OpenCPI
Tutorial 10: Introduction to OpenCPI Application Development

OpenCPI Release: v2.4.6

Revision History

Revision	Description of Change	Date
1.0	Creation from Lab 1 developed by CNF	2019-10-01
1.1	Updated for Digital Radio Controller (DRC)	2021-12-01
1.2	Updates to GUI and CLI to reflect ocpidev changes	2022-08-31
1.3	Update language regarding CentOS7 to make it clear that there are other available platforms	2023-01-20
1.4		
1.5		
1.6		
1.7		
1.8		
1.9		
2.0		
2.1		

Table of Contents

1 Overview....4
- 1.1 Tutorial Objectives....6
- 1.2 What's Provided for This Tutorial....7
- 1.3 Prerequisites to Using This Tutorial....8
- 1.4 Documentation References....9

2 Create Project....10

3 Create OAS....11

4 Copy Reference FSK DRC Data Files....12

5 Edit OAS....13

6 Create and Build fsk_modem HDL Assembly....19
- 6.1 Create fsk_modem HDL Assembly Description....20
- 6.2 Update HDL Assembly Description....21
- 6.3 Copy Container XML and Constraints....25
- 6.4 Build HDL Assembly....26

7 Run FSK Application....27

8 Troubleshooting....29

9 Tutorial Summary....30

1 Overview

This tutorial focuses on OpenCPI application development and introduces you to more complex application scenarios and goals. Our example is the OpenCPI FSK DRC **`fsk_dig_radio_ctrlr`** reference application, which is a software-defined radio (SDR) demo application that uses the Frequency-Shift Keying (FSK) protocol to perform FSK modulation and demodulation and the OpenCPI Digital Radio Controller (DRC) utility to configure and use the radio frequency (RF) I/O hardware. We will re-create this reference application and run it on the ADALM-PLUTO (**`plutosdr`**) hardware platform.

The application's function is to transmit to and receive a JPEG image from a radio device using the FSK protocol and the DRC utility. The OpenCPI components **`mfsk_mapper`**, **`zero_pad`**, **`fir_real_sse`**, **`phase_to_amp_cordic`**, **`rp_cordic`**, **`baudTracking`**, and **`real_digitizer`** from the built-in **`ocpi.asset`** project's component libraries implement the FSK protocol, while the **`file_read`** and **`file_write`** utilities from the **`ocpi.core`** project handle the file I/O. The **`drc`** component from the **`ocpi.platform`** project performs the DRC function.

