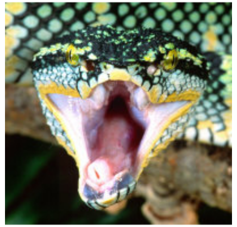
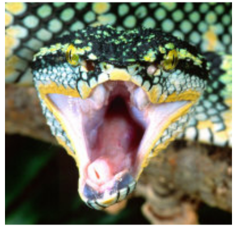


Lab 4: Complex Mixer

Integrating a 3rd party library into an RCC Worker

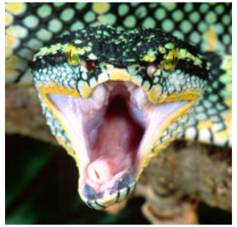
Objectives



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

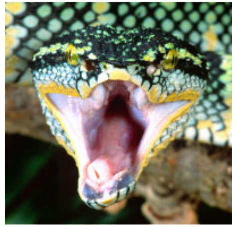
- Reiterate:
  - C++ conventions
    - Accessing port data and Properties
  - Framework interactions
    - RCC_ADVANCE vs. RCC_OK

Application Worker Development Flow



1. Protocol (OPS): Create new or select pre-existing
2. Component (OCS): Create new or select pre-existing
3. Create new App Worker (Modify OWD, Makefile, and source RCC/HDL 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

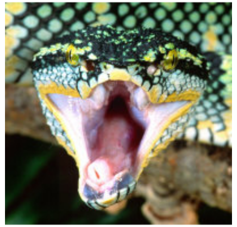


- The “Complex Mixer” component receives I/Q data and multiplies this signal by a tone that is generated using a Numerically Controlled Oscillator (NCO).

- 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.

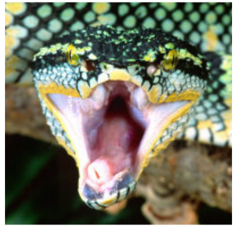
Step 1 – Protocol Selection



- Identify the OPS(s) declared by this component
  - Examine the “Component Ports” table in it's Component Datasheet
- Determine if OPS(s) exists
  - Current project's component library? /home/training/training_project/components/specs
  - Other project's components/specs/ directories within scope
    - Project Registry
    - ProjectDependencies in {my_project}/Project.mk
- If NO to all questions ⇨ Create new OPS

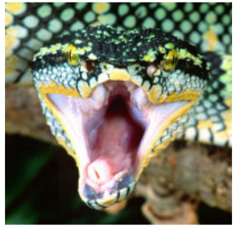
ANSWER: OPS XML file is available from framework (REUSE!)

Step 2 - Create Component



- Examine the Properties and Ports listed in the Component Datasheet
  - Use Properties/Ports information to answer the following questions
- Is the OCS XML file available in this project's component library?
  - HINT: Browse /home/training/training_project/components/specs/
- Is the OCS XML file available from the framework?
  - HINT: Browse IDE options
- If **NO** to all questions ⇨ custom OCS XML file must be created
- **ANSWER: Custom OCS XML file must be created.**

Step 2 - Create Component

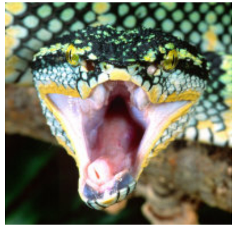


- The component datasheet is located in
  - `/home/training/provided/doc/Complex_Mixer.pdf`

- Review the component's datasheet and familiarize yourself with the properties and their functionality.

- Create a component called complex_mixer based on the data sheet's "Component Spec Properties" and "Component Ports"
  - **Note**: Ignore `data_select` which is a HDL worker only property.
  - **Note**: The `iqstream_protocol.xml` is located in Core Project

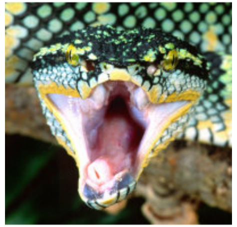
Step 3 - Create Worker



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

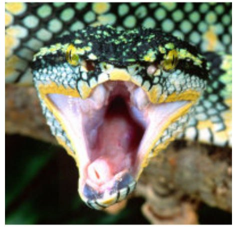
- In the OWD RCC Editor
  - Make sure to add initialize and release to the control operations
  - No additional worker properties and ports are needed from the datasheet because they will be inherited from the component-spec.

Step 3 - Create Worker (cont.)



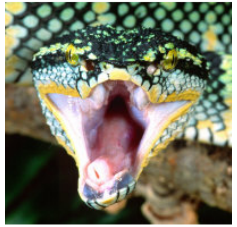
- Add the liquid prerequisite library.
  - In the RCC worker Makefile add “`RccStaticPrereqLibs=liquid`” before the line: “`include $(OCPI_CDK_DIR)/include/worker.mk`”

Step 3 - Write the Worker's Code



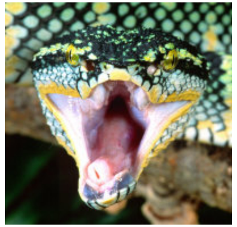
- Copy complex_mixer.cc
  - From: /home/training/provided/lab4/
  - To: /home/training/training_project/components/complex_mixer.rcc
- Update any “???” in the source with the correct code

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 Δθ).
  - `nco_crcf_set_phase(q,theta)`
    - sets the internal `nco` phase to θ

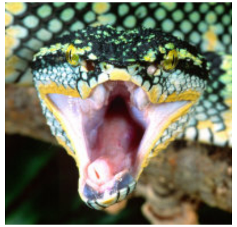
Liquid DSP NCO API (for reference)



- 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`

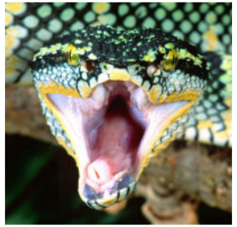
```
liquid_float_complex sample;
sample.I = 0;
sample.Q = 0;
```

