 002d 0000 002e 0000 002f 0000
00000c0 0030 0000 0031 0000 0032 0000 0033 0000
00000d0 0034 0000 0035 0000 0036 0000 0037 0000
00000e0 0038 0000 0039 0000 003a 0000 003b 0000
00000f0 003c 0000 003d 0000 003e 0000 003f 0000
0000100 0040 0000 0041 0000 0042 0000 0043 0000
0000110 0044 0000 0045 0000 0046 0000 0047 0000
0000120 0048 0000 0049 0000 004a 0000 004b 0000
0000130 004c 0000 004d 0000 004e 0000 004f 0000
0000140 0050 0000 0051 0000 0052 0000 0053 0000
0000150 0054 0000 0055 0000 0056 0000 0057 0000
0000160 0058 0000 0059 0000 005a 0000 005b 0000
```

Run the following command to view the output:

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

The output should look like the following:

```
0000000 0304 0102 0305 0102 0306 0102 0307 0102
0000010 0308 0102 0309 0102 030a 0102 030b 0102
```

```
0000020 030c 0102 030d 0102 030e 0102 030f 0102
0000030 0310 0102 0311 0102 0312 0102 0313 0102
0000040 0314 0102 0315 0102 0316 0102 0317 0102
0000050 0318 0102 0319 0102 031a 0102 031b 0102
0000060 031c 0102 031d 0102 031e 0102 031f 0102
0000070 0320 0102 0321 0102 0322 0102 0323 0102
0000080 0324 0102 0325 0102 0326 0102 0327 0102
0000090 0328 0102 0329 0102 032a 0102 032b 0102
00000a0 032c 0102 032d 0102 032e 0102 032f 0102
00000b0 0330 0102 0331 0102 0332 0102 0333 0102
00000c0 0334 0102 0335 0102 0336 0102 0337 0102
00000d0 0338 0102 0339 0102 033a 0102 033b 0102
00000e0 033c 0102 033d 0102 033e 0102 033f 0102
00000f0 0340 0102 0341 0102 0342 0102 0343 0102
0000100 0344 0102 0345 0102 0346 0102 0347 0102
0000110 0348 0102 0349 0102 034a 0102 034b 0102
0000120 034c 0102 034d 0102 034e 0102 034f 0102
0000130 0350 0102 0351 0102 0352 0102 0353 0102
0000140 0354 0102 0355 0102 0356 0102 0357 0102
0000150 0358 0102 0359 0102 035a 0102 035b 0102
0000160 035c 0102 035d 0102 035e 0102 035f 0102
```

9.2 Run an Application in 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 `/home/root/opencpi/xilinx18_2/artifacts`. After running `mysetup.sh`, navigate to `/home/root/opencpi/xml`. Run the application using the following command:

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

The output should be similar to the output shown in Section 9.1.

Run the following commands to view the input and output, and reference Section 9.1 for the expected results:

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

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

10 Running Reference Applications

Now that you have set up OpenCPI and the ZCU111 board, you can run the FSK reference application found in `assets/applications` in its testbench/file-read-write mode. This assumes that the build commands in Section 3.3 were run successfully.

To build the `fsk_filerw` assembly and `FSK` application, run the following from the host:
`$ ocpidev build hdl assembly fsk_filerw --hdl-platform zcu111;`
`$ ocpidev build application FSK --rcc-platform xilinx18_2;` To run the application on the ZCU111, the `OCPI_LIBRARY_PATH` must first be set to point to all build artifacts needed for this application. For more information regarding the required `OCPI_LIBRARY_PATH` and other runtime requirements, reference the *FSK_App_Getting_Started_Guide.pdf* and *FSK_app.pdf* documents. The application can be run `filerw` mode on the ZCU111:

```
$ ./target-xilinx18_2/FSK filerw
```

Appendices

A Building OpenCPI, its RPMs and ZCU111 SD card contents from source

A.1 Building OpenCPI for centos7 and xilinx18_2 cross-compilation

In order to build the framework from source, you must first clone Geontech's OpenCPI fork and checkout the `release_1.4_zynq_ultra` branch:

```
$ git clone https://github.com/Geontech/opencpi.git --branch release_1.4_zynq_ultra;
```

Next, enter the OpenCPI repository and install the framework for the `centos7` host (make sure you have no `OCPI_*` environment variables set before this step):

```
$ cd opencpi;
$ unset OCPI_PREREQUISITES_DIR OCPI_CDK_DIR OCPI_TOOL_DIR OCPI_TOOL_ARCH OCPI_TOOL_PLATFORM \
        OCPI_TOOL_OS OCPI_TOOL_OS_VERSION OCPI_TOOL_PLATFORM_DIR;
$ ./scripts/install-opencpi.sh;
```

Set up your environment (this can optionally be added to your `~/.bashrc`):

```
$ source ./cdk/opencpi-setup.sh -r;
```

This will set various environment variables such as `OCPI_CDK_DIR`.

A.2 Cloning and registering BSP and SW projects within the source repository

Next, you must clone the `bsp_zcu1xx` and `sw_xilinx18_2` repositories into OpenCPI's `projects/bsps/` directory and register them:

```
$ cd projects/bsps;
$ git clone https://github.com/Geontech/sw_xilinx18_2.git --branch release_1.4_zynq_ultra;
$ ocpidev register project sw_xilinx18_2;
$ git clone https://github.com/Geontech/bsp_zcu1xx.git    --branch release_1.4_zynq_ultra;
$ ocpidev register project bsp_zcu1xx;
$ ocpidev show registry;
Project registry is located at: <path-to>/opencpi/cdk/../project-registry
-------------------------------------------------------------------------------------------
| Project Package-ID         | Path to Project                                | Valid/Exists |
| ------------------         | ---------------                                | ------------ |
| ocpi.assets                | <path-to>/opencpi/projects/assets              | True         |
| ocpi.core                  | <path-to>/opencpi/projects/core                | True         |
| com.geontech.bsp.zcu1xx    | <path-to>/opencpi/projects/bsps/bsp_zcu1xx     | True         |
| com.geontech.sw.xilinx18_2 | <path-to>/opencpi/projects/bsps/sw_xilinx18_2  | True         |
-------------------------------------------------------------------------------------------
$ cd -; # return to top level
```

A.3 Building OpenCPI for xilinx18_2 cross-compilation

Run the following to build and install the framework for `xilinx18_2` cross-compilation:

```
$ ./scripts/install-opencpi.sh xilinx18_2;
```

A.4 Populate the SD card directory for the ZCU111

After building the framework for cross-compilation with `xilinx18_2`, the ZCU111 SD card contents can be staged for deployment:

```
$ make deploy Platform=xilinx18_2;
$ make Platform=zcu111;
$ make deploy Platform=zcu111;
```

Note: it is okay if some "`mv`" or "`rmdir`" commands fail towards the end of "`make deploy`" involving the "`tmp`" directory.

There should now be a directory called `cdk/zcu111/zcu111-deploy/sdcard-xilinx18_2/opencpi` which is the directory to copy over to the "`meta-opencpi`" layer (or just SD card root partition's `/home/root/`) in the steps in Section 4.2.

A.5 Generate RPMs (`centos7`, `xilinx18_2` and `zcu111`)

If you wish to generate RPMs with this OpenCPI installation packaged up, the following commands will generate the necessary RPMs:

```
$ make rpm;
$ make rpm Platform=xilinx18_2;
$ make rpm Platform=zcu111;
```

RPMs can now be found in `packages/target-*` for each target platform (`centos7`, `xilinx18_2` and `zcu111`).