

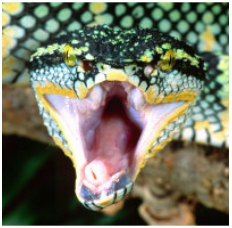
Lab 2:

OpenCPI Application Development & Integration

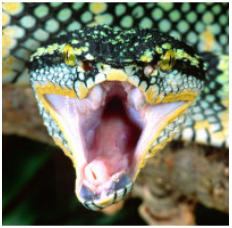


Objectives

1. Create application using pre-existing workers from an imported project
2. Identify which components of the application can be deployed on the FPGA versus processor
3. Create and build two different HDL Assemblies which can be used by the application
4. Deploy the application with ocpirun using both of the HDL Assemblies



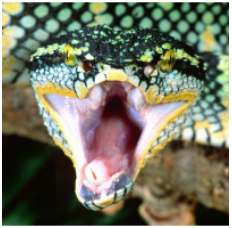
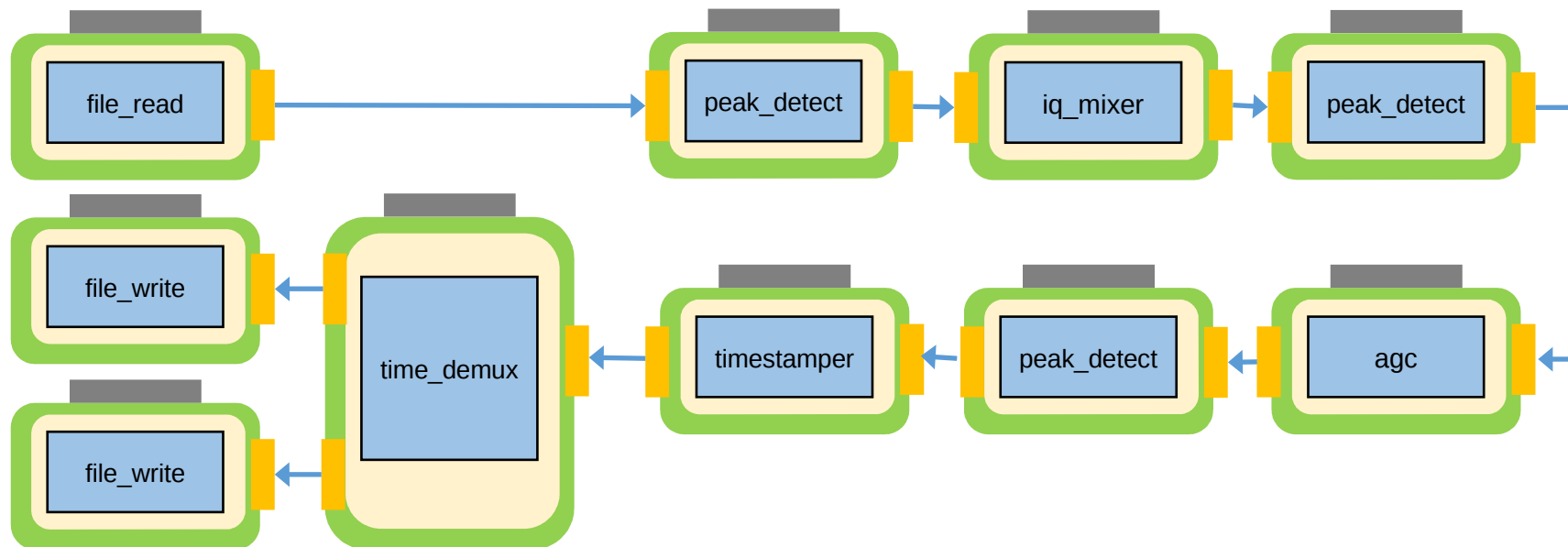
Overview

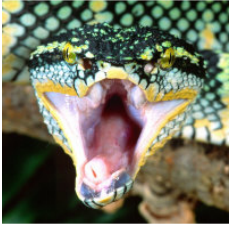


- A common use case for OpenCPI is the reuse of components from multiple libraries to construct applications, that are deployed onto heterogeneous systems.
- Once the required components for an application have been determined, the subset of components which will execute in the FPGA must be identified, specified, and built prior to running the application. This portion of the application is referred to as the *HDL Assembly*.
- Applications can be executed to leverage various HDL Assemblies.

Overview

- The application in this lab will use components developed during this training in addition to components included with OpenCPI
- The application will:
 - read I/Q sample data from a file
 - perform complex mixing, automatic gain control, and timestamping on the I/Q sample data
 - write the resulting I/Q sample data and timestamps to 2 different files



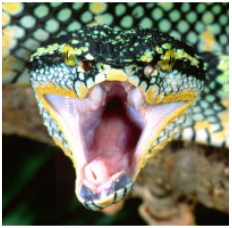


Part 1

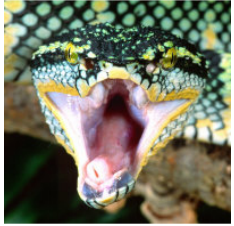
Create application using pre-existing workers from an imported project

Step 1.1

- Copy the training_project to the ~ directory
 - This project contains the components which will be created over the course of the training
 - In a terminal window:
 - `$ cp -r ~/provided/lab2/training_project/ ~`



Step 1.2



1. Launch IDE

2. Import training_project:

1. To import project into eclipse:
 1. File → Import...
 1. "Existing Projects into Workspace"
2. Refresh "OpenCPI Projects"
3. Right-click on "ocpi.training" and select "register project"

3. Open Terminal

4. Confirm registry

1. `ocpidev show registry`

```
$ ocpidev show registry
Project registry is located at: /opt/opencpi/project-registry
```

Project	Package-ID	Path to Project	Valid/Exists
ocpi.assets		/home/training/ocpi_projects/assets	True
ocpi.assets_ts		/home/training/ocpi_projects/assets_ts	True
ocpi.core		/home/training/ocpi_projects/core	True
ocpi.bsp.bsp_e310		/home/training/ocpi_projects/bsp_e310	True
ocpi.training		/home/training/training_project	True

Step 1.3

- Create new Application using IDE called lab2_app
 - XML only
- Add the components in this table to the application
 - Do so in the order they are listed here
 - Remove "nothing"
- Hints
 - * Remember to name components with multiple instances uniquely

Component Specs Required	
Name	Project : Library
file_read_spec.xml	Core : components
peak_detector-spec.xml*	Training : components
complex_mixer-spec.xml	Training : components
peak_detector-spec.xml*	Training : components
agc_complex-spec.xml	Training : components
peak_detector-spec.xml*	Training : components
timestamperspec.xml	Assets : components/util_comps
time_demux-spec.xml	Training : components
file_write_spec.xml*	Core : components
file_write_spec.xml*	Core : components



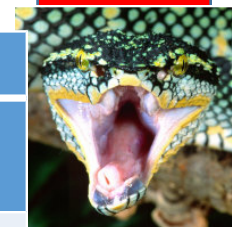
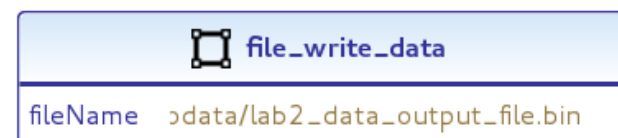
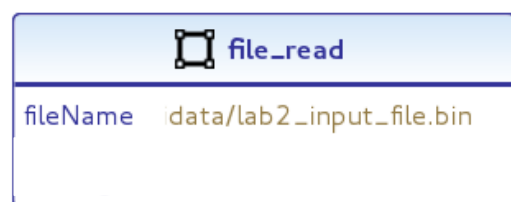
Step 1.4

- Specify property values

- When not specified in the XML, the properties of a component assume a default value

