# RPM Installation Guide

Version 1.3

***Revision History***

| Revision | Description of Change | Date |
|---|---|---|
| v1.0 | Initial Release | 2/2016 |
| v1.1 | Updated for OpenCPI Release 1.1 | 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

## List of Tables

## 1 References

This document assumes a basic understanding of the Linux command line environment. It does not require a working knowledge of OpenCPI. However, it is recommended that the user read the *Getting Started* document (up to the "Installation of OpenCPI" section) or reference the *Acronyms and Definitions* document for various terms used within.

Table 1: References

| **Title** | **Published By** | **Link** |
|---|---|---|
| Getting Started | ANGRYVIPER Team | `Getting_Started.pdf` |
| Acronyms and Definitions | ANGRYVIPER Team | `Acronyms_and_Definitions.pdf` |
| Overview | OpenCPI | `https://goo.gl/RskxiV` |
| Installation Guide[1] | OpenCPI | `https://goo.gl/VWo2jX` |
| Component Development Guide | OpenCPI | `https://goo.gl/zBwIe0` |
| RCC Development Guide | OpenCPI | `https://goo.gl/0ix1E0` |
| HDL Development Guide | OpenCPI | `https://goo.gl/0VmRhI` |
| FPGA Vendor Tools Installation Guide | ANGRYVIPER | `FPGA_Vendor_Tools_Installation_Guide.pdf` |
| Managing Software with `yum` | CentOS Project | `https://www.centos.org/docs/5/html/yum/` |
| CentOS Deployment Guide: Useful `yum` commands (*e.g.* `yum localinstall`) | CentOS Project | `https://www.centos.org/docs/5/html/5.2/Deployment_Guide/s1-yum-useful-commands.html` |

[1]The RPM installation process is quite different from the process explained in the OpenCPI Installation Guide, but the OpenCPI Installation guide has applicable post-installation information for PCI-based boards, etc.

## 2 Document Overview

This document describes how to install OpenCPI on a development host via RPMs. The host installation allows for local software-based execution of OpenCPI applications and components, cross-building for non-x86 platforms, simulation of HDL, and, when available, hardware testing. It is recommended that users install from RPMs.

To allow for RPM packaging, the build system was significantly modified compared to the original OpenCPI Build System, and the changes are still being resolved. If you are working in a branch on GitHub that includes a top-level script `ocpi-configure`, and you do *not* want to use the RPM files, consult Appendix C.

The default host installation platform for OpenCPI development is CentOS 6 or CentOS 7 Linux x86_64 (64-bit). Other Linux variants and 32-bit systems have been used successfully, but this document expects the OS to be CentOS 7. Development hosts can either be actual physical systems or virtual machine installations.

This document assumes that CentOS is already installed, has the necessary packages installed for software compilation, and proper administrative privileges have been established.

Additional installation options exist for other target processors and technologies such as the Xilinx Zynq SoC (with ARM processor cores and FPGA resources). Preference when targeting non-x86 architectures is given to *cross-building*, rather than self-hosting development. This limits the complexity of installing tools on different development hosts.

Installation of OpenCPI is completed in the following steps:

1. **Section 3**: Acquiring the OpenCPI framework
2. **Section 5**: Installing the OpenCPI prerequisites
3. **Section 6**: Installing the OpenCPI framework
4. **Section 7**: Setting up the OpenCPI environment
5. **Section 8**: Testing the OpenCPI installation

These steps result in a development system with tooling and runtime software ready to support development and native execution of OpenCPI components and applications.

## 3 Acquiring the OpenCPI framework

Currently, the ANGRYVIPER Team releases DVD-Rs containing the RPMs and PDF documentation of the OpenCPI framework. These should be available on `github.io`.

## 4 Installing FPGA vendor tool prerequisites

For HDL bitstream generation, OpenCPI requires vendor-provided tools (*e.g.* Xilinx Vivado, Xilinx ISE, Altera Quartus) for FPGA bitstream compilation, as well as various other operations. Refer to `FPGA_Vendor_Tools_Installation_Guide.pdf` for instruction in installing and configuring these tools for use with OpenCPI.