Step 4 - Building the App Worker for x86 and ARM

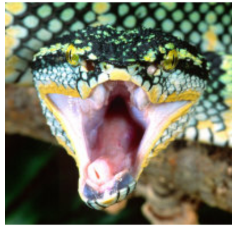


1. Use the IDE to “**Add**” the App Worker to the Project Operations Panel
2. **Highlight** “centos7” *and* “xilinx13_3” in RCC Platforms panel
3. **Check** “Assets”
4. **Click** “Build”
5. Review the Console window messages
6. Fix any syntax errors and repeat

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)

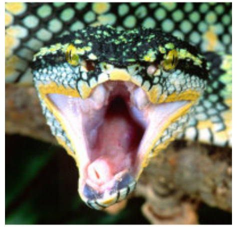


Step 5(a) - Create Unit Test

- Create Unit Test
  - **File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard → Unit Test**
  - **Add to Project:** *training_project*
  - **Add to Library:** *components*
  - **Component Spec*:*** *complex_mixer-spec.xml*

- **OR** in a terminal window
  - `ocpidev create test complex_mixer`

Step 5(a) - Create Unit Test



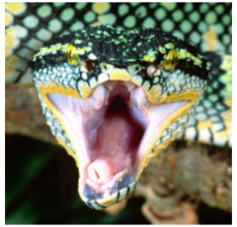
- Copy unit test scripts: generate.py, verify.py, and view.sh

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

- Update complex_mixer-test.xml

```
<tests useHDLFileIo='true'>
  <input port='in' script='generate.py 100 12.5 32767 16384' messagesize='8192'/>
  <output port='out' script='verify.py 100 16384' view='view.sh'/>
  <property name='phs_inc' values='-8192'/>
  <property name='enable' values='0,1'/>
</tests>
```

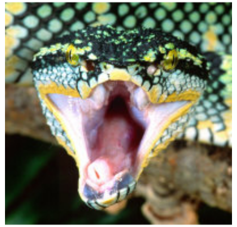
Step 6(a) - Build Unit Test (x86)



- 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)