- No ValueFiles in this lab**

Property Values Required		
Component	Property Name	Value
file_read	fileName	idata/lab2_input_file.bin
agc_complex	mu	0x144E
agc_complex	ref	0x1B26
file_write_time	fileName	odata/lab2_time_output_file.bin
file_write_data	fileName	odata/lab2_data_output_file.bin



Step 1.5

- Make connections

1. Click "Advanced Connection" on Palette Menu
2. Click originating instance
3. Click destination instance
4. Populate "Port Name" fields for connections

- **time_demux uses non-default Port Names**

- timestamp: "out" → time_demux: "Mux_In"
- time_demux: "Data_Out" → file_write_data: "in"
- time_demux: "Time_Out" → file_write_time: "in"

- All other components use default Port Names (Expected XML on next slide)

Application

3

file_write_time

fileName odata/lab2_time_output_file.bin

2

time_demux

file_write_data

fileName odata/lab2_data_output_file.bin

1

Palette

Select

+ Marquee

Connections

Advanced Connection

Simple Connection

Objects

Instance

Application Details Source

Define Connection

Connection Name:

4

Ports:

Instance	Port Name
time_demux	Data_Out
file_write_data	in
file_write_time	in

Step 1.5

<!-- The lab2_app application xml file -->

<Application>

<Instance Component="ocpi.core.file_read" Name="file_read">

<Property Name="filename" Value="idata/lab2_input_file.bin"></Property>

</Instance>

<Instance Component="ocpi.core.file_write" Name="file_write_time">

<Property Name="filename" Value="odata/lab2_time_output_file.bin"></Property>

</Instance>

<Instance Component="ocpi.core.file_write" Name="file_write_data">

<Property Name="filename" Value="odata/lab2_data_output_file.bin"></Property>

</Instance>

<Instance Component="ocpi.training.peak_detector" Name="peak_detector_file_out"></Instance>

<Instance Component="ocpi.training.complex_mixer" Name="complex_mixer"></Instance>

<Instance Component="ocpi.training.peak_detector" Name="peak_detector_agc_in"></Instance>

<Instance Component="ocpi.training.agc_complex" Name="agc_complex">

<Property Name="mu" Value="0x144e"></Property>

<Property Name="ref" Value="0x1b26"></Property>

</Instance>

<Instance Component="ocpi.training.peak_detector" Name="peak_detector_agc_out"></Instance>

<Instance Component="ocpi.training.time_demux" Name="time_demux"></Instance>

<Instance Component="ocpi.assets.util_comps.timestampers" Name="timestampers"></Instance>

<Connection>

<Port Instance="file_read" Name="out"></Port>

<Port Instance="peak_detector_file_out" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="peak_detector_file_out" Name="out"></Port>

<Port Instance="complex_mixer" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="complex_mixer" Name="out"></Port>

<Port Instance="peak_detector_agc_in" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="peak_detector_agc_in" Name="out"></Port>

<Port Instance="agc_complex" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="agc_complex" Name="out"></Port>

<Port Instance="peak_detector_agc_out" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="peak_detector_agc_out" Name="out"></Port>

<Port Instance="timestampers" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="time_demux" Name="Time_Out"></Port>

<Port Instance="file_write_time" Name="in"></Port>

</Connection>

<Connection>

<Port Instance="time_demux" Name="Data_Out"></Port>

<Port Instance="file_write_data" Name="in"></Port>

</Connection>

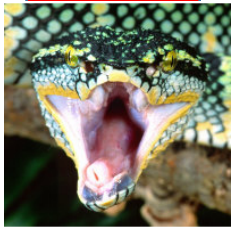
<Connection>

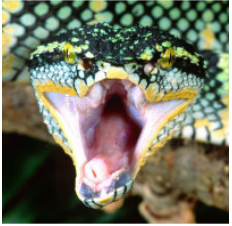
<Port Instance="timestampers" Name="out"></Port>

<Port Instance="time_demux" Name="Mux_in"></Port>

</Connection>

</Application>



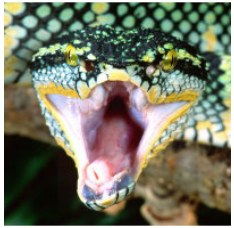
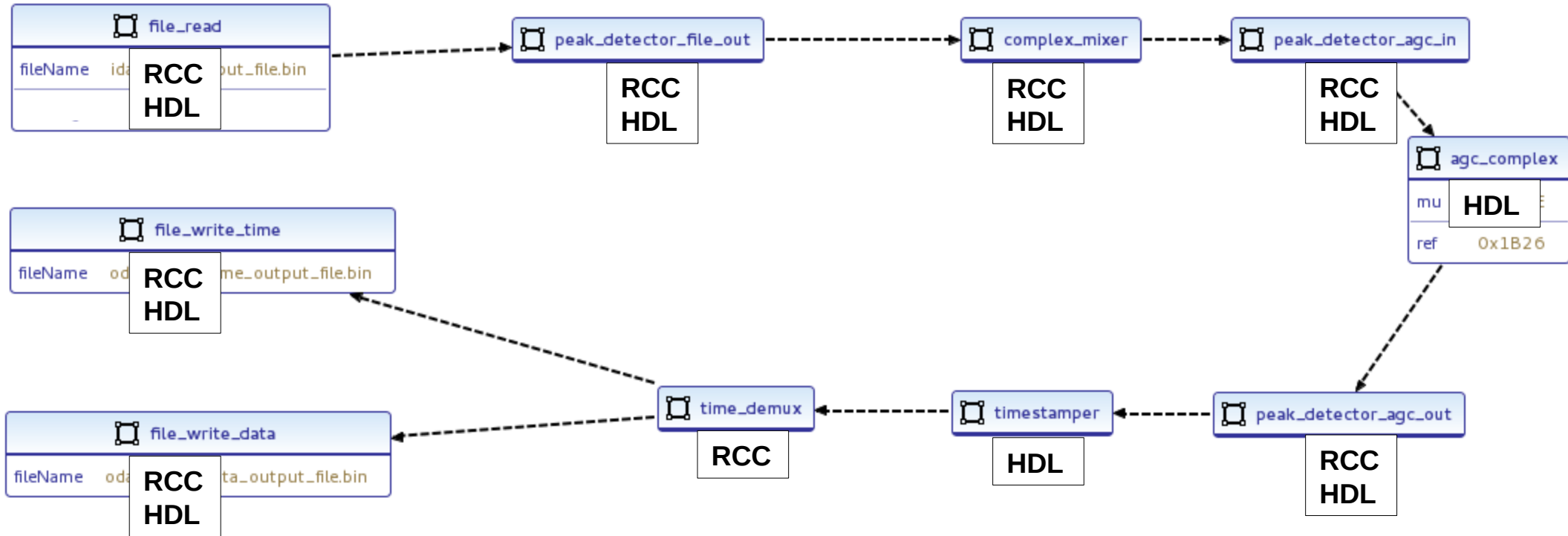


Part 2

Identify which components of the application can be deployed on the FPGA

Which components have HDL implementations?

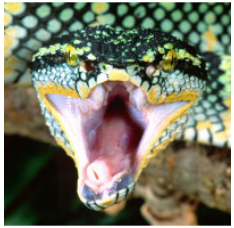
- Determine which workers have been implemented for use in FPGAs
 - Part 1 selected which **components** were needed for the application. This part identifies the **workers** which will be used.

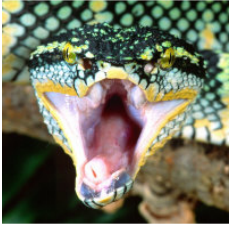


Which components make sense to use on FPGA?