## 5 Installing OpenCPI third-party prerequisites

OpenCPI development is dependent on various external packages from the Open Source community. These include:

1. Analog Devices' AD9361 no-OS Library
2. GNU Multiple Precision Arithmetic Library (GMP)
3. Google Test (GTEST)
4. Lempel-Ziv-Markov chain algorithm (LZMA/XZ)
5. Liquid DSP
6. PatchELF

These packages have OpenCPI-specific patches and are provided as RPMs. This packaging ensures they will not conflict with other[1] installed copies by using a nonstandard installation location of `/opt/opencpi/prerequisites`. Appendices A.1 – A.6 contain a synopsis of the changes that were made to these packages.

These prerequisites are packaged into multiple RPMs due to different usage configurations. Not all are required for a standard development install; consult Table 2.

Table 2: RPM Prerequisite Descriptions

| **RPM Name** | **Description** |
|---|---|
| AD9361 No-OS Library | |
| `ocpi-prereq-ad9361-*.rpm`[1] | Header file(s) and software library: ADI's No-OS library for AD9361 command/control. |
| `ocpi-prereq-ad9361-platform-*.noarch.rpm`[2] | Cross-compiled platform-specific library. |
| GNU Multiple Precision Arithmetic Library | |
| `ocpi-prereq-gmp-6.1.1-*.rpm`[1] | Header file(s) and software library: GNU Multiple Precision Arithmetic Library (GMP). |
| `ocpi-prereq-gmp-platform-*.noarch.rpm`[2] | Cross-compiled platform-specific library. |
| Google Test | |
| `ocpi-prereq-gtest-1.7.0-.rpm`[1] | Header file(s) and software library: Google's C++ test framework. |
| `ocpi-prereq-gtest-platform-*.noarch.rpm`[2] | Cross-compiled platform-specific library. |
| Liquid DSP | |
| `ocpi-prereq-liquid-1.2.0-*.rpm`[3] | Header file(s) and software library: Liquid DSP library. |
| `ocpi-prereq-liquid-platform-*.noarch.rpm`[3] | Cross-compiled platform-specific library. |
| LZMA / XZ | |
| `ocpi-prereq-xz-5.2.2-*.rpm`[1] | Header file(s) and software library: "xz" utils, a set of FOSS lossless data compressors (formerly known as "lzma"). |
| `ocpi-prereq-xz-platform-*.noarch.rpm`[2] | Cross-compiled platform-specific library. |
| Miscellaneous RPMs | |
| `ocpi-prereq-build_support-inode64-*.rpm`[1] | Shim library to allow 32-bit compilers to compile code on 64-bit file systems |
| `ocpi-prereq-patchelf-0.9-*.rpm`[1] | A utility for modifying existing ELF executables and libraries. |

[1]Only required for development
[2]Always required for development and deployment using matched platform name
[3]Required to build some components in `ocpi.assets` but not required for base OpenCPI usage; provided as a courtesy for RCC Workers

For simplicity, it is recommended that the user installs *all* available prerequisite RPMs; this may be completed using `yum`:

```
% sudo yum localinstall <location of prerequisite RPMs>/*rpm
```

[1]OS vendor, EPEL, other third-party-packagers, etc.

## 6 Installing OpenCPI framework

The ANGRYVIPER Team's recommended installation method for development is through the use of RPMs. The framework can be built from source for a development host, but is not recommended. In either case, the prerequisites described in section 5 must be installed prior to the following section.

The ANGRYVIPER Team provides RPMs to their direct customers and end users can consult Appendix B for instructions on re-building them.

### Understanding OpenCPI RPM naming convention

OpenCPI's RPM naming follows that of the Red Hat Package Manager recommendations of `<name>-<version>-<release>.<dist>.<architecture>.rpm` where:

1. *name* is the name describing the packaged software
2. *version* is the version of the packaged software
   (a) version following the Major.Minor.Sub-minor naming schema
3. *release* is the number of times this version of software has been packaged
   (a) this number is independent of the version
4. *dist* is the OS distribution that the package is built for (*e.g.* `.el7.centos`)
5. *architecture* is shorthand name describing the type of hardware the packaged software is to be installed on
6. "*devel*" is sometimes appended to the package name to indicate development RPMs which are required for building from source

### When to Install

It is recommended that the user install these packages *before* additional tools, e.g. ModelSim, because the `devel` subpackage forces the installation of otherwise-hidden dependencies, *e.g.* 32-bit X11 libraries for ModelSim.