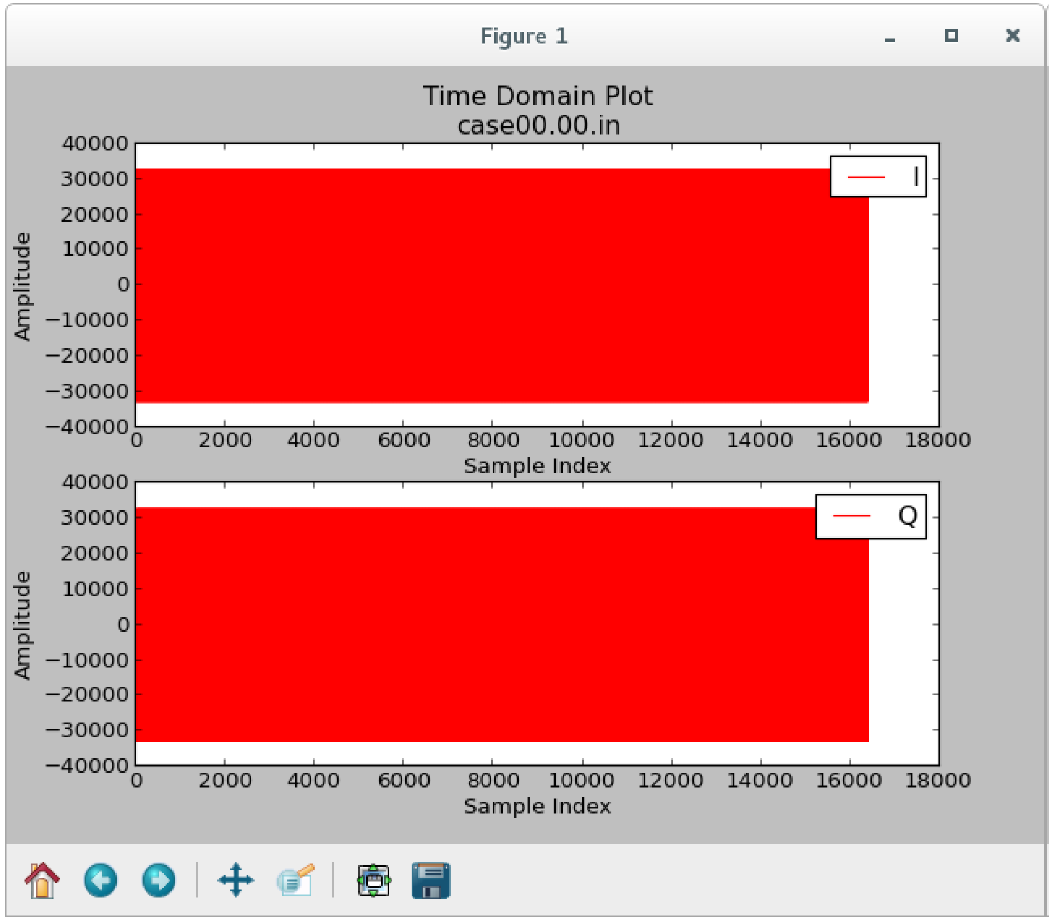


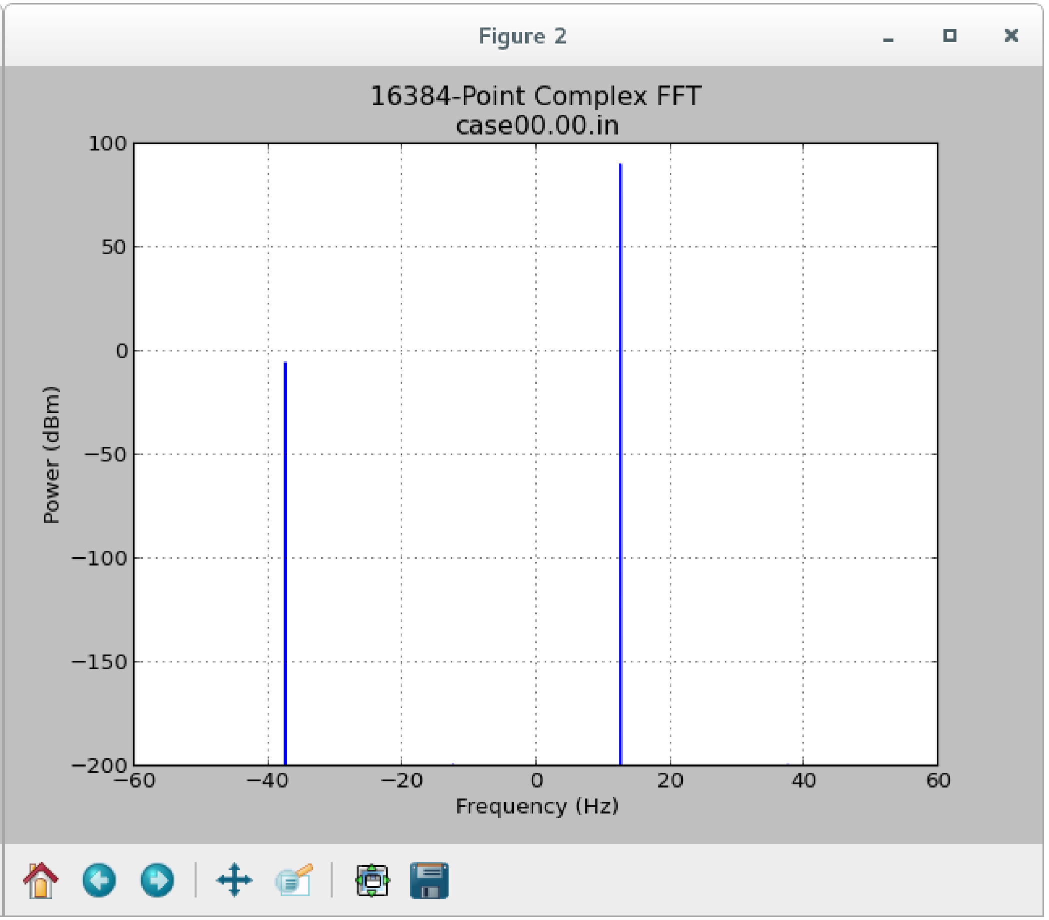
- Click “prep + run + verify” button to run the test
- Click the “view” button to view the test results
- **OR** in a terminal window, browse to complex_mixer.test/ and execute

$ make run View=1

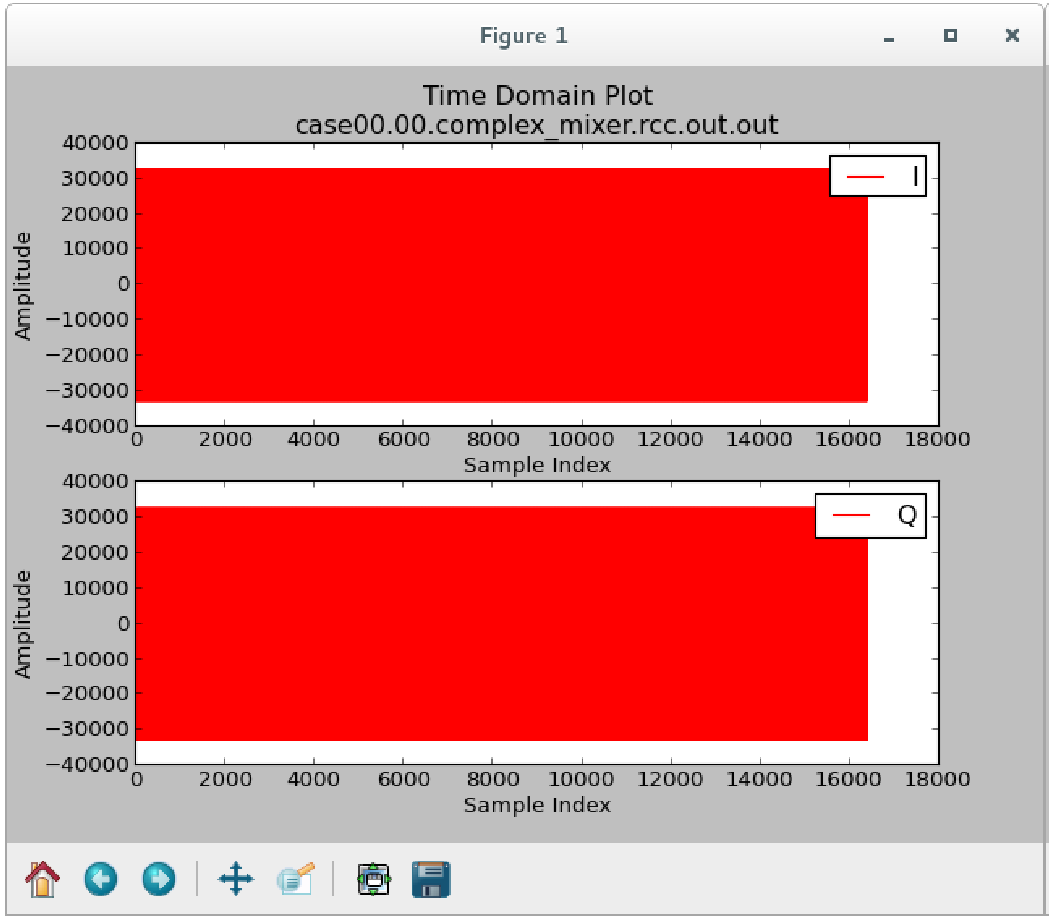
  - This uses the default centos7
- The test should run quickly. Upon completion you should see “**PASSED**” along with input and output plots (expected results in following slides)
- Fix any errors with your component to make sure the testbench passes