- Select implementations to build
 - Depending on the resources of the intended deployment platform, more HDL workers than RCC workers may be desirable or vice versa
- This lab will build and execute two implementations
 - Most possible HDL workers
 - Most possible RCC workers
- Application is the same for both!

Most possible HDL workers	Most possible RCC workers
file_read.rcc	file_read.rcc
peak_detector.hdl	peak_detector.rcc
complex_mixer.hdl	complex_mixer.rcc
peak_detector.hdl	peak_detector.rcc
agc_complex.hdl	agc_complex.hdl
peak_detector.hdl	peak_detector.hdl
timestamp.hdl	timestamp.hdl
time_demux.rcc	time_demux.rcc
file_write.rcc	file_write.rcc
file_write.rcc	file_write.rcc
RCC Worker	
HDL Worker	

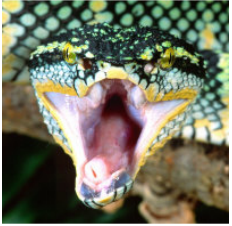
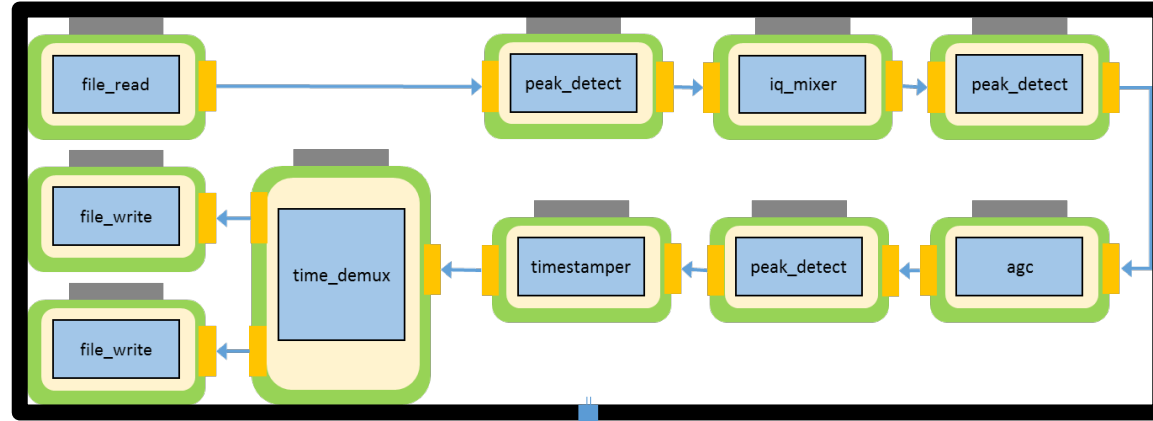




Part 3

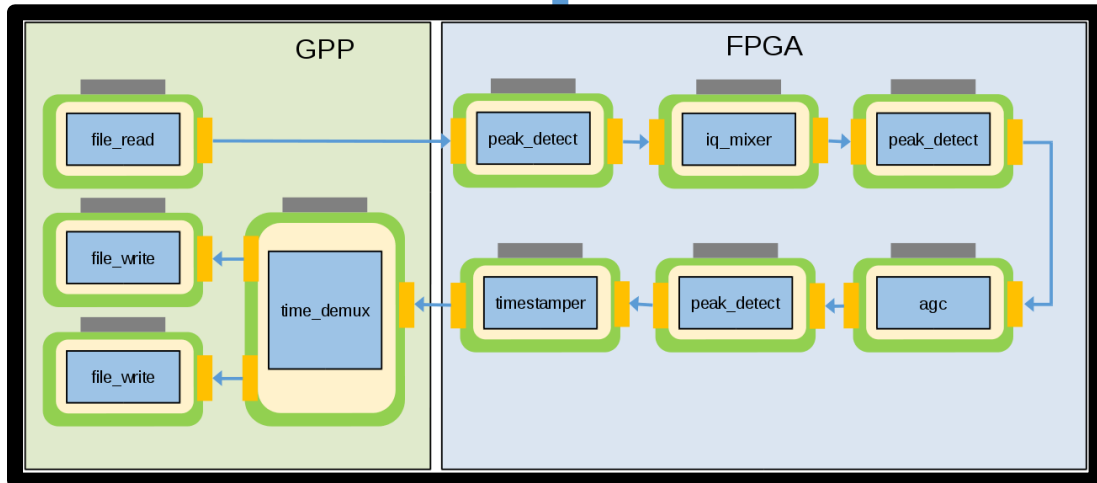
Create and build two different HDL Assemblies which can be used by the application