## 6.1 Installing OpenCPI from RPMs

After installation of the OpenCPI prerequisite RPMs[2], the main RPMs may be installed. Again, as with the prerequisites, it is recommended that the user installs all available packages whenever possible. If limited by available disk space, Table 3 can be used to help determine which of the packages should be installed based upon the intended use of the target machine.

Within OpenCPI, there are two types of implementations, called *Workers*, that are used in this framework: Resource-Constrained C Language (RCC) Workers and Hardware Description Language (HDL) Workers. RCC Workers are written using either C or C++ and are designed for either x86 or ARM architecture, while HDL Workers are written in VHDL and are designed for Field Programmable Gate Arrays (FPGAs). For further details regarding RCC and HDL Workers see the *OpenCPI RCC Development Guide* and the *OpenCPI HDL Development Guide*.

Table 3: Main RPM Decision Guide

| | Runtime RCC Host | Runtime HDL Host | RCC-Only Development (x86 RCC exclusive) | RCC/HDL Development (x86 RCC, non-hybrid[1] FPGA HDL) | RCC/HDL Development (Targeting non-x86 HW/SW platform) |
|---|---|---|---|---|---|
| `angryviper-ide...rpm` | | | ✓ | ✓ | ✓ |
| `opencpi-...rpm` | ✓ | ✓ | ✓ | ✓ | ✓ |
| `opencpi-devel...rpm` | | | ✓ | ✓ | ✓ |
| `opencpi-driver...rpm` | | ✓ | | ✓ | ✓ |
| `opencpi-interface-python...rpm`[2] | ✓ | ✓ | ✓ | ✓ | ✓ |
| `opencpi-project-assets...rpm` | | | ✓ | ✓ | ✓ |
| `opencpi-project-bsp...rpm`[3] | | | ✓ | ✓ | ✓ |
| `opencpi-*-platform...rpm` | | | | | ✓ |

[1] "Non-hybrid" meaning a standalone FPGA *without* an integrated processor, *e.g.* Xilinx ML605.
[2] Only required when running or developing ACIs written in python
[3] BSP RPMs may not be provided with the standard/basic RPMs, but represent a placeholder for RPMs providing Board Support Package Projects. There are certain BSPs which are located in the `ocpi.assets` Project and therefore do not require their own separate BSP RPMs.

The main RPMs each have specific usage. Table 4 outlines what each of the main RPMs are used for.

[2] All main and prerequisite RPMs can be simultaneously installed.

Table 4: Main RPM Descriptions

| Main RPMs | Description |
|---|---|
| `angryviper-ide-*.x86_64.rpm` | The ANGRYVIPER IDE (Eclipse with plugins). |
| `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-debuginfo-*.x86_64.rpm` | Debug symbols needed to debug the framework. |
| `opencpi-devel-*.x86_64.rpm` | Additional header files and scripts for developing new assets as HDL and/or RCC. |
| `opencpi-driver-*.noarch.rpm` | OpenCPI driver. Once installed, any subsequent kernel updates will cause the driver to be built automatically on restart. |
| `opencpi-hw-platform-*.noarch.rpm` | Additional files necessary to build the framework targeting specific hardware platforms. Automatically require the appropriate `sw-platform` package. |
| `opencpi-interface-python*.x86_64.rpm` | This package includes SWIG bindings to allow Python-based top-level ACI applications. |
| `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-project-bsp*.noarch.rpm` | A `*.bsp.*` Project contains a Board Support Package for a particular platform, e.g. RCC/HDL Platform Support, Device Workers, etc. There are certain BSPs which are located in the `ocpi.assets` Project and therefore do not require their own separate BSP RPMs. |
| `opencpi-sw-platform-*.noarch.rpm` | Additional files necessary to build the framework targeting specific software platforms. |

Installation may be completed using yum and the following command:

```
% sudo yum localinstall <location of main RPMs>/*rpm
```

### 6.2 Installing OpenCPI from Source

See Appendix C.

## 7 Setting up the OpenCPI Environment

#### Notes about installing HDL simulator(s) and/or compiler(s)

Before attempting the next section, ensure that the desired HDL simulators and HDL tools are installed. Installation of these is outside the scope of this document.

Keep note of where the *license files* are, the *version number* of the tools, and *where the tools are installed*, as they will be needed to configure the required environment variables.

