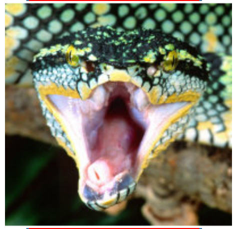
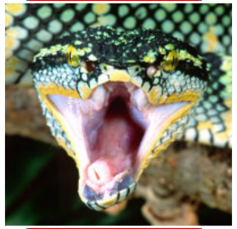


Lab 4: Complex Mixer

Integrating a 3rd party library into an RCC Worker

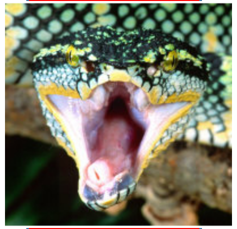
Objectives



- Learn how [RCC] workers can:
  - Import a 3rd party library (liquid dsp) functionality into a worker

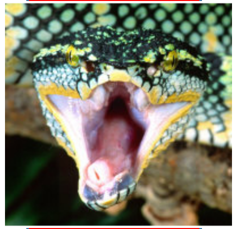
- Reiterate:
  - C++ conventions
    - Accessing port data and Properties
  - Framework interactions
    - RCC_ADVANCE vs. RCC_OK

Application Worker Development Flow



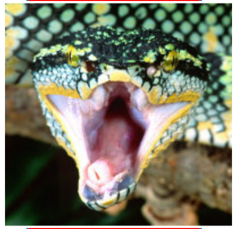
1. Protocol (OPS): Use pre-existing or create new
2. Component (OCS): Use pre-existing or create new
3. Create new App Worker (Modify OWD, Makefile, and source HDL/RCC code)
4. Build the App Worker for target device(s)
5. Create Unit Test ({component}-test.xml, generate, verify and view scripts)
6. Build Unit Test
7. Run Unit Test

Overview



- The "Complex Mixer" component receives I/Q data and multiplies this signal by a tone that is generated using a Numerically Controlled Oscillator (NCO).
- This causes the input signal to be shifted in the Frequency Domain by the frequency of the NCO that is generated in the worker.
- The frequency of the NCO is controlled by the properties of this worker.

Step 1 – OPS: Use pre-existing or create new



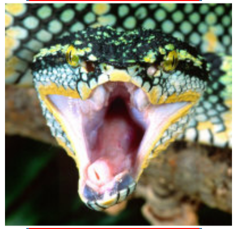
1) Identify the OPS(s) declared by this component
   - Examine the "Component Ports" table in the Component Datasheet
2) Determine if OPS(s) exists
   1) Current project's component library?

      /home/training/training_project/components/specs
   2) Other projects' components/specs/ directories within scope

      Intersection of Project-registry and ProjectDependencies= in {my_project}/Project.mk
3) If NO to all questions ⇨ Create new OPS

ANSWER: REUSE! OPS XML file is available from framework

Step 2 – OCS: Use pre-existing or create new



1) Review Component Spec Properties and Ports in Component Datasheet
   - Use Properties and Ports information to answer the following questions
2) Determine if an OCS exists that satisfies the requirements.
   1) Current project's component library?

      /home/training/training_project/components/specs/
   2) Other projects' components/specs/ directories within scope

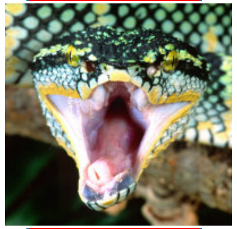
      Intersection of Project-registry and ProjectDependencies= in {my_project}/Project.mk
3) If NO to all questions ⇨ Create new OCS

ANSWER: Must create a new OCS XML file

Step 2 - Create Component

- Via IDE:
  - Create new Asset Type: Component
  - Component Name: complex_mixer
  - Add to Project: ocpi.training
- Or via command-line:
  $ ocpidev -d /home/training/training_project create spec complex_mixer -l components
- The component datasheet is located in
  - /home/training/provided/doc/Complex_Mixer.pdf
  - Review the component's datasheet and familiarize yourself with the properties and their functionality
- Modify the Spec in the IDE:OCS Editor
  - Edit the OCS based on the data sheet's "Component Spec Properties" and "Component Ports"
  - Hint: The iqstream_protocol.xml is located in Core Project
  - Note: Ignore “data_select” which is a HDL Application Worker only property

Step 2 - Create Component (cont.)

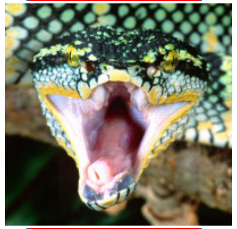


- Manually add the “Default” attribute and value to properties
  - Currently, IDE does not provide the “Dtrefault” field attribute for a property
  - Must be manually added by modifying the XML source

1) In the OCS Editor, which view from “Design” tab to the “Source” tab
2) Per the datasheet, add the “Default” attribute and value, to the appropriate property

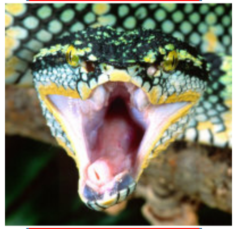
```xml
<ComponentSpec>
      <Property Name="enable" Type="bool" Writable="true" Default="true"></Property>
      <Property Name="phs_inc" Type="short" Writable="true" Default="-8192"></Property>
      <Port Name="in" Protocol="iqstream_protocol"></Port>
      <Port Name="out" Protocol="iqstream_protocol" Producer="true"></Port>
</ComponentSpec>
```

Step 3 - Create Worker



- Create new Asset Type: Worker
  - Worker Name: complex_mixer
  - Library: components
  - Component: complex_mixer-spec.xml
  - Model: RCC
  - Prog. Lang: C++

Step 3 – Create new App Worker (cont.)