One Application

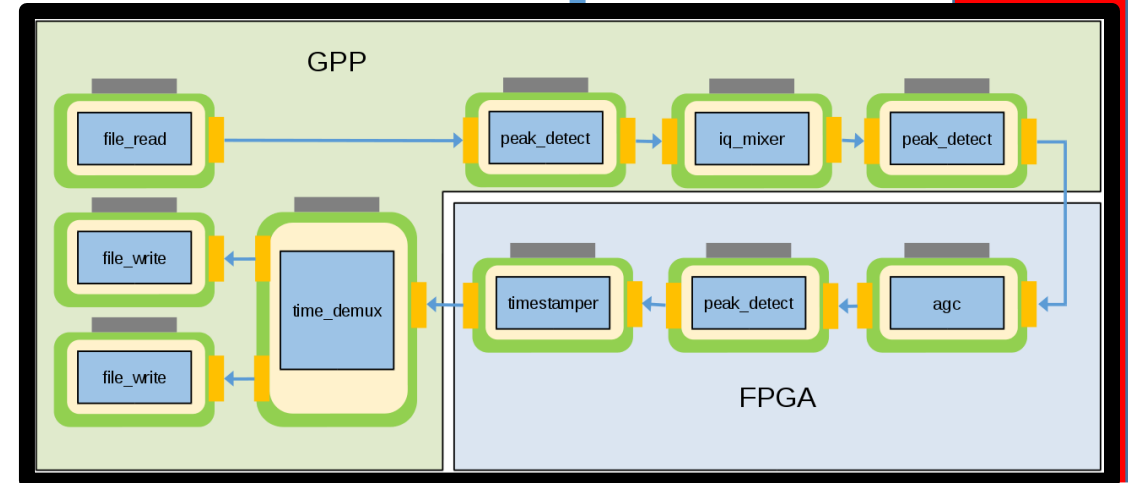


Open
CPI

Multiple Implementations



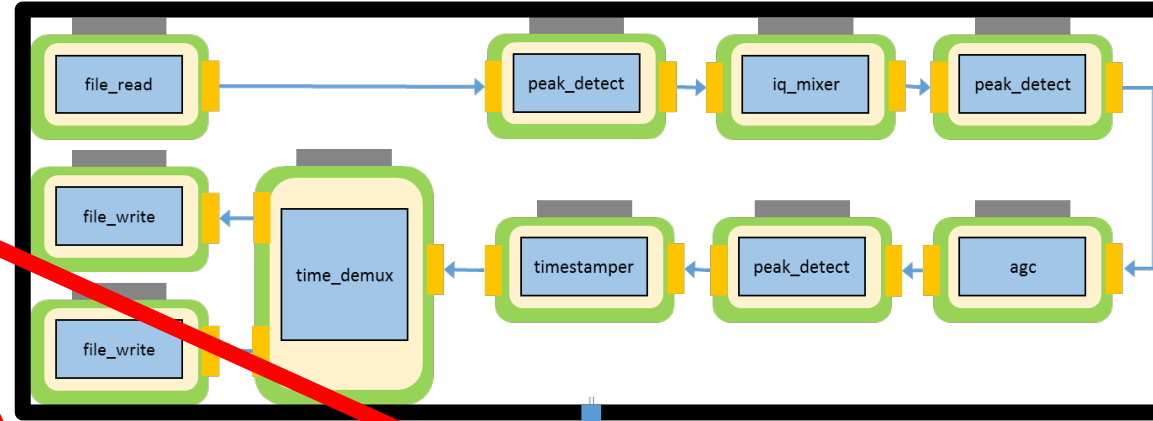
Most possible HDL workers



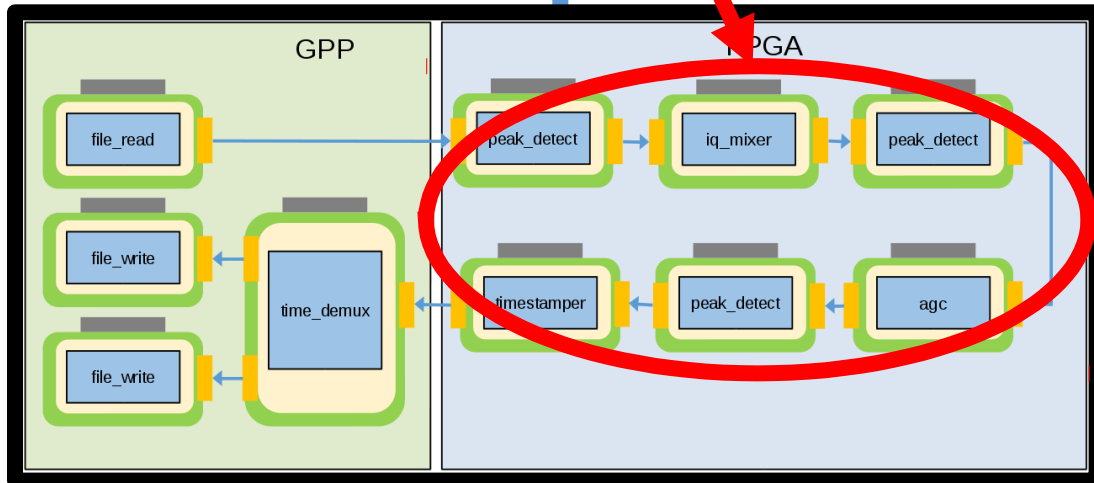
Most possible RCC workers

One Application

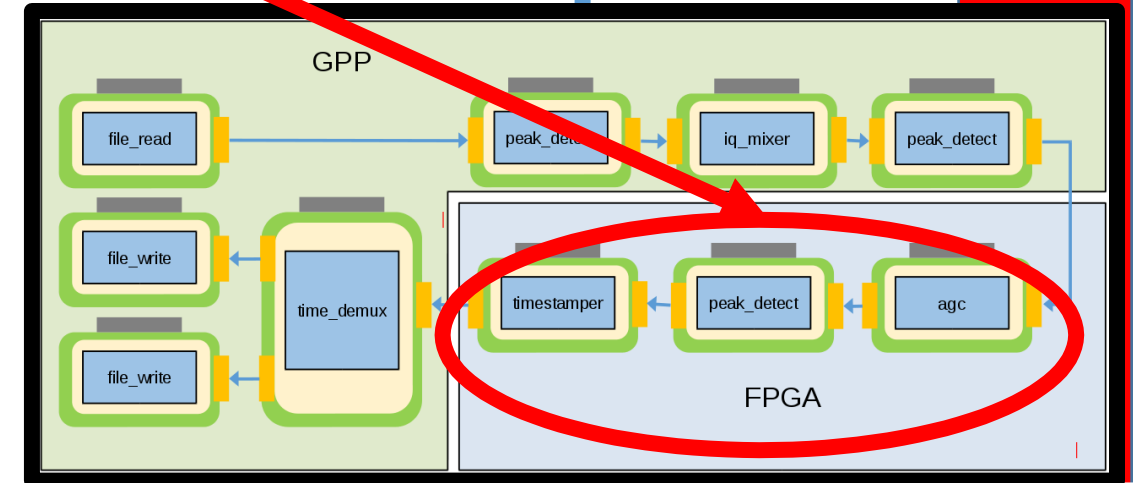
Need to
specify and
build this
piece



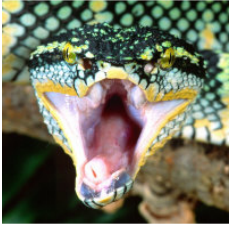
Multiple Implementations



Most possible HDL workers



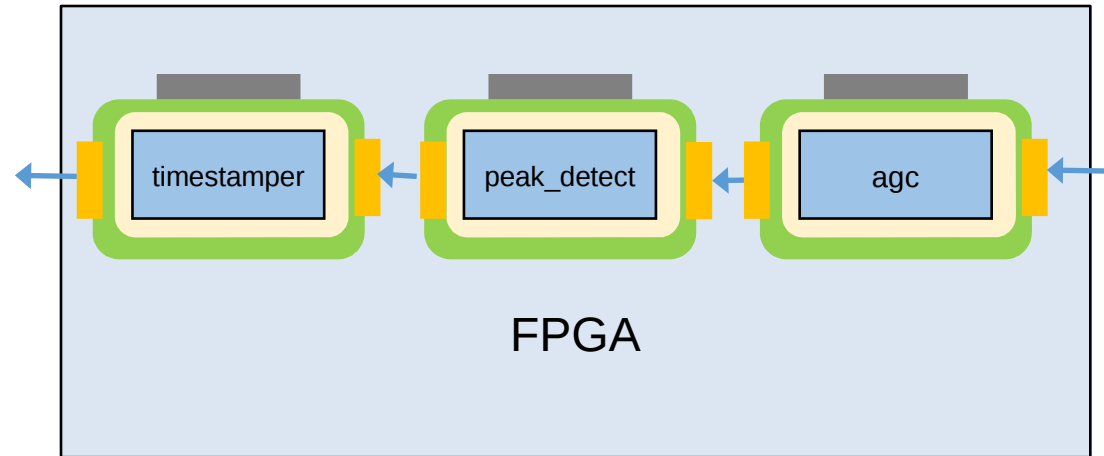
Most possible RCC workers



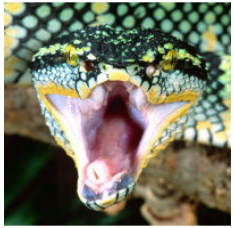
OpenCPI

What is specified in an HDL Assembly?