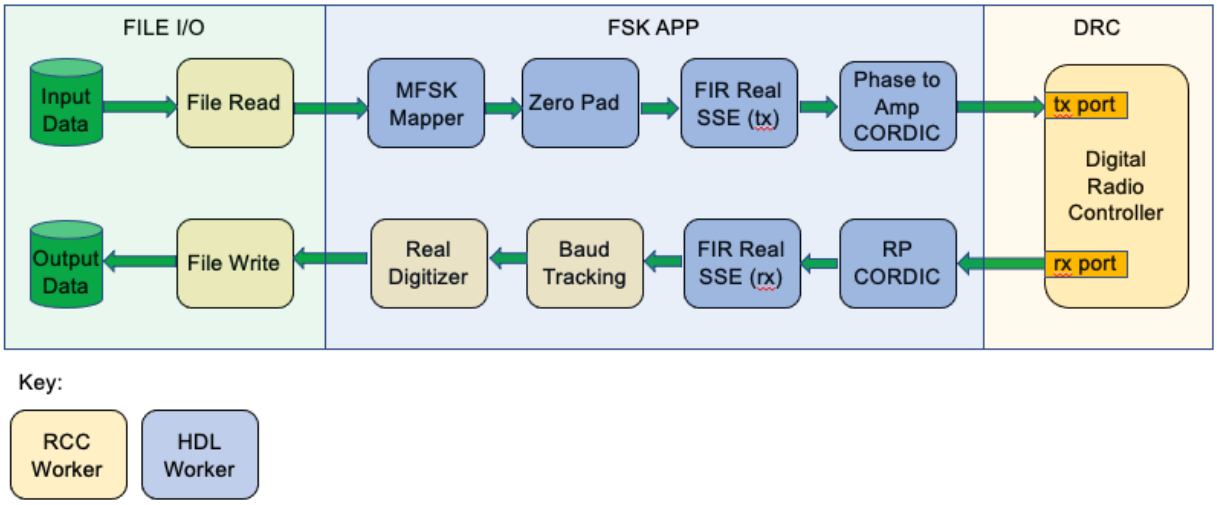


Figure 1: OpenCPI FSK DRC Reference Application

As shown in the figure:

- OpenCPI RCC and HDL application workers perform the FSK modulation and demodulation.
- OpenCPI RCC application workers handle the file I/O.
- A DRC RCC device proxy worker controls the RF I/O ports. See the section "Controlling Slave Workers from Proxies" in the ***OpenCPI RCC Development Guide*** for details on device proxy workers and how they function.

For details on the components used in the FSK DRC reference application, consult the component data sheets. Detailed information about the DRC utility can be found in the ***OpenCPI Application Development Guide*** in the section “Utility Components for Applications” and also in Briefing 14.

1.1 Tutorial Objectives

This tutorial demonstrates how to create an SDR application that uses FSK modulation and demodulation for data transmit/receive and the DRC utility for radio hardware configuration and control. It also demonstrates how to design an HDL assembly for execution on a target hardware platform.

The tutorial shows you how to:

- Create the SDR application using components that exist in OpenCPI built-in projects and input data and target platform-specific XML files from the OpenCPI **`fsk_dig_radio_ctrlr`** reference application.
- Create and build an HDL assembly to run on the ADALM-PLUTO radio device (the **`plutosdr`** HDL platform in OpenCPI).
- Deploy the application.

After running this tutorial, you should understand the general process of developing an SDR application from existing components and how to use the DRC utility to configure and control a radio.

1.2 What's Provided for This Tutorial

This tutorial directs you to use OpenCPI assets and scripts from the built-in OpenCPI projects **`ocpi.core`**, **`ocpi.assets`**, and **`ocpi.platform`**. You will also use assets and data files from the PLUTO SDR OSP **`ocpi.osp.plutosdr`**.

Instructions for copying required XML files and data files from these projects into the "demo" project that you create as part of the tutorial exercises are provided in the relevant sections of this document. The XML source for these items is also available as text in the relevant sections of this document that you can copy and paste into the relevant files following the instructions given in the section.

Note: when copying and pasting text, be sure to remove any line breaks in copied code and commands.

Note that all of the OpenCPI built-in projects described in the ***OpenCPI User Guide*** including the projects referenced here are built for the CentOS7 (**`centos7`**) and ADALM-PLUTO (**`plutosdr`**) platforms as part of the OpenCPI installation process. You do not need to build, register or import these projects in order to use them in this tutorial. Development hosts other than CentOS7 can also be used. If you are using a development host other than CentOS7, replace **`centos7`** with your development host's name.

You will need to download and build the PLUTO SDR OSP **`ocpi.osp.plutosdr`** to use it in this tutorial. See the ***OpenCPI Installation Guide*** for instructions.

1.3 Prerequisites to Using This Tutorial

This tutorial (Tutorial 10) has the system prerequisites described in ***OpenCPI Tutorial 1: Component-based Development***. See the prerequisites section in Tutorial 1 for details.

In addition to these requirements, this tutorial uses the **`plutosdr`** platform, and so the corresponding OSP should be downloaded and installed. See the section "Installation Steps for Platforms" in the ***OpenCPI Installation Guide***.

Before you begin this tutorial, you should have successfully completed Tutorial 1 through Tutorial 9.