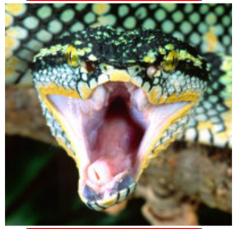


- In the RCC App Worker OWD Editor
  - Add “initialize” and “release” to the ControlOperations
  - Add the liquidDSP prerequisite library by entering “liquid” in the StaticPreReqLibs attribute
- Manually add version=2 into the xml source (can’t use IDE)
- No additional worker properties and ports are needed from the datasheet because they will be inherited from the component-spec.

```
<RccWorker language='c++' spec='complex_mixer-spec' controlOperations="initialize,release" StaticPreReqLibs="liquid" Version="2">

</RccWorker>
```

Step 3 - Write the Worker's Code

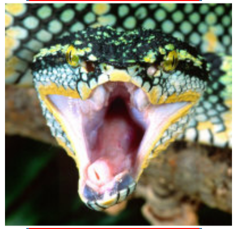


- Copy complex_mixer.cc
  - From: /home/training/provided/lab4/
  - To: /home/training/training_project/components/complex_mixer.rcc/

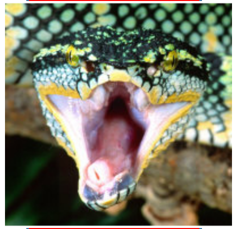
    $ cp /home/training/provided/lab4/complex_mixer.cc \
    /home/training/training_project/components/complex_mixer.rcc/

- Update any "???" in the source with the correct code

Liquid DSP NCO API (for reference)

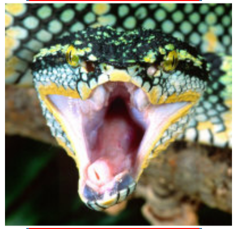


- From liquidsdr.org:
  - nco_crcf_create(type)
    - creates an nco object of type LIQUID_NCO or LIQUID_VCO
  - nco_crcf_destroy(q)
    - destroys an nco object, freeing all internally-allocated memory
  - nco_crcf_set_frequency(q,f)
    - sets the frequency f (equal to the phase step size $\Delta\theta$)
  - nco_crcf_set_phase(q,theta)
    - sets the internal nco phase to θ



Liquid DSP NCO API (for reference)

- From liquidsdr.org:
  - nco_crcf_step(q)
    - increments the internal nco phase by its internal frequency, $\theta \leftarrow \theta + \Delta\theta$
  - nco_crcf_mix_down(q,x,*y)
    - rotates an input sample x by $e-j\theta$, storing the result in the output sample y
  - All samples are of type liquid_float_complex
    - liquid_float_complex sample;
    - sample.I = 0;
    - sample.Q = 0;

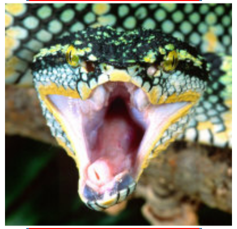


Step 4 - Building the App Worker for x86 and ARM

- Execute build for CentOS7-x86 and ARM
  1) Use the IDE to "Add" the App Worker to the Project Operations Panel
  2) Highlight "centos7" and “xilinx13_4” in RCC Platforms panel
  3) Check "Assets" Radio button
  4) Click "Build"
  5) Review the Console window messages
- Alternatively, build from Command-line:
  - Browse to the top-level of the project's directory and run
    - Similar operation ran by IDE

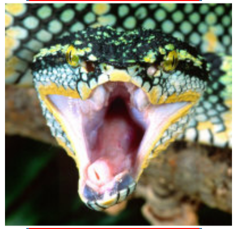
  $ ocpidev build worker complex_mixer.rcc --rcc-platform centos7

Step 5(a) – 7(a) CentOS7 - x86



- These slides cover employing the framework's Unit Test Suite to generate:
  - OAS (OpenCPI Application Specification) XML file(s)
    - Used by the framework for running the Worker on a given platform
  - Input test data file(s)

Step 5(a) - Create Unit Test



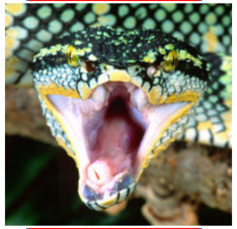
- Create a unit test for the "peak_detector" component, which results in generation of the "peak_detector.test/" directory
  1) File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard → Unit Test
  2) Add to Project: training_project
  3) Add to Library: components
  4) Component Spec: complex_mixer-spec.xml

- OR in a terminal window

  $ ocpidev create test complex_mixer

  – Note the Makefile and stub files complex_mixer-test.xml, generate.py, verify.py, view.sh

Step 5(a) - Create Unit Test



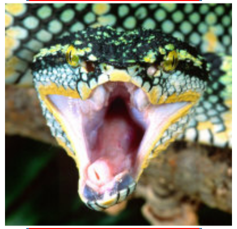
- Copy `generate.py`, `verify.py`, and `view.sh`

```
cp -a ~/provided/lab4/complex_mixer.test/*  ~/training_project/components/complex_mixer.test/
```

- Update complex_mixer-test.xml

```
<!-- This is the test xml for testing component "complex_mixer" -->
<Tests UseHDLFileIo='true'>
  <!-- Here are typical examples of generating for an input port and verifying results
       at an output port-->
  <Input Port='in' Script='generate.py 100 12.5 32767 16384'/>
  <Output Port='out' Script='verify.py 100 16384' View='view.sh'/>
  -->
  <!-- Set properties here.
       Use Test='true' to create a test-exclusive property. -->
  <Property Name='phs_inc' Values='-8192'/>
  <Property Name='enable' Values='0,1'/>
</Tests>
```

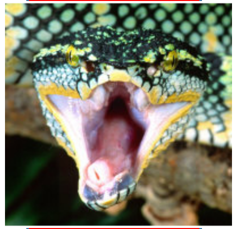
Step 6(a) - Build Unit Test (x86)



- Build the Unit Test Suite for the target software platform
  1) Use the IDE to "**Add**" the Unit Test to the Project Operations panel
  2) **Highlight** "centos7" in the RCC Platforms panel
  3) Select "Tests" Radio button
  4) Click "gen + build"
  5) Review the Console window messages and address any errors

- Observe new artifacts in complex_mixer.test/gen/
  – cases.txt – "Human-readable" file which lists various test configurations.
  – cases.xml – Used by framework to execute tests.
  – cases.xml.deps – List of dependent files
  – applications/ - OAS files and scripts used by framework to execute applications.