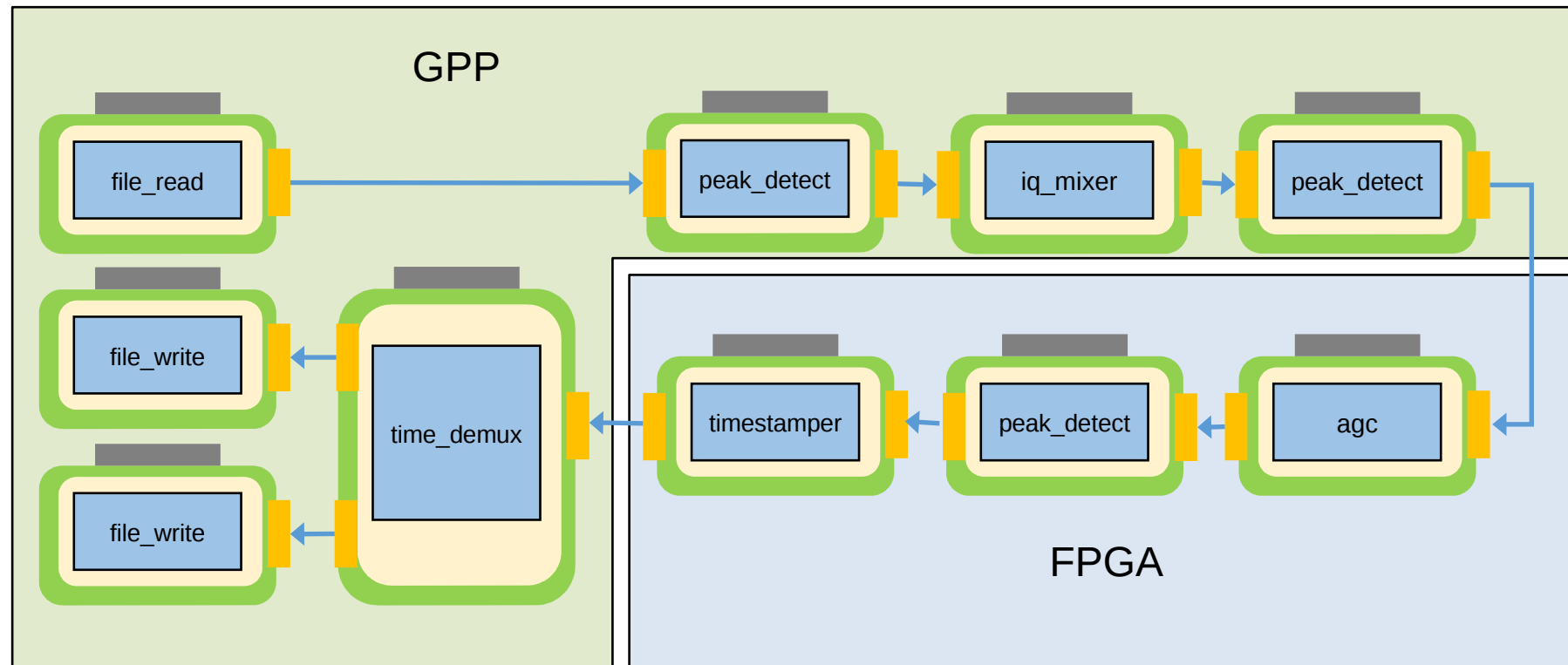
1. The HDL workers being used
2. The connections
 - Between HDL workers
 - Connections external to the assembly



```
<HdlAssembly>
  <Instance worker="agc_complex" name="agc_complex" external="in"></Instance>
  <Instance worker="timestamper" name="timestamper" external="out"></Instance>
  <Instance worker="peak_detector" name="peak_detector"></Instance>
  <Connection>
    <Port instance="agc_complex" name="out"></Port>
    <Port instance="peak_detector" name="in"></Port>
  </Connection>
  <Connection>
    <Port instance="peak_detector" name="out"></Port>
    <Port instance="timestamper" name="in"></Port>
  </Connection>
</HdlAssembly>
```



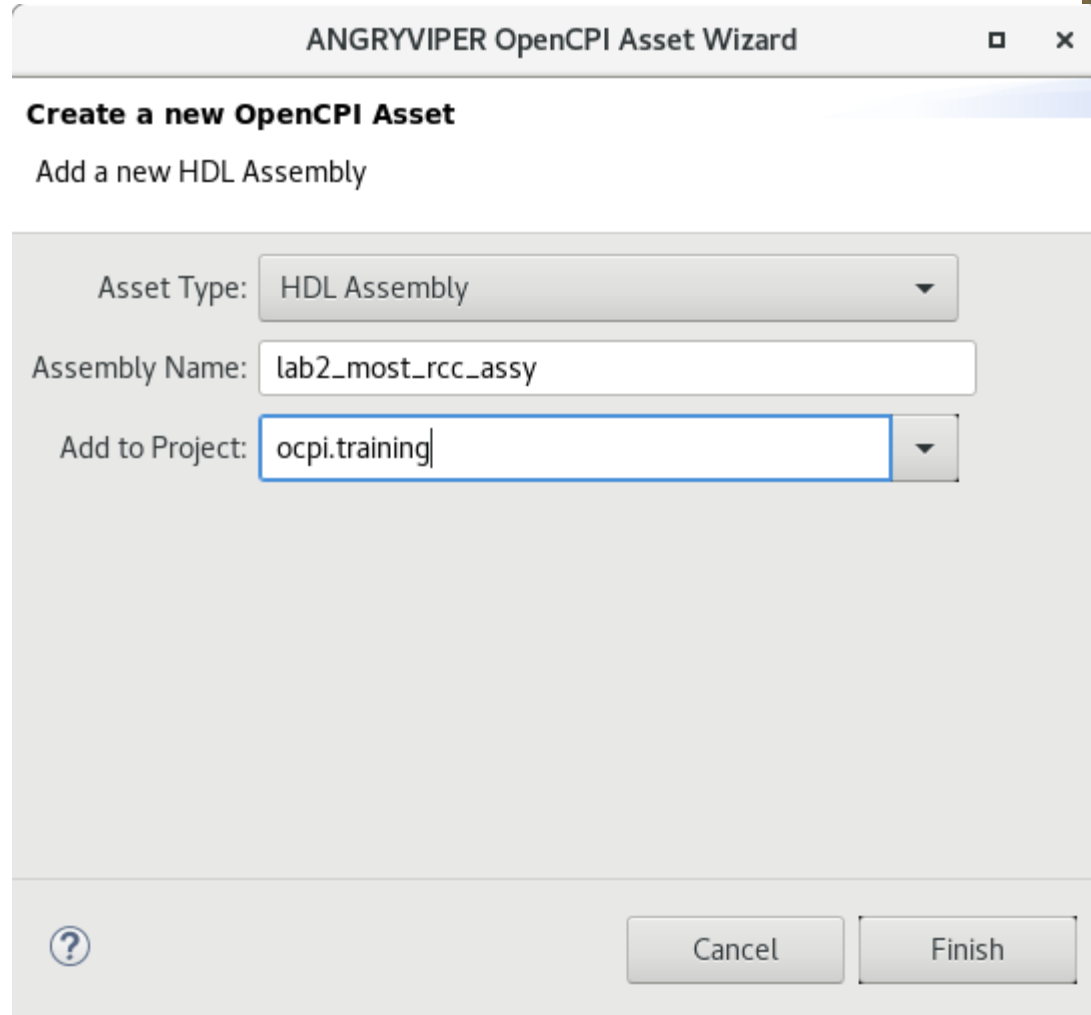
First Implementation: Most Possible RCC Workers



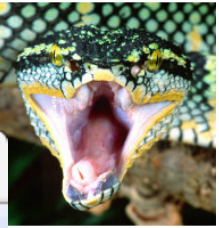
Step 3.1

Create new HDL Assembly in an existing project

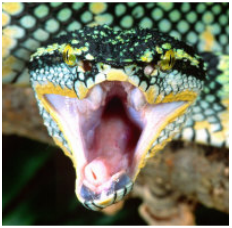
1. In OpenCPI Project View, Right Click “ocpi.training” and launch the “asset wizard”
2. Asset Type: “HDL Assembly”
3. Assembly Name: “lab2_most_rcc_assy”
4. Add to Project: “ocpi.training”



The screenshot shows the 'ANGRYVIPER OpenCPI Asset Wizard' dialog box. The title bar includes a close button. The main heading is 'Create a new OpenCPI Asset', followed by the instruction 'Add a new HDL Assembly'. The form contains three fields: 'Asset Type' is a dropdown menu set to 'HDL Assembly'; 'Assembly Name' is a text box containing 'lab2_most_rcc_assy'; and 'Add to Project' is a dropdown menu set to 'ocpi.training'. At the bottom left is a help icon (a question mark in a circle), and at the bottom right are 'Cancel' and 'Finish' buttons.