We recommend reading ***Briefing 14 An Introduction to the Digital Radio Controller (DRC) model and APIs, for Controlling and Configuring RF I/O in OpenCPI Systems that are "radios" (SDRs)*** before getting started with this tutorial.

For a guide on using the **`ocpiremote`** tool, see the video: OpenCPI - Framework Installation – Zedboard. It is timestamped and begins when **`ocpiremote`** is being set up and executed. **Note**: It will be necessary to disable the firewall for **`ocpiremote`** to function properly. For more information, see the section "Using Remote Containers from Client/Development Systems" in the ***OpenCPI Application Development Guide*** and the **`ocpiremote(1)`** man page.

1.4 Documentation References

The documents listed in the following table provide detailed information about subjects discussed in this tutorial. The documents listed here are not required reading for this tutorial but will likely be of interest for more in-depth learning.

Table 1 - OpenCPI Reference Documentation

Title	Published By	Public URL
OpenCPI Overview	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/Overview.pdf
OpenCPI User Guide	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/OpenCPI_User.pdf
OpenCPI Application Development Guide	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/OpenCPI_Application_Development.pdf
OpenCPI Component Development Guide	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/OpenCPI_Component_Development.pdf
OpenCPI HDL Development Guide	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/OpenCPI_HDL_Development.pdf
OpenCPI RCC Development Guide	OpenCPI	https://opencpi.gitlab.io/releases/v2.4.6/docs/OpenCPI_RCC_Development.pdf

2 Create Project

First, we need to create an OpenCPI project in which to develop our FSK DRC demo application. We'll call it **DemoProject10**.

To create a new project from the command line, use the following **ocpidev** command:

```
ocpidev  -D ocpi.platform -D ocpi.tutorial -D ocpi.osp.plutosdr -D
ocpi.assets create project DemoProject10 --register
```

In the **Project.xml** file, add the following line after the **ProjectDependencies** attribute:

```
ComponentLibraries='components comms_comps dsp_comps misc_comps
util_comps'
```

3 Create OAS

We need to create an OAS (aka “app XML”) for our **`fsk_drc_demo`** application. We’ll use the same technique to create it that we used in Tutorial 2: we’ll create an empty OAS in the **`applications/`** directory and then edit this XML file to add our definitions.

We’ll call our application **`fsk_drc_demo`**.

To create the OAS with **`ocpidev`**, run the following command from the **`DemoProject10/`** directory:

```
ocpidev -X create application fsk_drc_demo
```

The **`-X`** option to **`ocpidev`** directs the tool to create the empty OAS in the applications directory.

4 Copy Reference FSK DRC Data Files

Our **fsk_drc_demo.xml** application needs the following data files in order to run:

- **opencpi.jpeg** – the input data file that will be transmitted through the Tx port and ultimately received on the Rx port.
- **rx_rrcos_taps.dat** – Filter coefficients that define the response of the receive FIR Filter component named **rx_fir_real**.
- **tx_rrcos_taps.dat** – Filter coefficients that define the response of the transmit FIR Filter component named **tx_fir_real**.

These data files are part of the **fsk_drc_dig_radio_crtlr** reference application in the PLUTO SDR OSP you downloaded and installed as a prerequisite to this tutorial, in the location:

```
<path-to-plutosdr-osp>/applications/fsk_drc_dig_radio_crtlr/
```

We'll simply copy these files into our **DemoProject10**: run the following command from the **DemoProject10/applications/** directory:

```
cp -rf <path-to-plutosdr-osp>/applications/fsk_dig_radio_ctrlr/
{opencpi.jpeg,rx_rrcos_taps.dat,tx_rrcos_taps.dat} .
```

5 Edit OAS

Now we'll edit our **fsk_drc_demo.xml** OAS with the necessary XML definitions.