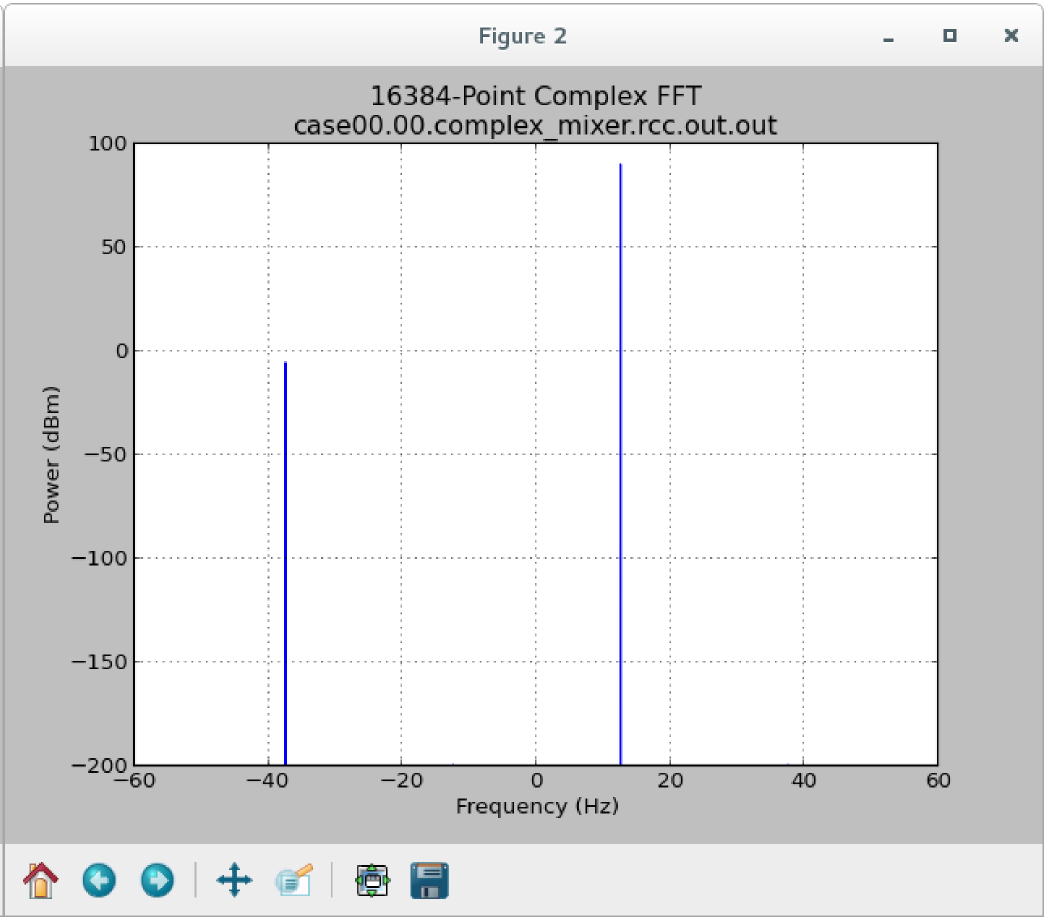
Expected input file plot



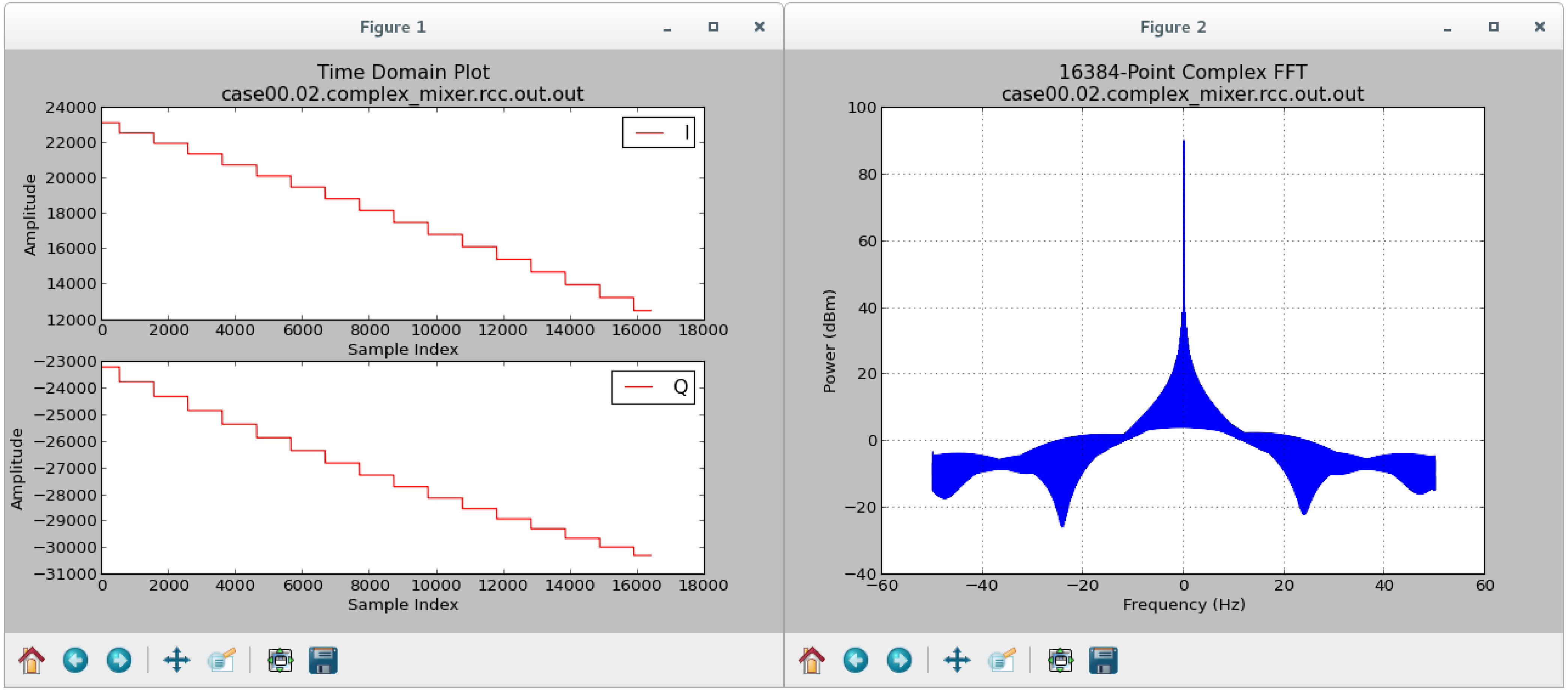


Expected output file plot case 0 (bypass)