Step 3.2



1. Delete nothing worker

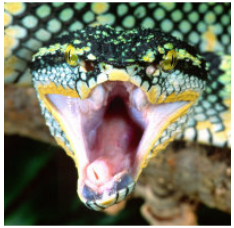
This worker is automatically placed by the framework to ensure the generated HDL assembly can be executed without editing the generated file

2. Generate HDL Assembly XML using IDE

1. Drag and drop workers into assembly (diagram on next slide).
Workers not specs. Located in .hdl directories
2. Make connections in between workers



Adding Workers to HDL Assembly



The screenshot displays the OpenCPI IDE interface. On the left, the 'Project Explorer' window shows a tree structure with folders 'ocpi.assets', 'ocpi.core', and 'training_project'. Under 'training_project', there are sub-folders 'applications' and 'components'. The 'components' folder is expanded, showing 'agc_complex.hdl' and its sub-folders 'gen', 'provided', and 'target-zynq'. Below these, the files 'agc_complex.vhd' and 'agc_complex.xml' are listed, with 'agc_complex.xml' selected. At the bottom of the tree is a 'Makefile'.

The main workspace is titled 'lab2_most_rcc_assy.xml' and contains an 'HDL Assembly' diagram. A box labeled 'agc_complex' is placed in the diagram. A large black arrow points from the 'agc_complex.xml' file in the Project Explorer to the 'agc_complex' box in the HDL Assembly diagram.

A text box with the instruction 'Drag and drop OWD XML files from <worker>.hdl directories' is positioned near the arrow. On the right side of the workspace, there is a 'Palette' panel with sections for 'Connections' (Advanced Connection, Simple Connection) and 'Objects' (Instance).

At the bottom of the workspace, there are tabs for 'HDL Assembly', 'Details', and 'Source'.

Step 3.3

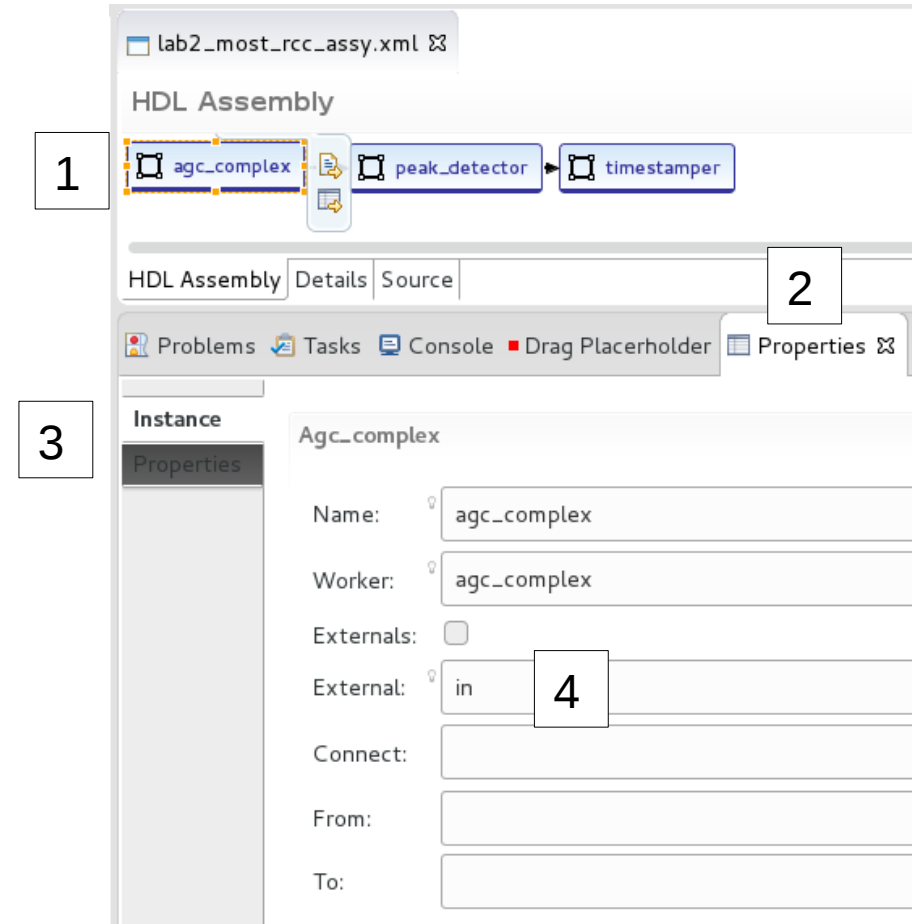
Specify External connections for input and output ports of assembly

agc_complex → “in”

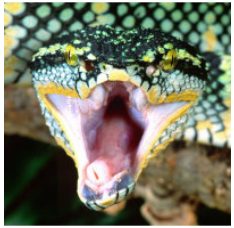
timestamp → “out”

To specify external port for worker

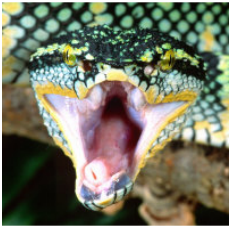
1. Click worker in IDE
2. Navigate to Properties tab
3. Click the Instance subtab
4. Enter name of port in "External" field



The screenshot shows the OpenCPI IDE interface. At the top, a file named 'lab2_most_rcc_assy.xml' is open. Below it, the 'HDL Assembly' view displays a block diagram with three components: 'agc_complex', 'peak_detector', and 'timestamp'. The 'agc_complex' block is highlighted with a dashed orange border, and a box with the number '1' points to it. Below the block diagram, there are tabs for 'HDL Assembly', 'Details', and 'Source'. The 'Details' tab is selected, and a box with the number '2' points to it. In the 'Details' tab, there are subtabs for 'Instance' and 'Properties'. The 'Instance' subtab is selected, and a box with the number '3' points to it. In the 'Instance' subtab, the 'Name' field is set to 'agc_complex', the 'Worker' field is set to 'agc_complex', and the 'Externals' checkbox is unchecked. The 'External' field is set to 'in', and a box with the number '4' points to it. The 'Connect' field is empty, and the 'From' and 'To' fields are also empty.