First, we need to add the following component instances:

- One instance of **file_read**. The component spec file **file_read_spec.xml** resides in the **ocpi.core** project in the **specs/** directory of the **components** library.
- One instance of the **mfsk_mapper** component. The component spec file **mfsk_mapper-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/comms_comps** library.
- One instance of the **zero_pad** component. The component spec file **zero_pad-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/util_comps** library.
- Two instances of the **fir_real_sse** component: one to filter on the transmit side which we'll name **tx_fir_real**, and one to filter on the receive side named **rx_fir_real**. The component spec file **fir_real_sse-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/dsp_comps** library.
- One instance of the **phase_to_amp_cordic** component. The component spec **phase_to_amp_cordic-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/dsp_comps** library.
- One instance of the **drc** component. The component spec **drc-spec.xml** resides in the **ocpi.platform** project in the **specs/** project.
- One instance of the **rp_cordic** component. The component spec **rp_cordic-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/dsp_comps** library.
- One instance of the **Baudtracking_simple** component. The component spec **baudTracking-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/dsp_comps** library.
- One instance of the **real_digitizer** component. The component spec **real_digitizer-spec.xml** resides in the **ocpi.assets** project in the **specs/** directory of the **components/dsp_comps** library.
- One instance of the **file_write** component, to receive the output data from the **real_digitizer**. The component spec **file_write_spec.xml** resides in the **ocpi.core** project in the **specs/** directory of the components library.

Next, we need to specify the component properties:

- For the **file_read** instance, we need to set values for four properties: the **fileName** property, which we'll set to our input file **opencpi.jpeg,** the **messagesInFile** property, which we'll set to **false,** the **opcode** property set to **1** and the **granularity** property also set to a value of **1** (the number of samples that will constitute a message).
- For the **mfsk_mapper** instance, we need to set the values of one property: the property **symbols** that we'll set to the value **-32768,32767.** The **symbols** property controls the values that can appear on the output.
- For the **zero_pad** instance, we need to set the value of the **num_zeros** property to **38**, which sets the number of zeroes between the output samples.
- For the **tx_fir_real** instance, we need to set the value of the **taps** property to **tx_rrcos_taps.dat**, which is our already generated file.
- For the **phase_to_amp_cordic** instance, we need to set values for four properties: **magnitude** should be set to **20000**, **DATA_WIDTH** should be **16** (our input and output sizes) **DATA_EXT** to **6** (the number of extension bits) and **stages** set to **5** (the number of CORDIC stages implemented).
- For the **drc** instance, we need to specify a **configurations** property that defines a radio configuration with two channels: one for receive (**rx=true**) and one for transmit (**rx=false**). We also need to specify the **start** property so that this radio configuration is activated when the application starts. For more information on DRC component concepts and properties, see the section "DRC Properties" in the ***OpenCPI Application Development Guide***.
- For the **rp_cordic** instance, we need to set three values which are similar to the **phase_to_amp_cordic** instance: **DATA_WIDTH** should be **16**, **DATA_EXT** should be set to **6** and **stages** set to **5**.
- For the **rx_fir_real** instance, we'll set the **Model** to **HDL** and the **taps** property to **rx_rrcos_taps.dat**, which is our already generated file.
- For the **baudTracking** instance, we need to set three properties: **bypass** should be set to **false** (**true** passes the input to the output without processing), the **SPB** (samples per baud) property should be **39**, and the **BaudAvrCount** should be set to **10** (this is our number of bauds used to determine drift).
- For the **real_digitizer** instance, we need to set the value of the **need_sync** property to **true** (to find the baud synchronizing pattern; in this case, **0xFACE**), and the **enable_printing** property to **true** (when the pattern is found, print the data).

- For the **`file_write`** instance, we need to set the **`fileName`** property to the data output file **`fsk_drc_demo_out.bin`** and **`messagesInFile`** to **`false`**.

We also need to specify 11 connections. We'll use the shorthand **`Connect=`** in the instance element for all of the connections except for the ports we need to rename. For these, we'll use the full **`Port Instance=`** definition in the **`Connection`** element.