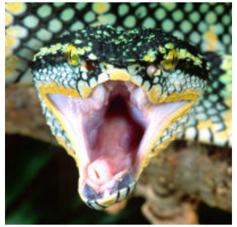




Expected output file plot case 1 (enabled)

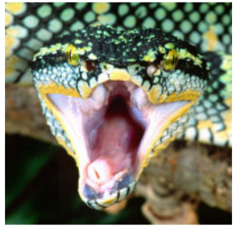


Step 5(b) – 7(b) Xilinx13_3 - ARM



- 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

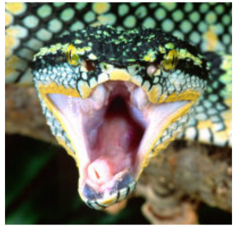


- Located in “complex_mixer.test/” directory
    - Same as used for CentOS7
        - **REUSE!**

- Reuse complex_mixer-test.xml

```
<tests useHDLFileIo='true'>
  <input port='in' script='generate.py 100 12.5 32767 16384' messagesize='8192'/>
  <output port='out' script='verify.py 100 16384' view='view.sh'/>
  <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

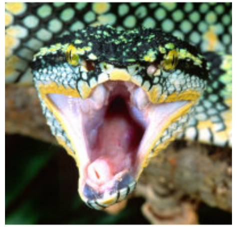
  1. Use the IDE to “**Add**” the Unit Test to the Project Operations panel

  2. Highlight “xilinx13_3” the RCC Platforms panel

  3. Select “Tests” Radio button
  4. Click “gen + build”
  5. Review the Console window messages and address any errors

- Observe possibly-updated 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)



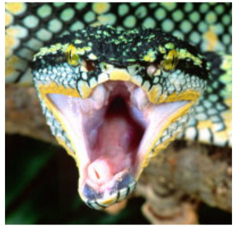
- Setup deployment platform
  1. Connect to serial port via USB on rear of Matchstiq-Z1 using host

     -
       ```
       screen /dev/matchstiq_z1_0 115200
       ```
  2. Boot and login into PetaLinux
     - User/Password = root:root
  3. Verify host and Matchstiq-Z1 have valid IP addresses
     - For training, they should both be on the same subnet
  4. Run setup script on Matchstiq-Z1

     -
       ```