Step 3.4

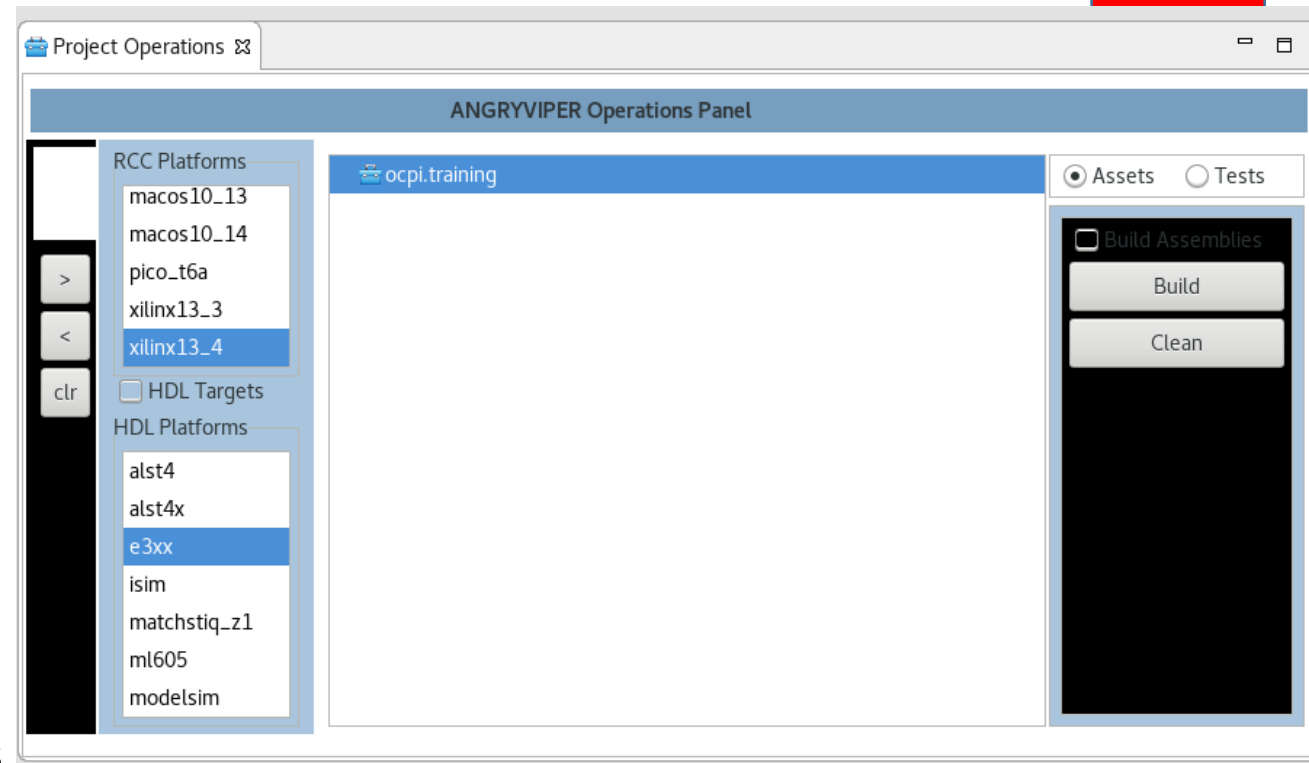


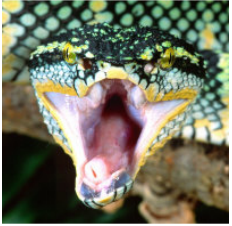
Build HDL Assembly

1. IDE: "Refresh" the OpenCPI Projects panel
2. Use the IDE to "Add" the training project to the ANGRVIPER Operations Panel
3. Highlight RCC Platforms "xilinx13_4" and HDL Platforms "e3xx"
4. Check the "Build Assemblies" option
5. Click "Build Assets"

This command will build a FPGA image for the Ettus E310 and compile all RCC workers for the ARM architecture

This step takes ~20 minutes to complete





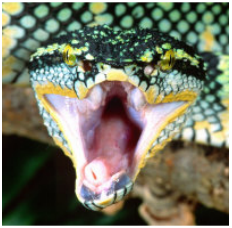
Part 4

Deploy the application on the Ettus E310

Using ocpirun to specify application preferences

- The utility program ocpirun provides a simple way to execute applications with static property values
- The arguments passed to ocpirun can specify how the application is run
- Examples
 - Restrict a worker to execute in the FPGA
 - `ocpirun -m<worker>=hdl`
 - Restrict a worker to execute on a specific platform
 - `ocpirun -p<worker>=<platform>`

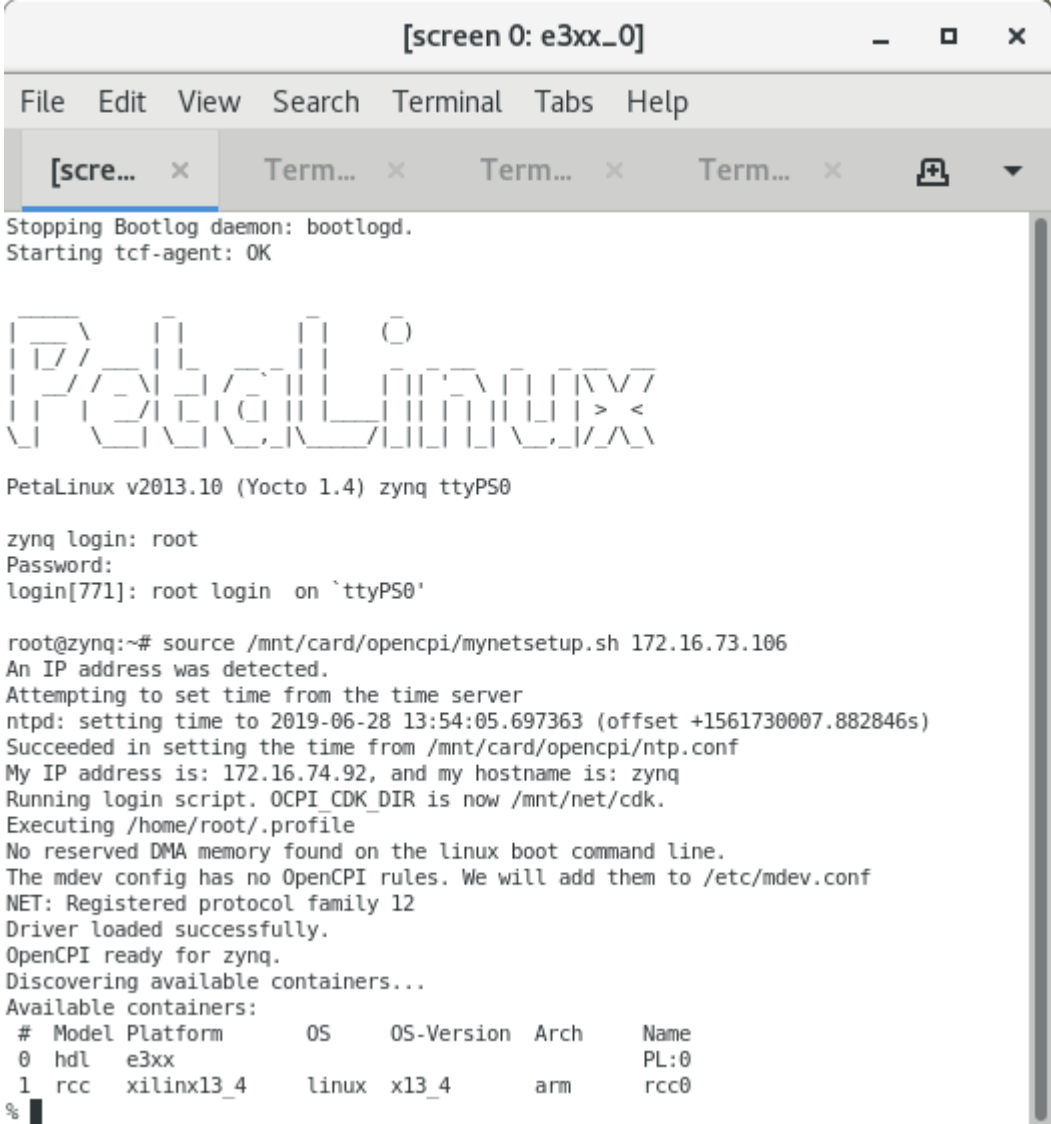
More detail on ocpirun can be found in the **OpenCPI Application Development Guide** document



Step 4.1

- 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



The image shows a terminal window titled "[screen 0: e3xx_0]" with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help) and a tab bar. The terminal output shows the following sequence of events:

```
Stopping Bootlog daemon: bootlogd.
Starting tcf-agent: OK

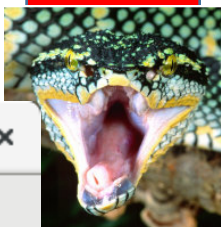
PetaLinux v2013.10 (Yocto 1.4) zynq ttyPS0

zynq login: root
Password:
login[771]: root login on `ttyPS0'

root@zynq:~# source /mnt/card/opencpi/mynetsetup.sh 172.16.73.106
An IP address was detected.
Attempting to set time from the time server
ntpd: setting time to 2019-06-28 13:54:05.697363 (offset +1561730007.882846s)
Succeeded in setting the time from /mnt/card/opencpi/ntp.conf
My IP address is: 172.16.74.92, and my hostname is: zynq
Running login script. OCPI CDK_DIR is now /mnt/net/cdk.
Executing /home/root/.profile
No reserved DMA memory found on the linux boot command line.
The mdev config has no OpenCPI rules. We will add them to /etc/mdev.conf
NET: Registered protocol family 12
Driver loaded successfully.
OpenCPI ready for zynq.
Discovering available containers...
Available containers:
```