Step 7(a) - Run Unit Test (x86)



- Via IDE:
  1) Click "prep + run + verify" button to run the test

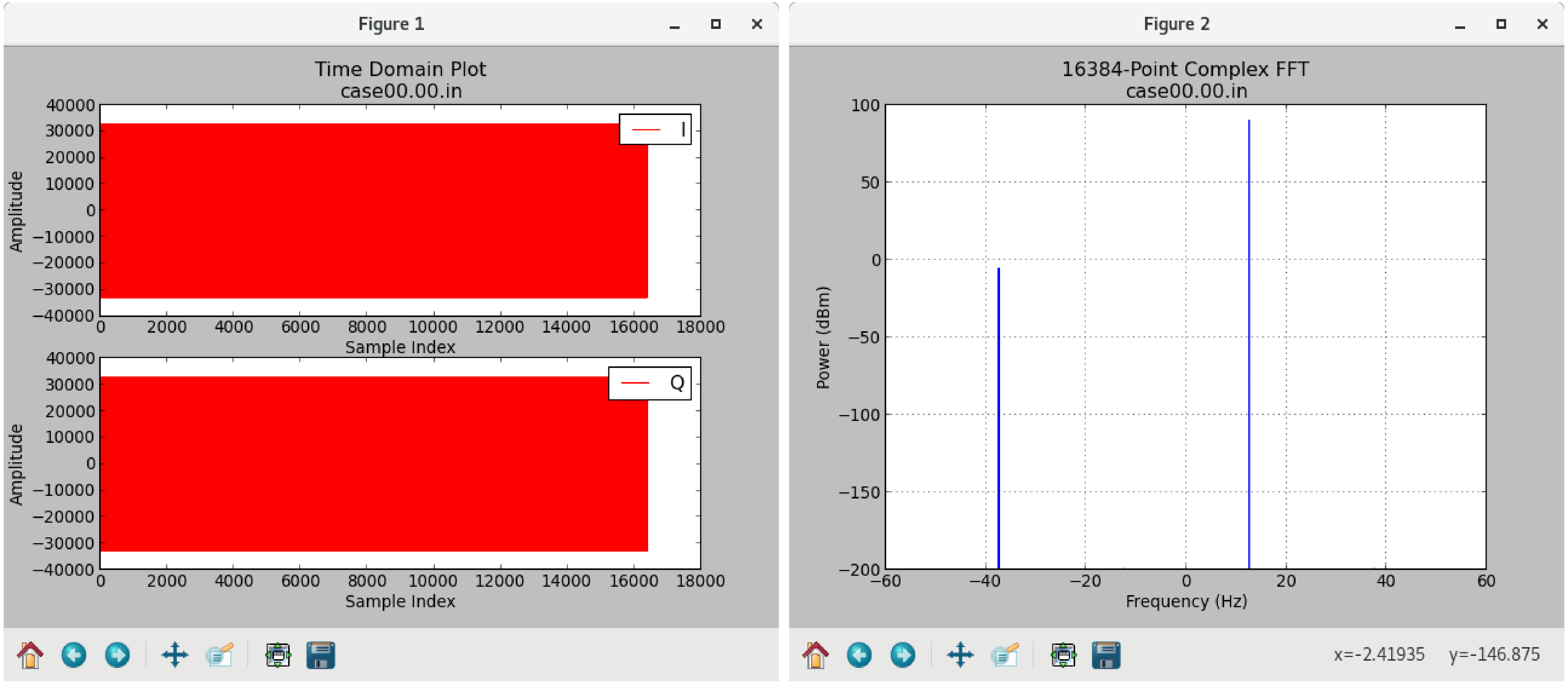
     The test should run quickly. Upon completion, you should see "PASSED" along with final values for the min/max peaks.

  2) Click the "view" button to view the test results

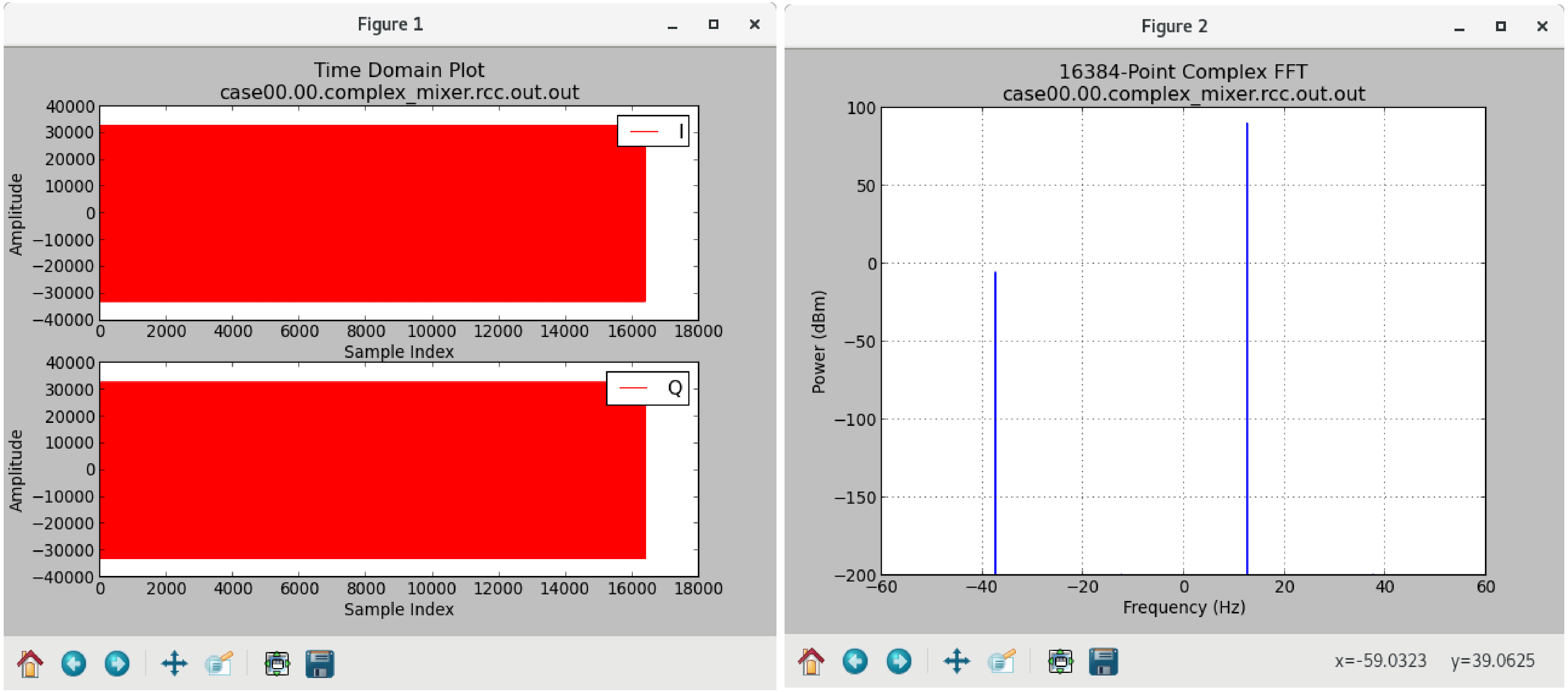
     Plots of input and output (time and frequency domain) will pop up.