Unlike what we did for Tutorial 1 and 2, we won't specify a **`done`** attribute for this application. Instead, when the command is run, we'll specify a duration for which the application should run.

To update the empty OAS, open **`DemoProject10/applications/fsk_drc_demo.xml`** with a text editor.

Now replace the contents with the following XML:

```
<Application>
  <!--
=============================================================================
=====-→
  <!-- FSK modulation/TX algorithm components sending data to the
DRC tx port -->

  <Instance Component="ocpi.core.file_read" Connect="mfsk_mapper">
    <Property Name="fileName"       Value="opencpi.jpeg"/>
    <Property Name="messagesInFile" Value="false"/>
    <Property Name="opcode"         Value="1"/>
    <Property Name="granularity"    Value="1"/>
  </Instance>
  <Instance Component="ocpi.assets.comms_comps.mfsk_mapper"
Connect="zero_pad">
    <Property Name="symbols" Value="-32768,32767"/>
  </Instance>
  <Instance Component="ocpi.assets.util_comps.zero_pad"
Connect="tx_fir_real">
    <Property Name="num_zeros" Value="38"/>
  </Instance>
  <Instance Component="ocpi.assets.dsp_comps.fir_real_sse"
Name="tx_fir_real"

Connect="phase_to_amp_cordic">
    <Property Name="taps" ValueFile="tx_rrcos_taps.dat"/>
  </Instance>
  <Instance Component="ocpi.assets.dsp_comps.phase_to_amp_cordic"
Connect="drc" from='out' to='tx'>
    <Property Name="magnitude"  Value="20000"/>
    <Property Name="DATA_WIDTH" Value="16"/>
    <Property Name="DATA_EXT"   Value="6"/>
    <Property Name="STAGES"     Value="5"/>
  </Instance>

  <!--
=============================================================================
=======-->
```

```
 <!-- FSK demodulation/RX algorithm components with input from the
DRC RX port -->

  <Connection>
    <Port Instance="drc" Name="rx"/>
    <Port Instance="rp_cordic" Name="in"/>
  </Connection>
  <Instance Component="ocpi.assets.dsp_comps.rp_cordic"
Model="hdl"

Connect="rx_fir_real">
    <Property Name="DATA_WIDTH" Value="16"/>
    <Property Name="DATA_EXT"   Value="6"/>
    <Property Name="STAGES"     Value="5"/>
  </Instance>
  <Instance Component="ocpi.assets.dsp_comps.fir_real_sse"
Name="rx_fir_real"
Model="hdl"
   Connect="baudTracking">
    <Property Name="taps" ValueFile="rx_rrcos_taps.dat"/>
  </Instance>
  <Instance Component="ocpi.assets.dsp_comps.baudTracking"
Connect="real_digitizer">
    <Property Name="bypass"       Value="false"/>
    <Property Name="SPB"          Value="39"/> <!-- samples/baud
-->
    <Property Name="BaudAvrCount" Value="10"/>
  </Instance>
  <Instance Component="ocpi.assets.dsp_comps.real_digitizer"
Connect="file_write">
    <!-- look for 0xFACE sync pattern -->
    <Property Name="need_sync" Value="true"/>

    <!-- prints out when 0xFACE sync pattern is found →
    <Property Name="enable_printing" Value="true"/>
  </Instance>
  <Instance Component="ocpi.core.file_write">
    <!-- ACI relies on this property being set here in the OAS -->
    <Property Name="fileName"       Value="fsk_drc_demo_out.bin"/>

    <Property Name="messagesInFile" Value="false"/>
  </Instance>

  <!--
==========================================================================
========-->
```

```
 <!-- The DRC component, specifying channel configurations -->

  <Instance Component="ocpi.platform.drc">
    <property name='configurations'
              Value="{description first,
                      channels {{rx true,
                                 tuning_freq_MHz 2450,
                                 bandwidth_3dB_MHz 0.24,
                                 sampling_rate_Msps 0.25,
                                 samples_are_complex true,
                                 gain_mode auto,
                                 tolerance_tuning_freq_MHz 0.01,
                                 tolerance_sampling_rate_Msps 0.01,
                                 tolerance_gain_dB 1},
                                {rx false,
                                 tuning_freq_MHz 2450,
                                 bandwidth_3dB_MHz 0.24,
                                 sampling_rate_Msps 0.25,
                                 samples_are_complex true,
                                 gain_mode manual,gain_dB -30,
                                 tolerance_bandwidth_3dB_MHz 0,
                                 tolerance_tuning_freq_MHz 0.01,
                                 tolerance_sampling_rate_Msps 0.01,
                                 tolerance_gain_dB 1}}}"/>
    <property name='start' value='0'/>
  </Instance>

</Application>
```