       source /mnt/card/opencpi/mynetsetup.sh <host ip address>
       ```

More detail on this process can be found in the **Matchstiq-Z1 Getting Started Guide** document

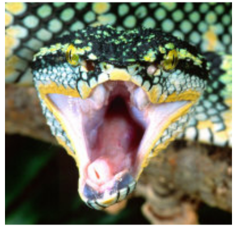
Step 7(b) - Run Unit Test (ARM) (cont.)



- GUI approach to running unit tests on remote platforms:
  - In the Project Operations Panel
    - **Check** “Tests”
    - **Select** “remotes” radio button
    - **Click** “+remotes”
    - **Change** remote variable text to use matchstiq’s ip and point to the training project:
      {IP of Matchstiq-Z1}=root=root=/mnt/training_project
    - **Select** the newly created remote. This will be the target remote test system. Unselected remotes will not be run.
- OR Prior to launching the IDE, OCPI_REMOTE_TEST_SYSTEMS must be set

  $ **export OCPI_REMOTE_TEST_SYSTEMS=<IP of Matchstiq-Z1>=root=root=/mnt/training_project**

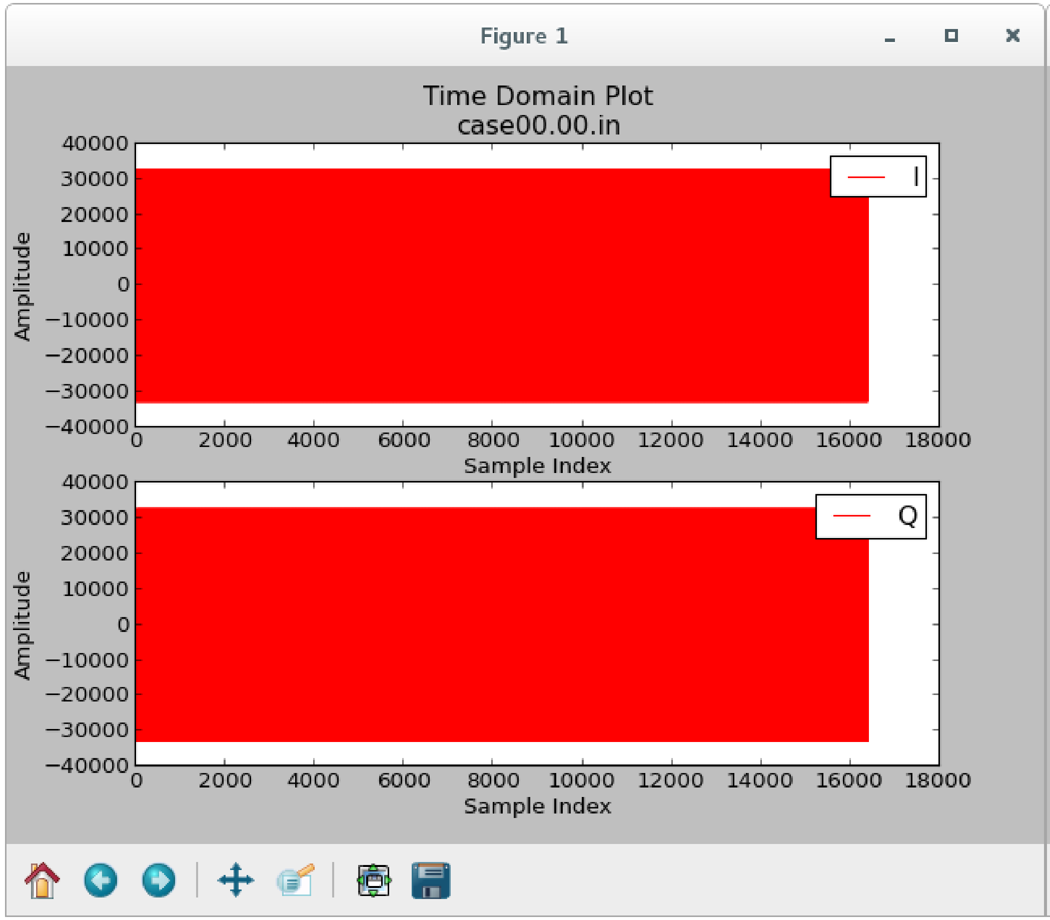
Step 7(b) - Run Unit Test (ARM) (cont.)

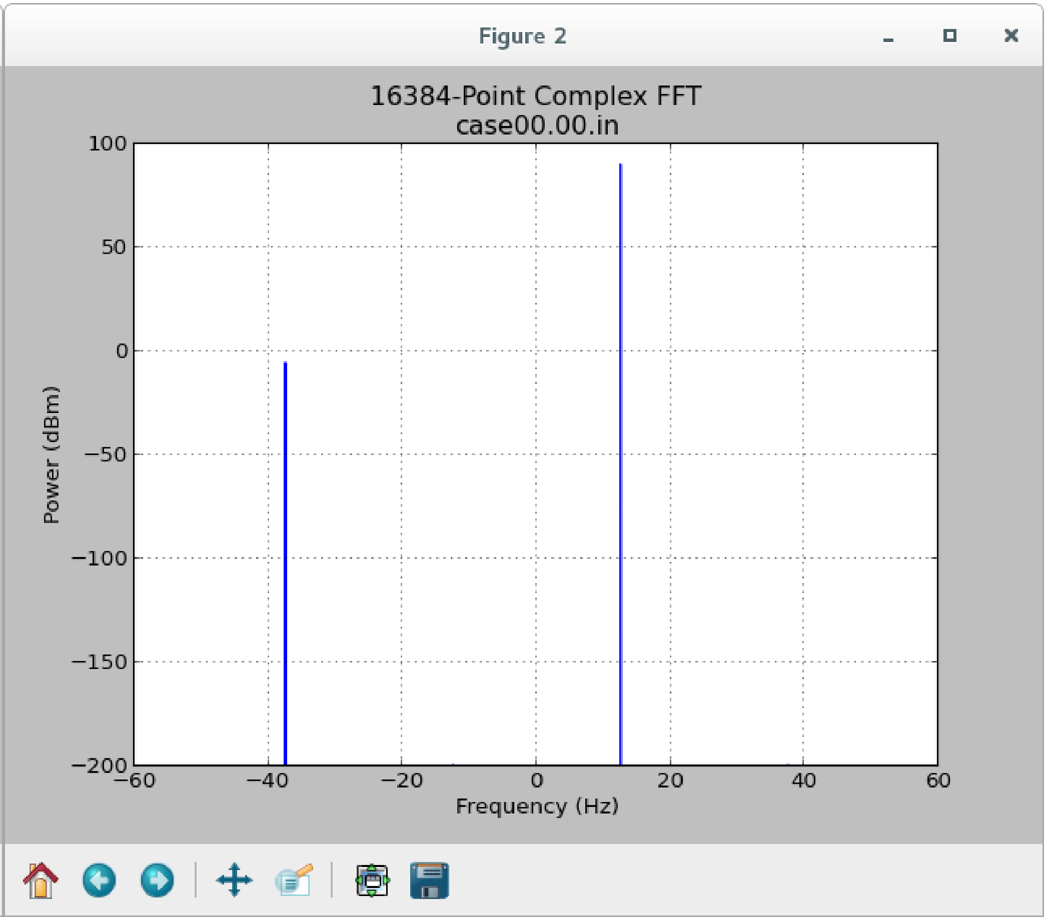


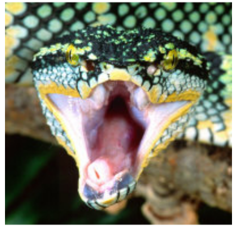
- Run Unit Test Suite for target hardware platform
  1) Use the IDE to "**Add**" the Unit Test to the Project Operations panel
  2) **Highlight** "xilinx13_3" in the <u>RCC Platforms</u> panel
  3) **Check** "run view script"
  4) **Click** "prep+run+verify" to Run Tests
  5) Review the Console window messages and address any errors
- Terminal operations:
  In a terminal window, execute within the {component}.test/