- Via Command-line:
  1) In a terminal, browse to complex_mixer.test/ and execute
  2) $ ocpidev run --mode prep_run_verify (This uses the default centos7)
  - Also try:
    - $ ocpidev run --mode prep_run_verify --only-platform centos7 --view {limits platforms to test}
    - $ ocpidev run --mode prep_run_verify {run on all available platforms, no plotting}
    - $ ocpidev run --mode verify {verify previous results}
    - $ ocpidev run --mode view {plot previous results}

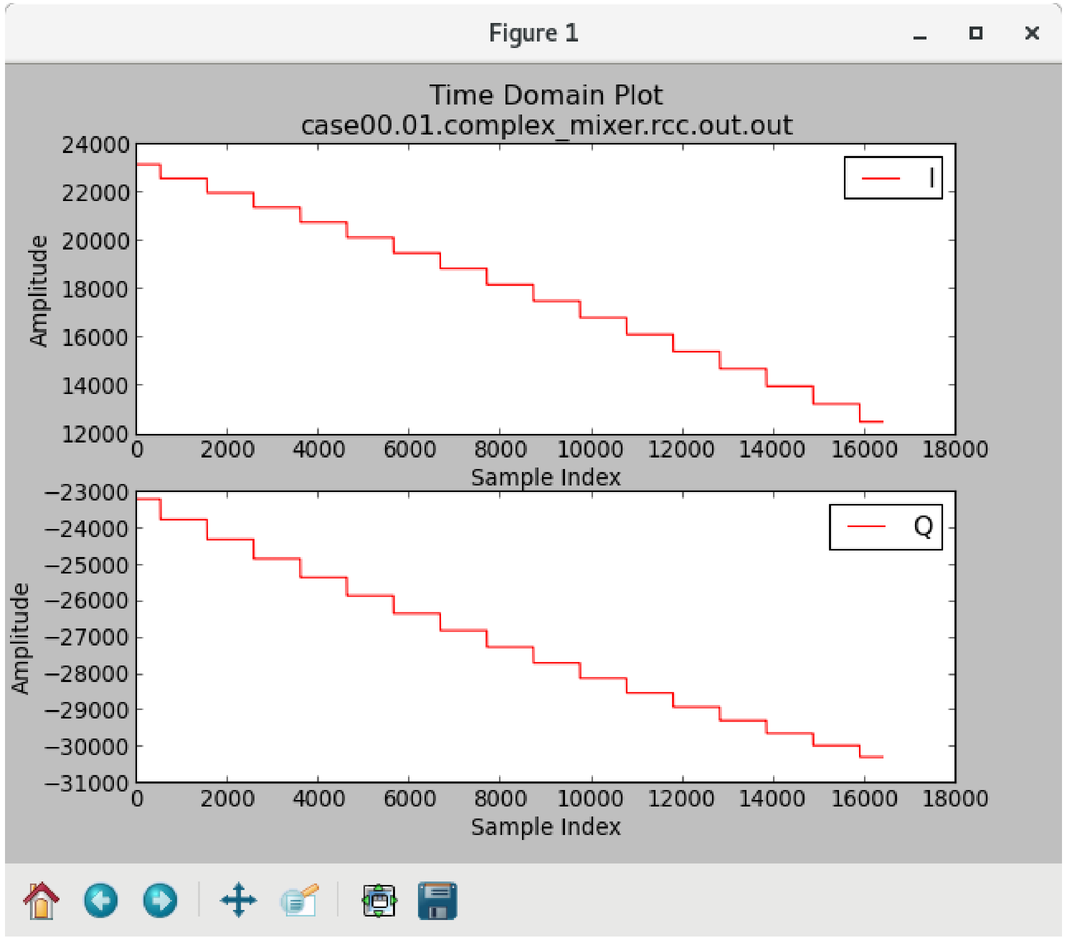
Expected input file plot for all cases*

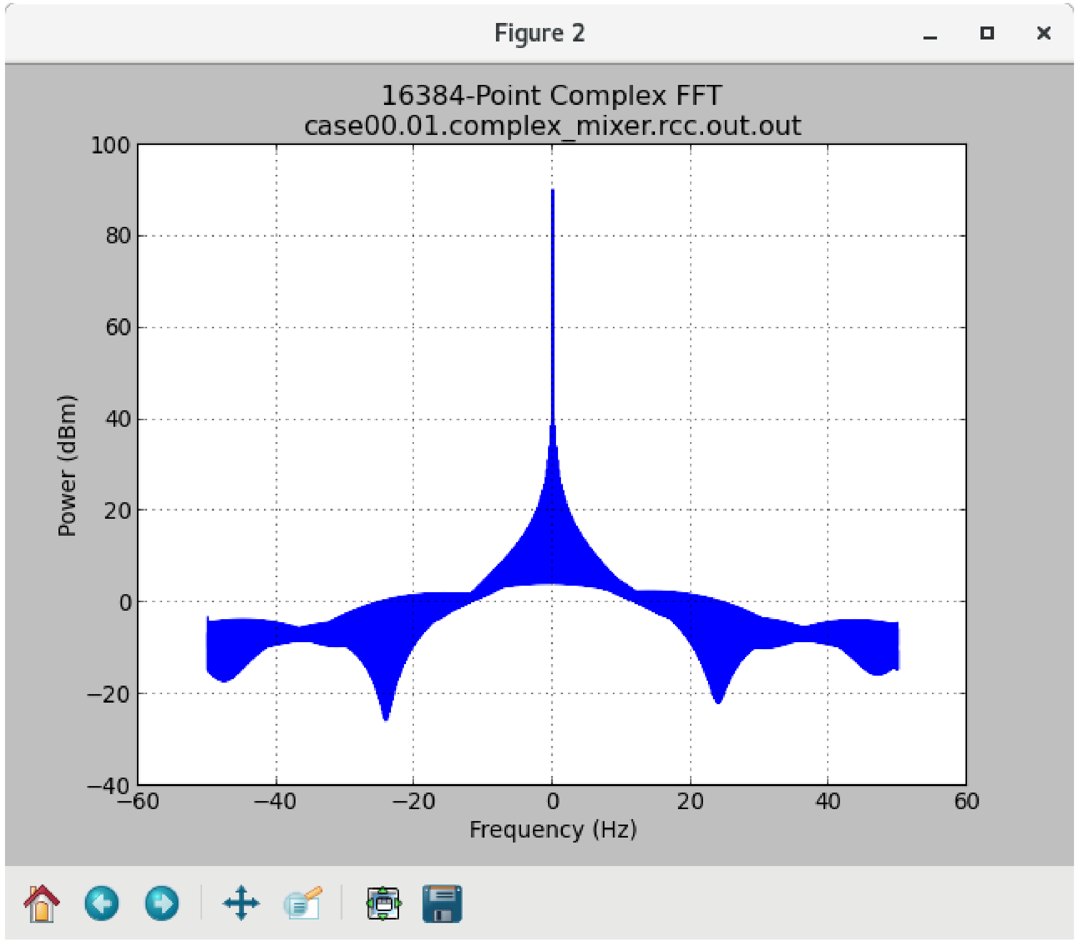


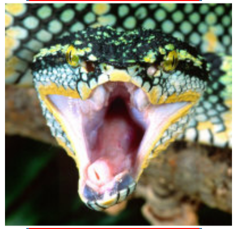
Expected output file plot case00.00 (bypass)



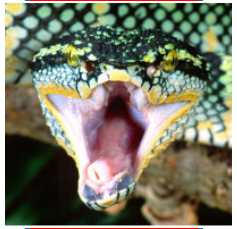
Expected output file plot case00.01 (enabled)





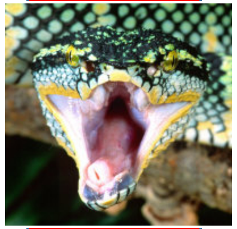


Step 5(b) – 7(b) xilinx13_4 - ARM



- These slides cover employing the framework's Unit Test Suite to generate:
  - OAS (OpenCPI Application Specification) XML file(s)
    - Used by the framework for running the Worker on a given platform
  - Input test data file(s)
  - Various scripts to manage the execution of the applications onto the target platform(s)

Step 5(b) - Create Unit Test

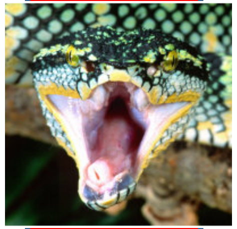


- Located in "complex_mixer.test/" directory
  - Same as used for CentOS7
    - **REUSE!**

- Reuse complex_mixer.test

```xml
<!-- This is the test xml for testing component "complex_mixer" -->
<Tests UseHDLFileIo='true'>
  <!-- Here are typical examples of generating for an input port and verifying results
       at an output port-->
  <Input Port='in' Script='generate.py 100 12.5 32767 16384'/>
  <Output Port='out' Script='verify.py 100 16384' View='view.sh'/>
  -->
  <!-- Set properties here.
       Use Test='true' to create a test-exclusive property. -->
  <Property Name='phs_inc' Values='-8192'/>
  <Property Name='enable' Values='0,1'/>
</Tests>
```

Step 6(b) – Build Unit Test (ARM)