## 7.1 The `opencpi` Group

At this point, certain users should be added to the `opencpi` group. When a user creates a Project, it is likely that the Project should be `registered`. Registering a Project allows other users and Projects to access its assets. The default Registry on an RPM-configured system is located at `/opt/opencpi/project-registry`. In order for a user to register Projects in this default location, the user will need to be a member of the `opencpi` group. To add a user to the `opencpi` group, run the following command:

```
 % sudo usermod -aG opencpi <username>
```

If this command is run as user `<username>`, the user will need to log out and back in to apply this change.

Note that users can use a non-default Project Registry. For more information on this, please visit the *OpenCPI Component Development* document or the *Getting Started Guide*.

## 7.2 Setup Environment

Setting up the environment when installing from RPM requires root privileges. Navigate to `$(OCPI_CDK_DIR)/env.d` and notice the following example scripts:

- `altera.sh.example`
- `modelsim.sh.example`
- `site.sh.example`
- `xilinx.sh.example`

Every time a new shell is opened, all `*.sh` files in `/opt/opencpi/cdk/env.d` are executed, and all `*.sh.example` files in `/opt/opencpi/cdk/env.d` are *ignored*. To enable a script for execution, the name of the script must be changed so that the `.example` suffix is removed. A simple demonstration is below:

```
 % sudo cp altera.sh.example altera.sh
```

Now `altera.sh` will execute every time a new shell is opened.

If using the Altera tools, the `altera.sh` will need to be created and the variables `OCPI_ALTERA_DIR`, `OCPI_ALTERA_VERSION`, and `OCPI_ALTERA_LICENSE_FILE` must be defined in `altera.sh`. The `altera.sh` script also calls another script to set up the rest of the variables needed for the Altera tools.

If using the Modelsim tools, the `modelsim.sh` will need to be created and the variables `OCPI_MODELSIM_DIR` and `OCPI_MODELSIM_LICENSE_FILE` must be defined in `modelsim.sh`.

If using the Xilinx tools, the `xilinx.sh` will need to be created and the variable `OCPI_XILINX_LICENSE_FILE` must be defined in `xilinx.sh`. If using an installation of Xilinx Vivado that was *not* installed in the default `/opt` directory then the variable `OCPI_XILINX_VIVADO_DIR` must be defined in `xilinx.sh`. If using a version other than the most recent one installed in that location, then the variable `OCPI_XILINX_VIVADO_VERSION` must be defined in `xilinx.sh`. If using an installation of Xilinx ISE that was *not* installed in the default `/opt` directory then the variable `OCPI_XILINX_DIR` must be defined in `xilinx.sh`. If not using the 14.7 version of ISE, then the variable `OCPI_XILINX_VERSION` must be defined in `xilinx.sh`. The `xilinx.sh` script also calls another script to set up the rest of the variables needed for the Xilinx tools. See the *FPGA Vendor Tools Installation Guide* for more information on Xilinx license setup.

The script `site.sh.example` has been provided as an example central location where any other variables can be defined globally. *Remember that the names of the scripts do not matter, only the `*.sh` extension.* More configuration variables can be found in the *Getting Started Guide*.

Once all the desired scripts have been created and edited, open a new shell and check to see that the environment is now set up.

### 7.3 Removing OpenCPI RPMs

In the event that the OpenCPI RPM needs to be uninstalled, or reinstalled, the best way to remove the OpenCPI RPM is to use `yum` to erase the RPMs from Table 4 as seen below:

```
% sudo yum erase <RPM name>
```

## 8 Testing the OpenCPI installation

To verify the OpenCPI installation run the following command from a new terminal[3]:

```
% test-opencpi-rpm
```

A successful install will output "All tests passed." at the end of the test.

[3]This command is only available if the `-devel` package was installed.

# Appendices

## A Prerequisite Modifications

This section provides an overview of changes made to various Free and Open Source software required to be implemented within the Framework. Exact `diff` files and various sources are available upon request. Implied with every list are patches to the build configuration to allow the library's final installation location to be under the `/opt/opencpi/prerequisite` tree, along with cross-compilation targeting various platforms.