```
$ make run {run on all available platforms, no plotting, discard resulting simulation}
$ make run OnlyPlatforms=xilinx13_3 View=1
$ make verify {verify previous results}
$ make view {plot previous results}
```

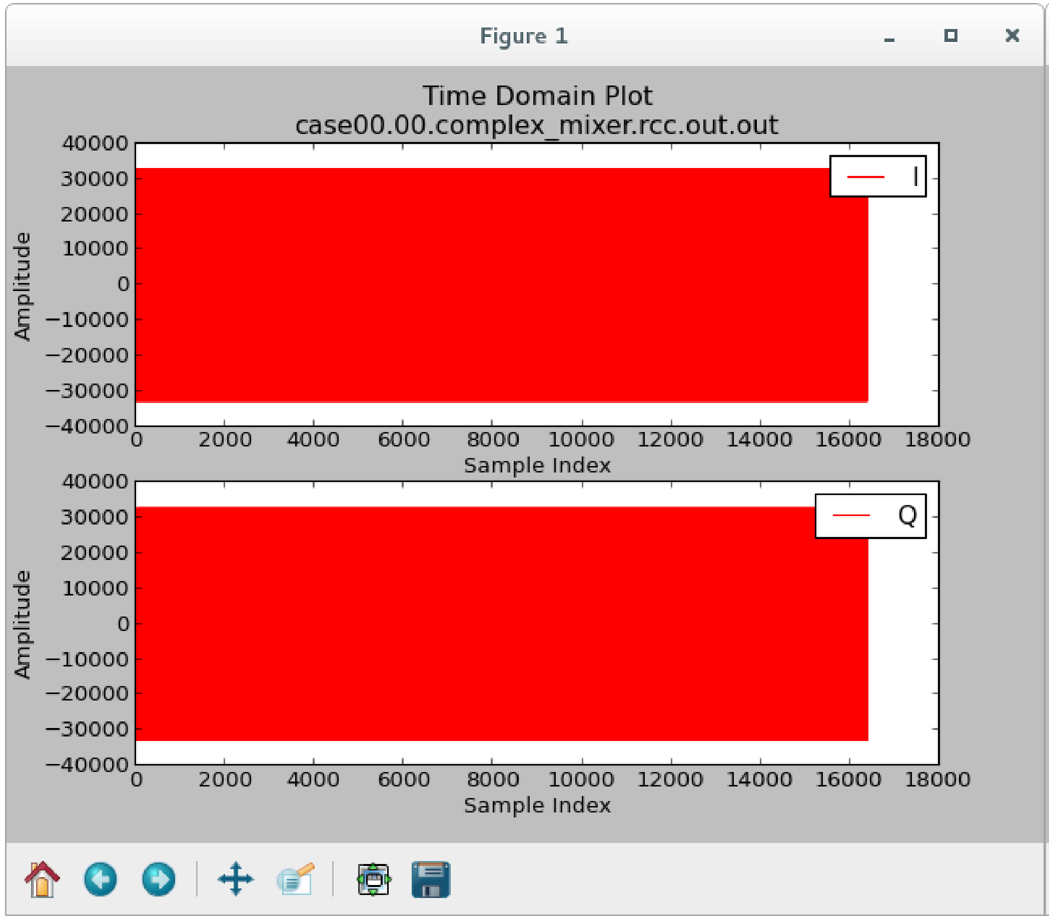
Expected input file plot

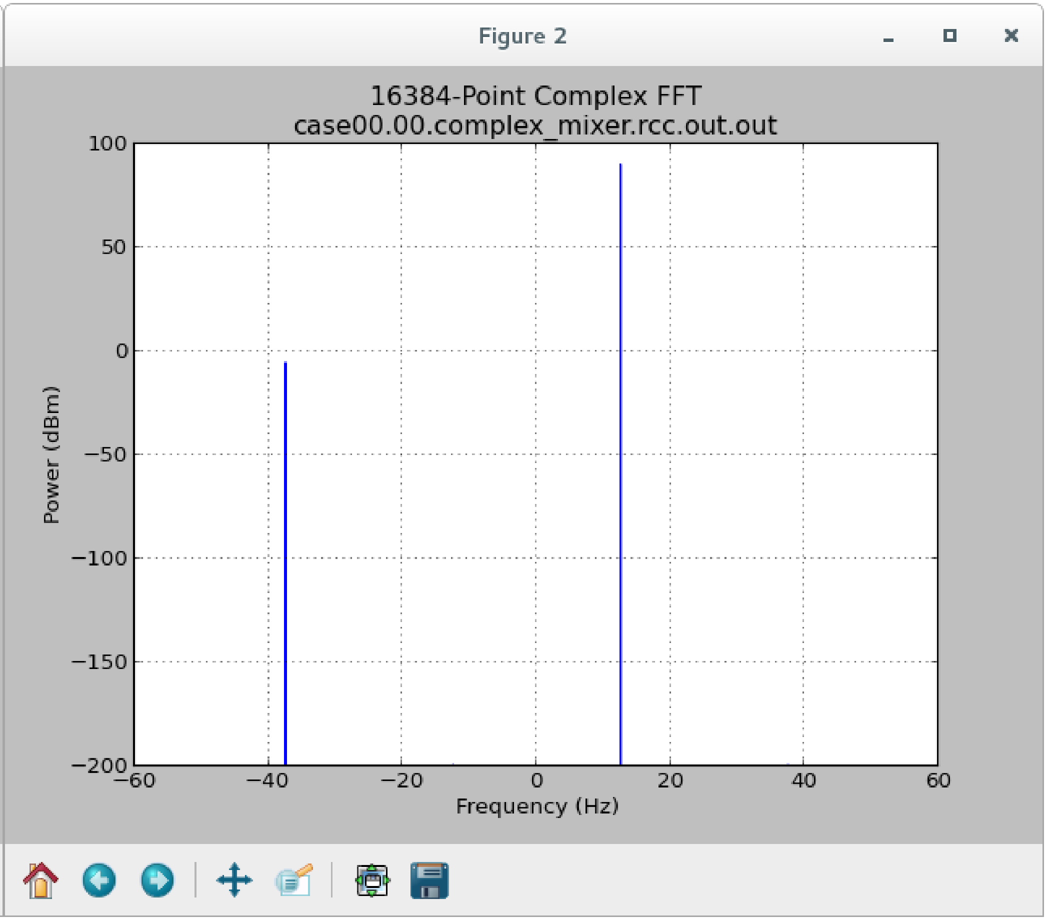




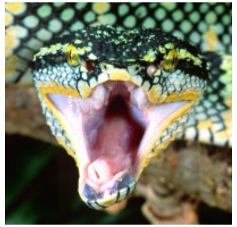