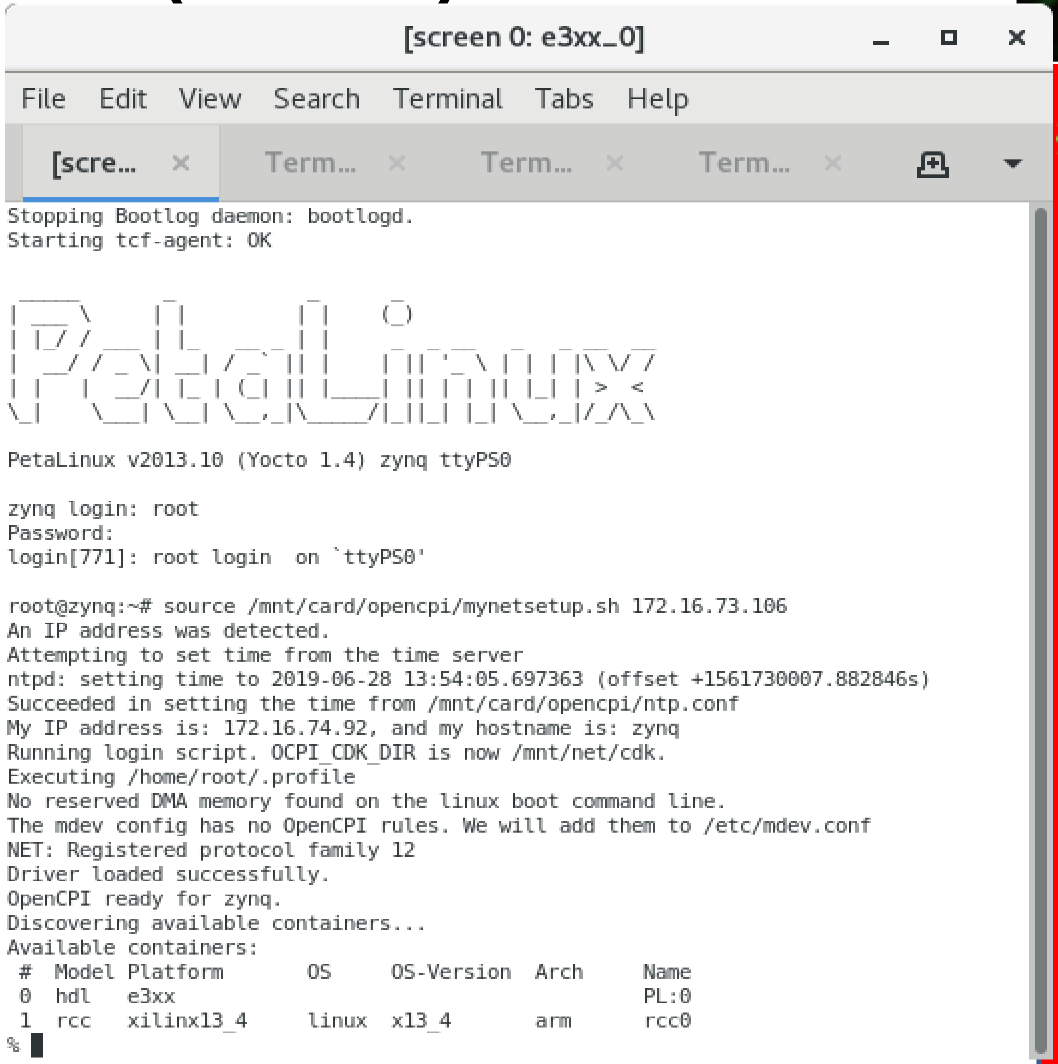
- Build the Unit Test Suite for the target software platform
  1) Use the IDE to "**Add**" the Unit Test to the Project Operations panel
  2) **Highlight** "xilinx13_4" in the RCC Platforms panel
  3) Select "Tests" Radio button
  4) Click "gen + build"
  5) Review the Console window messages and address any errors

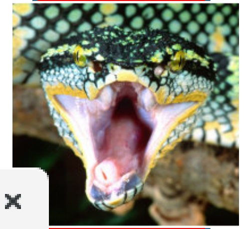
- Observe new artifacts in complex_mixer.test/gen/
  - cases.txt – "Human-readable" file which lists various test configurations.
  - cases.xml – Used by framework to execute tests.
  - cases.xml.deps – List of dependent files
  - applications/ - OAS files and scripts used by framework to execute applications.

Step 7(b) – Run Unit Test (ARM)

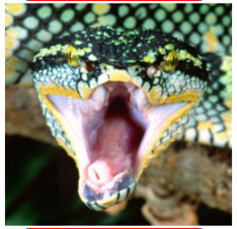
- Setup deployment platform
  1. Connect to serial port via USB on rear of Ettus E310 on Host
     - "screen /dev/e3xx_0 115200"
  2. Boot and login into Petalinux on E310
     - User/Password = root:root
  3. Verify Host and E310 have valid IP addresses
     - For training, they should both be on the same subnet
  4. Run setup script on E310
     - "source /mnt/card/opencpi/mynetsetup.sh <Host ip address>"

More detail on this process can be found in the **E3xx Getting Started Guide** document



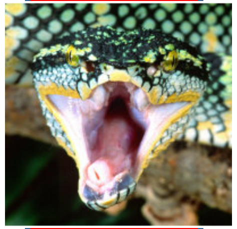


Step 7(b) - Run Unit Test (ARM) (cont.)



- AV IDE approach to running unit tests on remote platforms:
  1) In the “Project Operations” panel
  2) Select "remotes" radio button
  3) Click "+remotes"
  4) Change remote variable text to use Ettus E310’s IP and point to the training project:
  5) {IP of Ettus E310}=root=root=/mnt/training_project
  6) Select the newly created remote. This will be the target remote test system. Unselected remotes will not be targeted.
  7) Select “xilinx13_4” in the “RCC Platforms” panel
  8) Check "run view script" to view the output after verification.
  9) Click "prep + run + verify" to run the unit test scripts.

Step 7(b) – Run Unit Test (ARM) (cont.)