### A.1 Analog Devices' AD9361 no-OS Library

Source: `https://github.com/analogdevicesinc/no-OS`

- Patches to allow older compilers to compile (missing `stdint.h` includes)
- Move some top-level structs from `common.h` into `ad9361.h` to limit scope of items, *e.g.* "`struct clk`"

### A.2 GNU Multiple Precision Arithmetic Library (GMP)

Source: `https://ftp.gnu.org/gnu/gmp/gmp-6.1.1.tar.xz`

- A wrapper file, `gmp-mparam.h`, replaces the original one. This wrapper file is provided by Red Hat and used in their RPM packaging of `gmp`.

### A.3 Google Test (GTEST)

Source: `https://github.com/google/googletest/archive/release-1.7.0.zip`

- Removed most tests and examples
- Removed non-`gcc` source

### A.4 Liquid DSP

Source: `https://github.com/jgaeddert/liquid-dsp.git` (tied to specific git hash)

- Nothing beyond configuration modifications noted above

### A.5 Lempel-Ziv-Markov chain algorithm (LZMA/XZ)

Source: `https://github.com/xz-mirror/xz/releases/download/v5.2.2/xz-5.2.2.tar.xz`

- CentOS 6: Lower build environment expectations to `autoconf 2.63`, `automake 1.11`, and `gettext 0.17`
- CentOS 7: Nothing beyond configuration modifications noted above

### A.6 PatchELF

Source: `https://github.com/NixOS/patchelf.git` (tied to git tag `0.9`)

- Add new command-line option (`--ocpi-fix-soname`) that removes the suffix `_s.so` in an ELF's `SONAME` and replaces it with `.so` using only string manipulations (no sections are created or modified).

# B Appendix - Building OpenCPI RPMs

This section does *not* cover fundamentals such as basic Linux usage, cloning the `git` repository, etc.

## B.1 Prerequisites to Building

This document assumes that you already have:

- A working CentOS-based system (or docker container / virtual machine)

  Fully updated with `yum upgrade`[4]

- A `git` clone of the repository
- The GCC suite (e.g. the `gcc-c++` RPM)
- GNU Make (`make`)
- A cross-compiler for target systems

  This document uses Xilinx's EDK

- Certain packages already installed that are called out in each RPM's specfile's `BuildRequires` tag.

  This can usually be resolved with `sudo yum install missingpkg` or the more advanced `repoquery --whatprovides /path/to/missing/file`.

  The build system is pretty good about indicating what is missing (cf. B.1.1 for example)

  Some require the EPEL Repository (provided by the `epel-release` RPM)

- Both 32- and 64-bit versions of `glibc-static` and `glibc-devel` installed

  `sudo yum install glibc{,-static,-devel}{,.i686}`

### B.1.1 Example of Required Packages

As an example, to build the *prerequisite RPMs* on a fresh CentOS 7 docker container, the build actually took *eight* iterations to get all required packages installed to build all for the host:

```
$ yum install -y git make
$ make prereq
$ yum install -y gcc mlocate rpm-build
$ make prereq
$ yum install -y glibc-devel.i686
$ make prereq
$ yum install -y libgcc.i686
$ make prereq
$ yum install -y ed
$ make prereq
$ yum install -y autoconf automake libtool
$ make prereq
$ yum install -y gcc-c++
$ make prereq
$ yum install -y gettext-devel
$ make prereq # Success!
```

[4] Skipping this may result in "multilib" errors in `yum` due to installing the latest 32-bit version of a tool when the installed 64-bit version is out of date.

## B.2 Prerequisite RPMs

See Section 5 for an overview of the prerequisite RPMs that will be built and Appendices starting with A.1 for how the source code for each is modified.

To build all prerequisite RPMs:

1. Change to the `releng` subdirectory
2. `export CROSS_FAIL_OK=1`
3. `make prereq`

If applicable, the build system will build an RPM for each target platform for each prerequisite. After the builds are complete, the RPMs will be copied into the `prereq` directory and listed. *If you are missing a specific cross-compiler, that platform will be skipped unless explicitly requested* (because of the `CROSS_FAIL_OK`).

There are other prerequisite-related `make` targets available (cf. Table 5); running `make help` in `releng` will present additional information, *e.g.* `make prereq-host` will only build for the host machine.

## B.3 Building Main RPMs

*Note*: This section assumes the prerequisite RPMs have been installed. If they were built on another machine, then other required RPMs must be installed as well; they will be requested by the build system[5].