6 Create and Build fsk_modem HDL Assembly

To run our application, we'll need an HDL assembly. Here we'll include all the HDL workers that we'll need for our application. For this tutorial, we will only be making the one assembly unlike in Tutorial 2.

6.1 Create fsk_modem HDL Assembly Description

We'll name this HDL assembly **`fsk_modem`**. Note the use of all lowercase characters in the assembly name. *Assembly names must not use uppercase characters* because some FPGA tools cannot handle them.

To create the HDL assembly with **`ocpidev`**, run the following command from the **`DemoProject10/`** directory:

```
ocpidev create hdl assembly fsk_modem
```

6.2 Update HDL Assembly Description

Now we need to update the HDL assembly description (OpenCPI HDL Assembly Description, or OHAD) to define the HDL workers to be used, the connections between them, and the connections to the rest of the application.

To update the HDL assembly OHAD from the command line, navigate to the **`fsk_modem`** directory and then open **`fsk_modem.xml`** with a text editor.

Now insert the following XML (highlighted in red) between the **`HdlAssembly`** tags:

```
<HdlAssembly>
  <!-- TX Chain Connections -->
  <Instance Worker="mfsk_mapper">
    <Property Name="M_p" Value="2"/>
  </Instance>
  <Instance Worker="zero_pad">
    <Property Name="DWIDTH_p" Value="16"/>
  </Instance>
  <Instance Worker="fir_real_sse_for_xilinx"
Name="tx_fir_real">
    <Property Name="DATA_WIDTH_p" Value="16"/>
    <Property Name="COEFF_WIDTH_p" Value="16"/>
    <Property Name="NUM_TAPS_p" Value="64"/>
  </Instance>
  <Instance Worker="phase_to_amp_cordic">
    <Property Name="DATA_WIDTH" Value="16"/>
    <Property Name="DATA_EXT" Value="6"/>
    <Property Name="STAGES" Value="5"/>
  </Instance>
  <Instance Worker="cic_int">
    <Property Name="N" Value="3"/>
    <Property Name="M" Value="1"/>
    <Property Name="R" Value="16"/>
    <Property Name="ACC_WIDTH" Value="28"/>
  </Instance>
  <Instance Worker="iqstream_to_cswm"/>

  <Connection Name="in_to_asm_tx_path" External="consumer">
    <Port Instance="mfsk_mapper" Name="in"/>
  </Connection>
  <Connection>
    <Port Instance="mfsk_mapper" Name="out"/>
    <Port Instance="zero_pad" Name="in"/>
  </Connection>
  <Connection>
    <Port Instance="zero_pad" Name="out"/>
    <Port Instance="tx_fir_real" Name="in"/>
  </Connection>
  <Connection>
    <Port Instance="tx_fir_real" Name="out"/>
    <Port Instance="phase_to_amp_cordic" Name="in"/>
  </Connection>
  <Connection>
    <Port Instance="phase_to_amp_cordic" Name="out"/>
    <Port Instance="cic_int" Name="in"/>
  </Connection>
  <Connection>
```

```
    <Port Instance="cic_int" Name="out"/>
    <Port Instance="iqstream_to_cswm" Name="in"/>
  </Connection>
  <Connection Name="out_from_asm_tx_path_to_dac"
External="producer">
    <Port Instance="iqstream_to_cswm" Name="out"/>
  </Connection>

<!-- RX Chain Connections -->