Expected output file plot case 0 (bypass)





Expected output file plot case 1 (enabled)



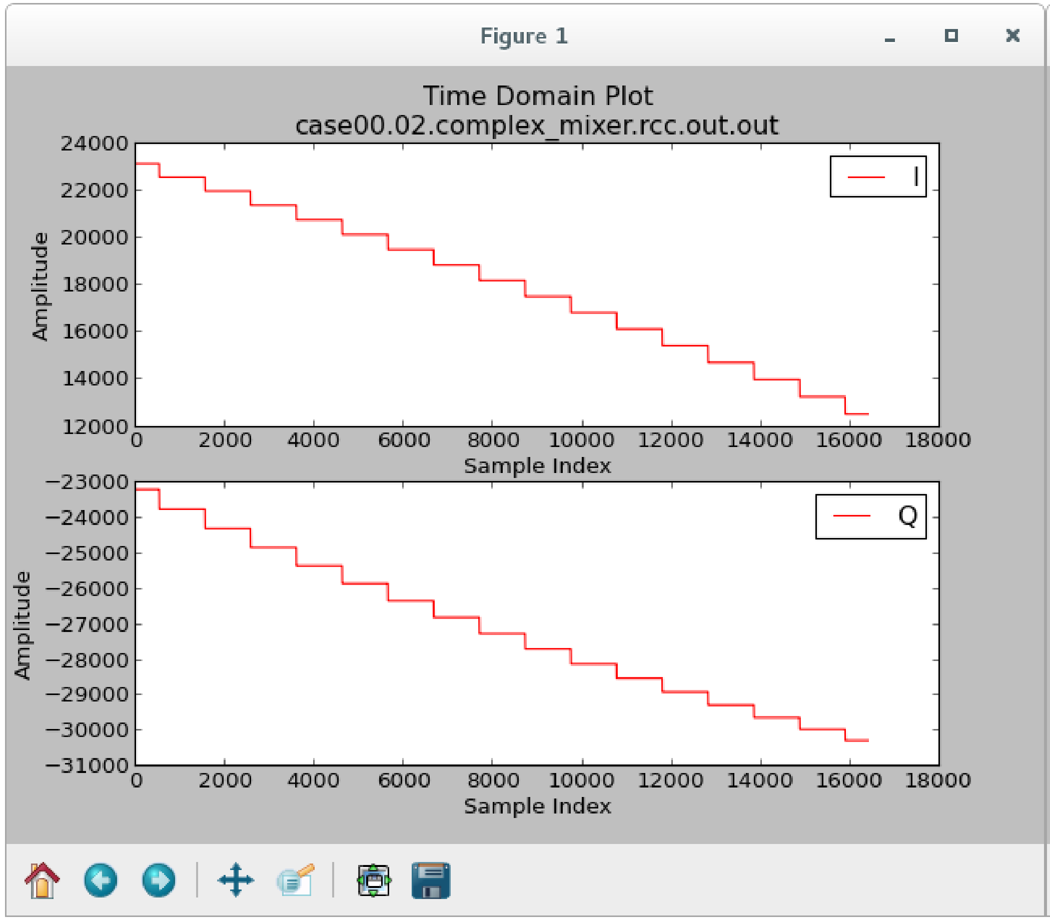


Figure 1
Time Domain Plot
case00.02.complex_mixer.rcc.out.out
Amplitude
I
Sample Index
Amplitude
Q
Sample Index

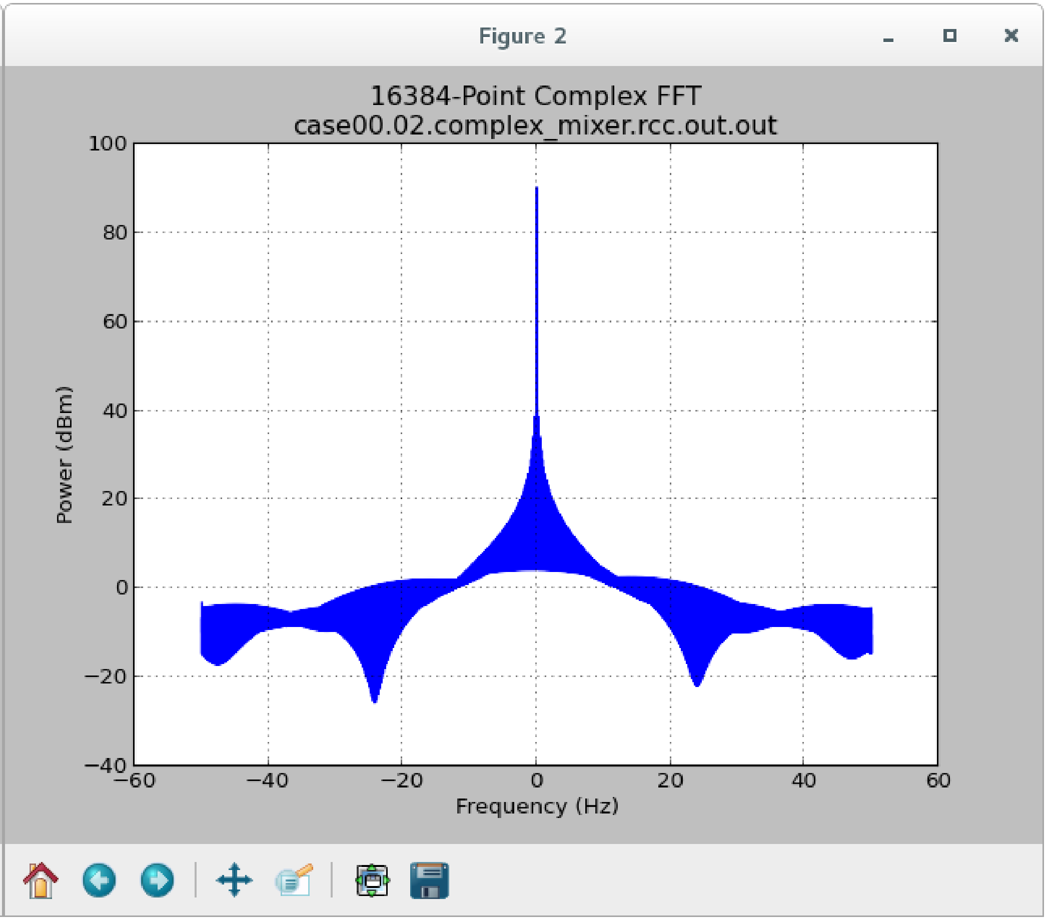


Figure 2
16384-Point Complex FFT
case00.02.complex_mixer.rcc.out.out
Power (dBm)
Frequency (Hz)