The RPM-building `Makefile` has various targets which are shown by running `make` from within the `releng` subdirectory. A sampling:

Table 5: `releng` directory `make` targets

| Target | Result |
|---|---|
| clean | Remove generated files |
| prereq | Builds the prerequisite RPMs (described above) |
| cdk | The CDK RPMs (base libraries and runtime) |
| cdk-all | The CDK RPMs (all platforms) |
| driver | The driver RPM |

If you are making modifications to the framework, to rebuild for testing the command `make cdk` is usually sufficient.

### B.3.1 Common Problems

If you are having trouble building the RPMs, please ensure:

- All of the prerequisite packages listed above are installed.

  Remove any manually-installed versions if applicable.

  `sudo yum install releng/prereq/*rpm`

- The cross-compilers can be found.

  For example, if you installed the cross-compilers today, "`sudo updatedb`" will refresh the `locate` command's database.

- Attempt your build independent of RPM building (see Appendix C).

  *Note*: This can interfere with RPM builds and you **must** clean up the non-RPM build files with `make distclean` or `git clean -fdx`.

[5]The ones without 1:1 direct RPM names are covered in section B.1

### B.3.2 Debugging Your Build Process

If you are having trouble compiling the source code there are a few options to try:

- Examine the automatically generated log file, `build.log` (or `build-`*`platform`*`.log`).
- Disable multithreaded building by removing `%{?_smp_mflags}` from the specfile.

  This will help you trace the error message by processing source files serially, keeping error messages in `build.log` local to each other.

# C Appendix - Building OpenCPI from Source

When installing OpenCPI from RPM, there is *no need* to build the OpenCPI Framework from source and the ANGRYVIPER Team *does not* recommend this process for standard users. However, there are certain conditions where it may be appropriate, such as:

1. Contributions to the OpenCPI architecture
2. Utilization of the latest features between releases
3. Identification and troubleshooting of bugs

This section only applies to the ANGRYVIPER Team's branch(es) of OpenCPI Consult the standard *Installation Guide* for more information as well as commands to be used elsewhere.

## C.1 Prerequisites

Before building from source, ensure that the OpenCPI prerequisites are installed (cf. section 5). If you need to rebuild the RPMs for the prerequisites, see appendix B.2

To build from source, there are additional packages beyond the prerequisites that need to be installed. The primary ones are: **gcc**, **gcc-c++**, **automake**, **autoconf**, and **libtool**. The most efficient method to install these is through the **yum group "Development Tools"**[6].

```
% sudo yum groupinstall development
```

## C.2 Git Access

Open a terminal window, change directories to the location the framework should be checkout in:

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

This will create an `opencpi` subdirectory and populate it with the current master branch of the OpenCPI code base. Change into this directory before issuing further commands:

```
$ cd opencpi
```

By default, the `git clone` operation downloads the `master` branch of the git repository. The master branch points to the latest stable release of the framework. This stable release will correlate to a git tag version.

The OpenCPI releases are identified by version numbers, as explained in section 6 OpenCPI RPM naming convention with major.minor.subminor releases. The release compatibility policy is to maintain component binary compatibility within the sub-minor releases, and API source compatibility (requiring rebuilding) for minor releases.
To set the release of the codebase downloaded, use the `git checkout` command with the release tag as an argument:

```
$ git checkout release_1.3_rc
```

[6]`yum grouplist --verbose` shows the `development` alias.

## C.3 Building OpenCPI From Source

Set the `OCPI_CDK_DIR` variable, from the top level of the repository:

```
$ export OCPI_CDK_DIR=$(pwd)/exports
```

The provided `ocpi-configure` script wraps the "usual" GNU autotools commands like `reconf` and `configure`. From the top level of the repository, run the following:

```
$ ./ocpi-configure
$ make -j
```

To cross-build for a target RCC platform, instead of calling `ocpi-configure`, you call `cross-configure`. To use this script, you must configure the variable `OCPI_TARGET_PLATFORM`. See the standard documentation for more details. An example:

```
$ export OCPI_TARGET_PLATFORM=xilinx13_3
$ ./cross-configure
$ make -j
```

**Note**: To use the recently-installed CDK, you will need to add `$(pwd)/exports/bin/linux-c7-x86_64/` (or appropriate host platform) to your `PATH` variable.