  <Instance Worker="cswm_to_iqstream"/>
  <!-- OpenCPI issue #1318 - AD9363 contains AGC,dc offset, iq
correction -->
  <Instance Worker="dc_offset_filter">
    <Property Name="PEAK_MONITOR_p" Value="true"/>
  </Instance>
  <Instance Worker="iq_imbalance_fixer">
    <Property Name="PEAK_MONITOR_p" Value="true"/>
  </Instance>

  <Instance Worker="complex_mixer">
    <Property Name="NCO_DATA_WIDTH_p" Value="12"/>
    <Property Name="INPUT_DATA_WIDTH_p" Value="12"/>
    <Property Name="PEAK_MONITOR_p" Value="true"/>
    <Property Name="CORDIC_STAGES_p" Value="12"/>
  </Instance>
  <Instance Worker="cic_dec">
    <Property Name="N" Value="3"/>
    <Property Name="M" Value="1"/>
    <Property Name="R" Value="16"/>
    <Property Name="ACC_WIDTH" Value="28"/>
  </Instance>
  <Instance Worker="rp_cordic">
    <Property Name="DATA_WIDTH" Value="16"/>
    <Property Name="DATA_EXT" Value="6"/>
    <Property Name="STAGES" Value="5"/>
  </Instance>
  <Instance Worker="fir_real_sse_for_xilinx"
Name="rx_fir_real">
    <Property Name="DATA_WIDTH_p" Value="16"/>
    <Property Name="COEFF_WIDTH_p" Value="16"/>
    <Property Name="NUM_TAPS_p" Value="64"/>
  </Instance>

  <Connection Name="in_to_asm_rx_path_from_adc"
External="consumer">
    <Port Instance="cswm_to_iqstream" Name="in"/>
  </Connection>
```

```
    <Connection>
      <Port Instance="cswm_to_iqstream" Name="out"/>
      <Port Instance="complex_mixer" Name="in"/>
    </Connection>
    <Connection>
      <Port Instance="complex_mixer" Name="out"/>
      <Port Instance="cic_dec" Name="in"/>
    </Connection>
    <Connection>
      <Port Instance="cic_dec" Name="out"/>
      <Port Instance="rp_cordic" Name="in"/>
    </Connection>
    <Connection>
      <Port Instance="rp_cordic" Name="out"/>
      <Port Instance="rx_fir_real" Name="in"/>
    </Connection>
    <Connection Name="out_from_asm_rx_path" External="producer">
      <Port Instance="rx_fir_real" Name="out"/>
    </Connection>
  </HdlAssembly>
```

Note: For reference, essentially the same OHAD exists in the **`ocpi.plutosdr`** project at **`hdl/assemblies/fsk_modem/fsk_modem.xml`**.

You'll notice here that some of these instances were not shown in our application OAS. This is because the DRC is able to infer that some of these instances will be necessary when the application is run and will automatically pull them in. Taking a look at the XML for the DRC, located at **`ocpi.osp.plutosdr/hdl/devices/drc_plutosdr.rcc`**, we can see which components the DRC will instantiate.

6.3 Copy Container XML and Constraints

Our HDL assembly needs an HDL container XML file to connect its multiple interconnects. You can read more about container XML files in the section “HDL Container XML Files” in the ***OpenCPI HDL Development Guide***. We also need to reference an FPGA constraints file for our target platform and use a Makefile that contains some special instructions.