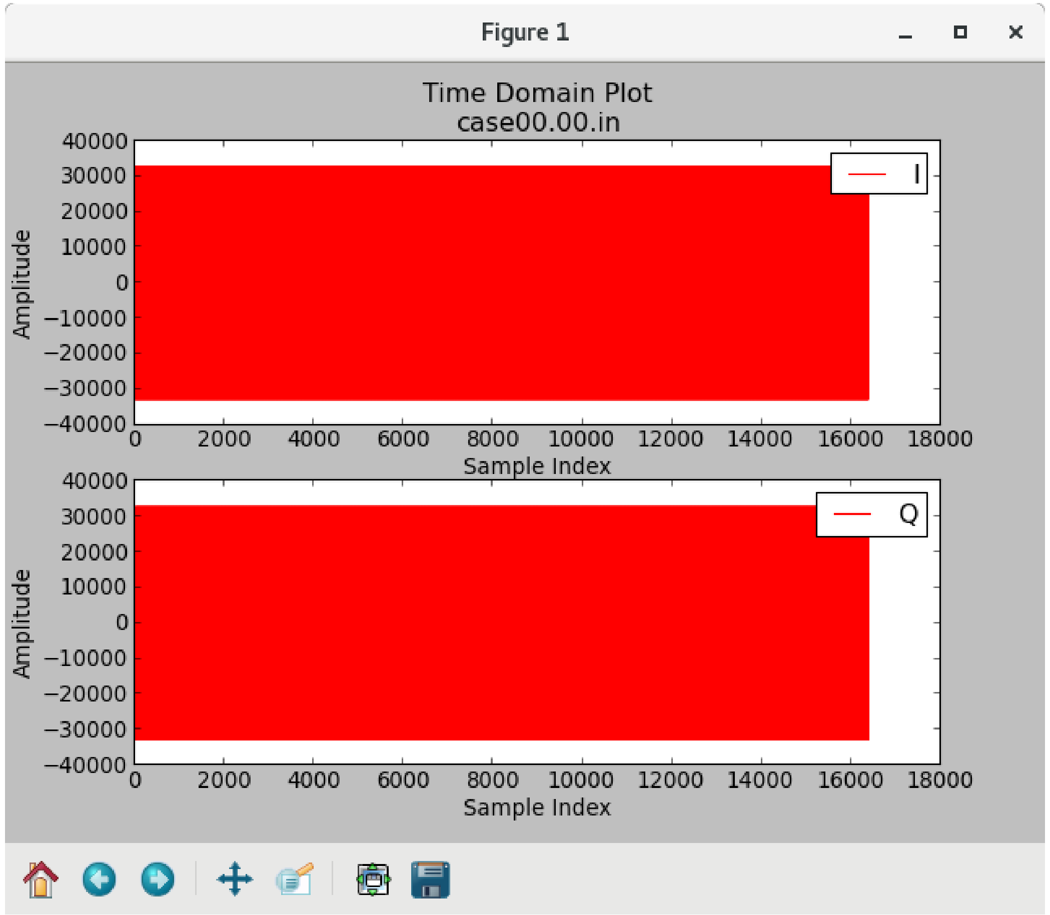


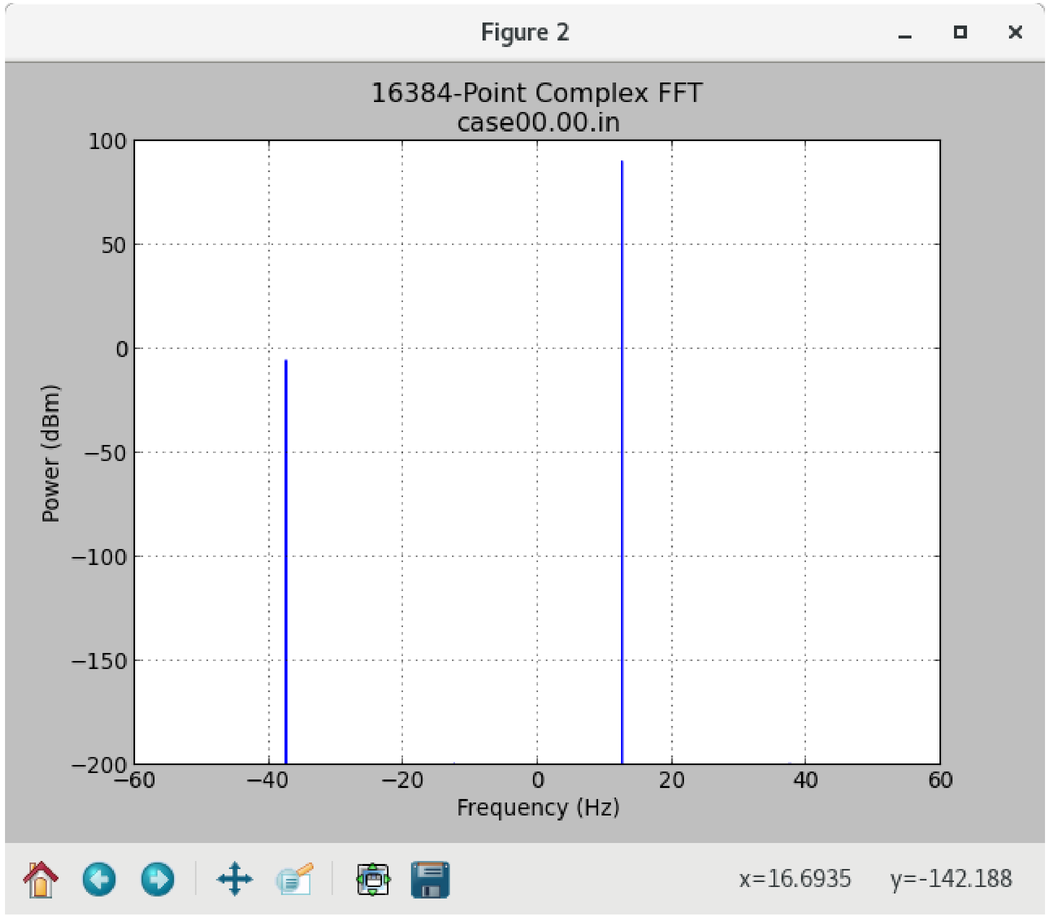
- Via a Command-line terminal (of the Development host) approach to running unit tests on remote platforms:
  1) Set OCPI_REMOTE_TEST_SYSTEMS, as shown:

     $ export OCPI_REMOTE_TEST_SYSTEMS={IP of Ettus E310}=root=root=/mnt/training_project
  2) Browse to complex_mixer.test/ and execute:

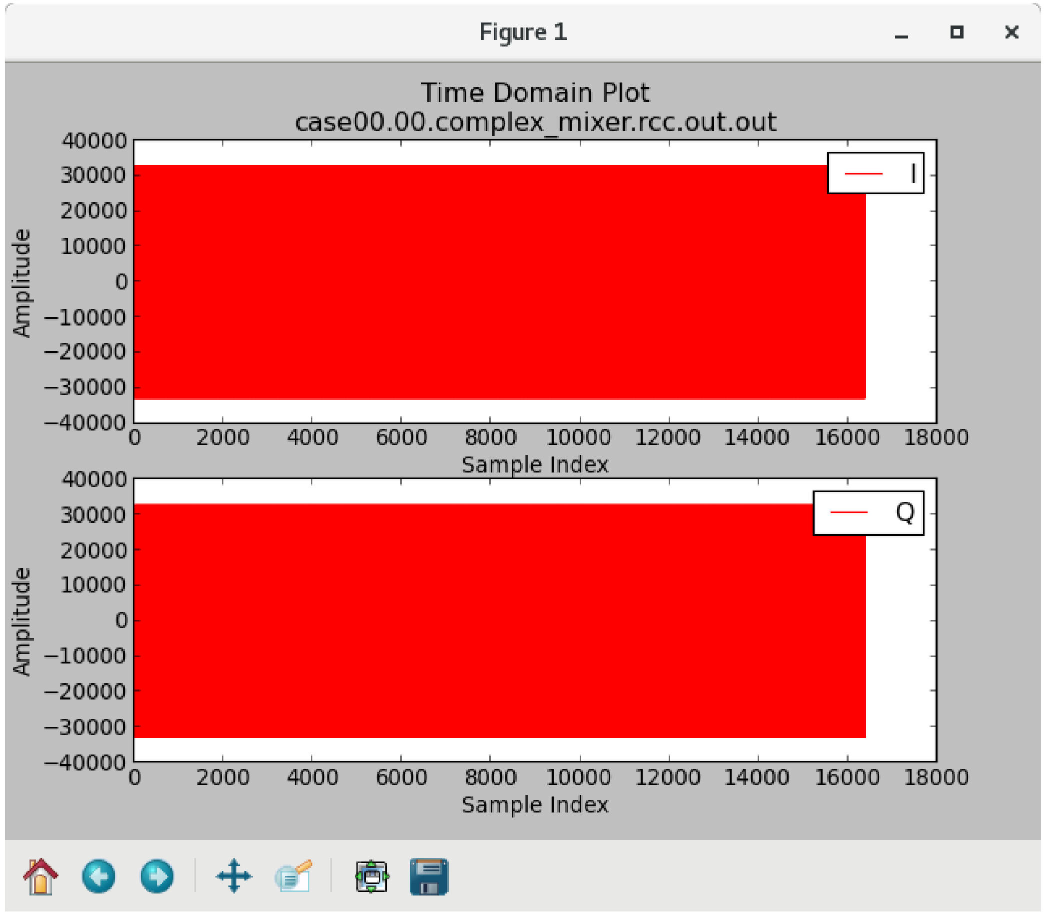
     $ ocpidev run --mode prep_run_verify –only-platforms xilinx13_4
     (This will run the unit test remotely (over ssh) on the Ettus E310's ARM)
     - Also try:
       - $ ocpidev run --mode prep_run_verify --only-platform xilinx13_4 --view {limits platforms to test}
       - $ ocpidev run --mode prep_run_verify {run on all available platforms, no plotting}
       - $ ocpidev run --mode verify {verify previous results}
       - $ ocpidev run --mode view {plot previous results}

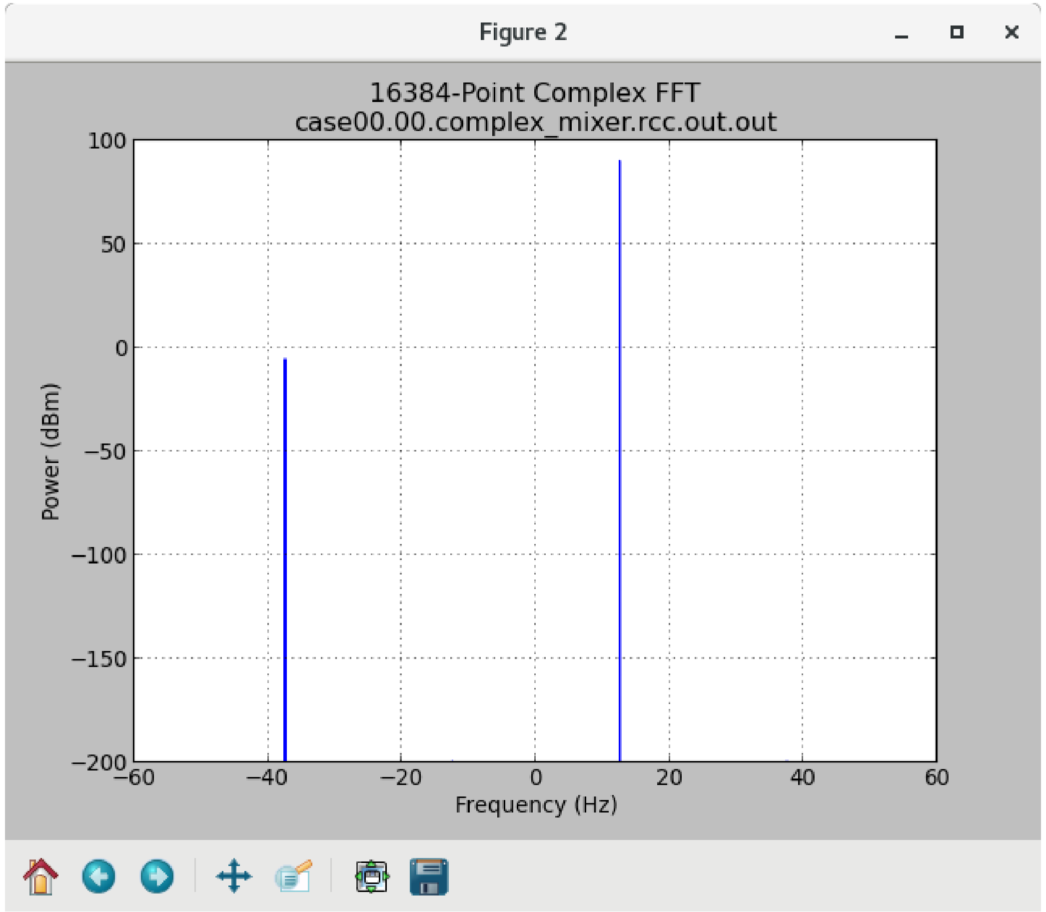
Expected input file plot for all cases*





Expected output file plot case00.00 (bypass)





Expected output file plot case00.01 (enabled)