All of these items exist as reference files in the PLUTO SDR OSP you downloaded and installed according to the pre-requisites for this tutorial. We'll simply copy these files from the PLUTO SDR OSP into our DemoProject10 so we can use them. Navigate to **`DemoProject10/hdl/assemblies/fsk_modem`** and then copy the files with this command:

```
cp -L <path-to-plutosdr-osp>/hdl/assemblies/fsk_modem/
{cnt.xml,plutosdr_data_src_qadc_ad9361_sub_data_sink_qdac_ad9361_
sub_mode_2_cmos.xdc,Makefile} .
```

6.4 Build HDL Assembly

Now we need to build the HDL assembly for the RCC and HDL platforms to create the FPGA artifact for the application.

To build the HDL assembly with **`ocpidev`**, run the following command from the **`DemoProject10/hdl/assemblies/fsk_modem/`** directory:

```
ocpidev build --hdl-platform plutosdr
```

The build takes a couple of minutes to run. You can confirm that it succeeded by locating the **`*.bitz`** file in the **`artifacts/`** subdirectory of **`DemoProject10`**. This file is the artifact file to be used when the application runs on the HDL platform.

7 Run FSK Application

First we'll check that the ADALM-PLUTO device is connected. Details of how to set up and configure the deployment platform are beyond the scope of this tutorial. See the prerequisites section in this tutorial for more information.

To get the status of the deployment platform:

- **`export OCPI_SERVER_ADDRESSES=<Pluto_IP>:12345`**
- **`ocpiremote status -p analog`**

There should be output that reads something similar to:

```
Executing remote configuration command: status
Server is running with port: 12345 and pid: 26224
```

Now we can run the **`fsk_drc_demo`** application on the **`plutosdr`** platform with the **`ocpidev run`** utility.

Navigate to the **`DemoProject10/applications/`** directory.

Execute **`ocpidev run`** to start the application:

```
ocpidev run application fsk_drc_demo -- -d -v -t 30
-wdrc=drc_plutosdr -P drc=centos7 -P file_write=centos7 -P
file_read=centos7
```

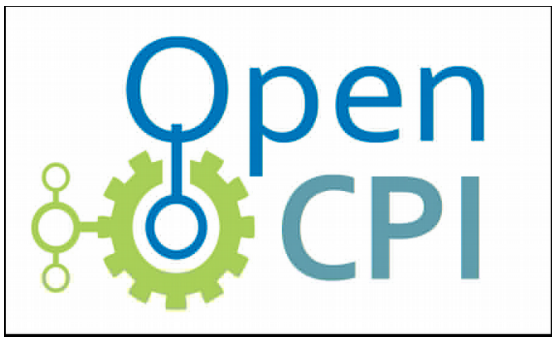
In this command, we are telling **`ocpidev run`** to:

- Write property values to **`stderr`** before starting the application and after it is done (**`-d`**).
- Be verbose (**`-v`**).
- Run the application for 30 seconds; if the application does not finish in this amount of time, the command will exit with an error (**`-t 30`**).
- Choose the instance of the **`drc`** RCC worker that runs on the **`plutosdr`** platform (**`-wdrc=drc_plutosdr`**).
- Run the **`drc`**, **`file_write`**, and **`file_read`** RCC workers on the **`centos7`** platform (**`-P`**).
- Run our **`fsk_drc_demo`** OAS XML.

Now run the following command from the same directory (**`DemoProject10/applications/`**):

```
eog fsk_drc_demo_out.bin
```

If you see the following image, the application ran successfully.


Open
CPI

8 Troubleshooting

To view a detailed log while running on your hardware platform, use the command:

```
export OCPI_LOG_LEVEL=8
```

Double-check your spelling for typos inside the component properties.

Problems relating to library entries not found:

- **`.gz`** library files were missing from the assets/exports.
- **`dsp_comps`** library HDL assemblies were cleaned and rebuilt to resolve missing **`.gz`** libraries.

9 Tutorial Summary

Now that this tutorial has been completed, you should be more familiar with more complex applications and HDL assemblies and how to include the DRC utility in an application to control a radio. As always, the briefings and guides listed and linked to in this tutorial provide more information and finer details on the subjects discussed in this tutorial.