#	Model	Platform	OS	OS-Version	Arch	Name
0	hdl	e3xx				PL:0
1	rcc	xilinxl3_4	linux	x13_4	arm	rcc0

```
%
```



Step 4.2



- Configure run-time artifact search path on target platform, i.e. `OCPI_LIBRARY_PATH=`
 - At run-time, applications must locate artifacts that satisfy its requirements, as defined in the OAS XML
 - Software worker .so files
 - HDL container .bitz files
 - The `OCPI_LIBRARY_PATH` environment variable defines the search path for locating deployable artifacts
 - Path are searched recursively, so this variable can be as very specific or as broad as needed for locating the artifacts.
 - Broader paths lead to longer search times when running an application
 - The exports directory at the top level of project contains links to artifacts contained in the project
 - Component instances were added from the component libraries contained within these projects.
 - Set `OCPI_LIBRARY_PATH` on target platform

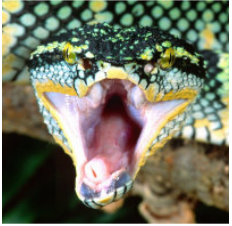
"export OCPI_LIBRARY_PATH=/mnt/ocpi_core/exports:/mnt/training_project/artifacts"

Step 4.3

- Run application on target platform using ocpirun
- To run application on target platform:
 1. Navigate to OAS XML:

```
"cd /mnt/training_project/applications"
```
 2. Pass OAS XML to ocpirun:

```
"ocpirun -t 1 -d -v -mcomplex_mixer=rcc lab2_app.xml"
```



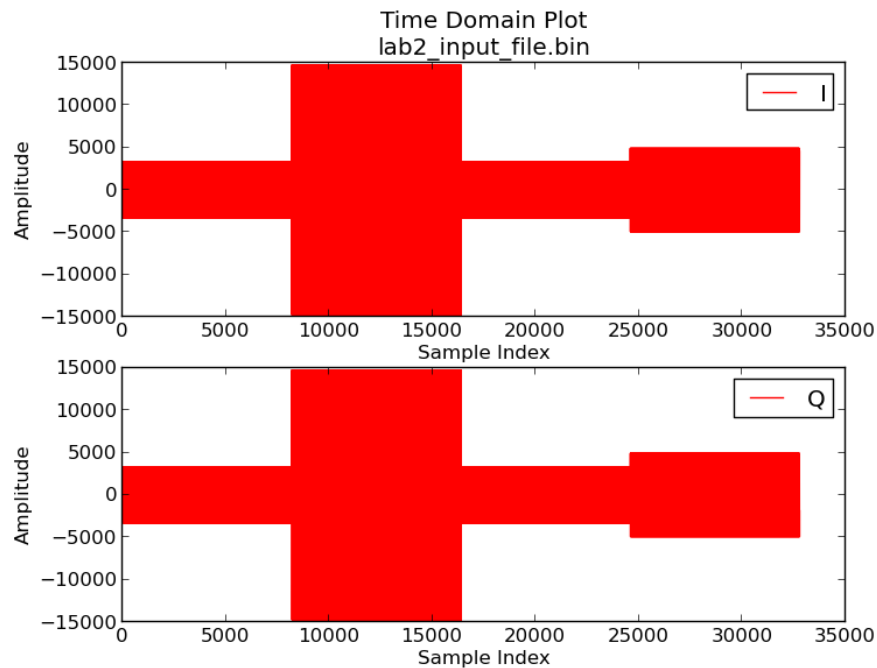
Expected result – Input Data



On Host:

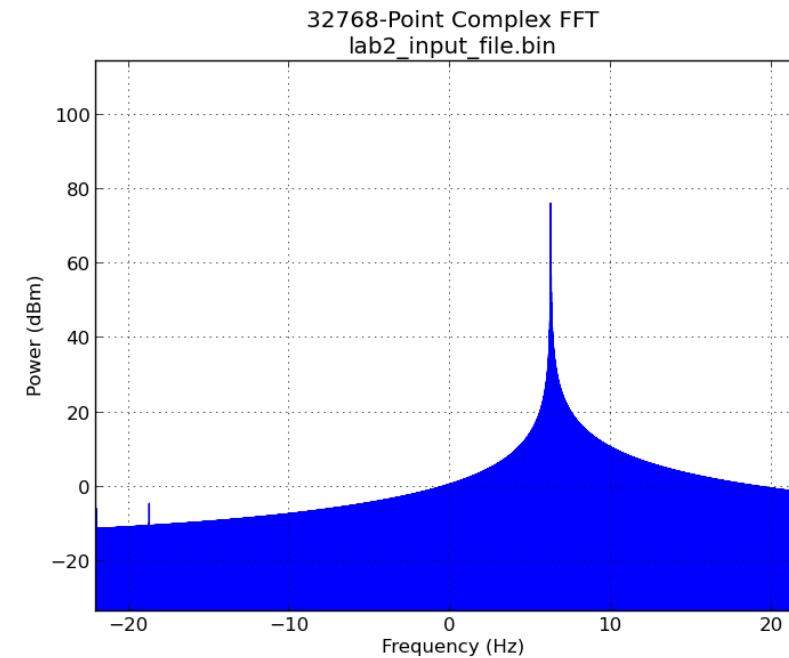
```
"cd ~/training_project/applications"
```

```
"python ../scripts/plotAndFft.py idata/lab2_input_file.bin complex 32768 100"
```



Time Domain

3 distinct input levels – from AGC unit test lab



Frequency Domain

Tone at $F_s/16$ – from AGC unit test lab

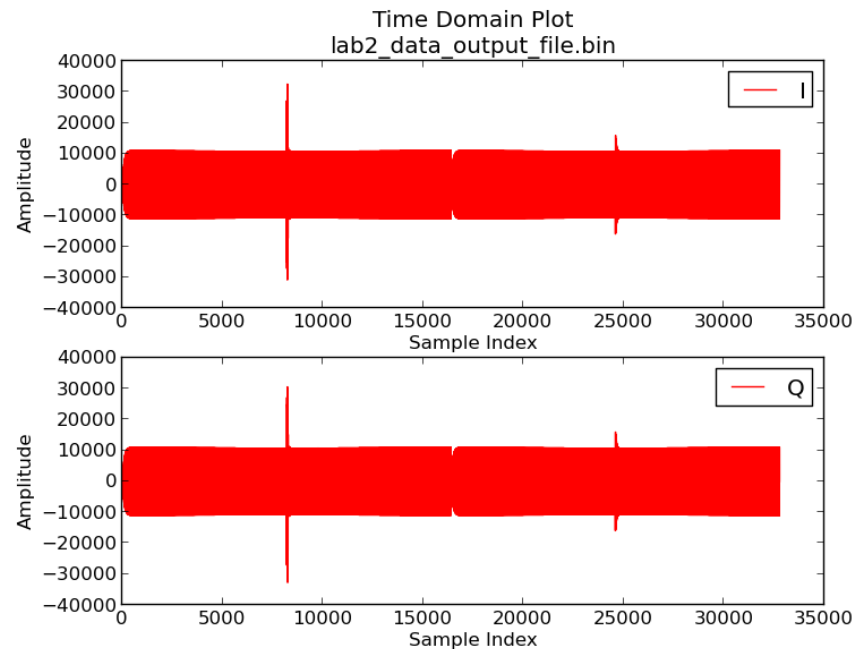
Expected result – Output Data



On Host:

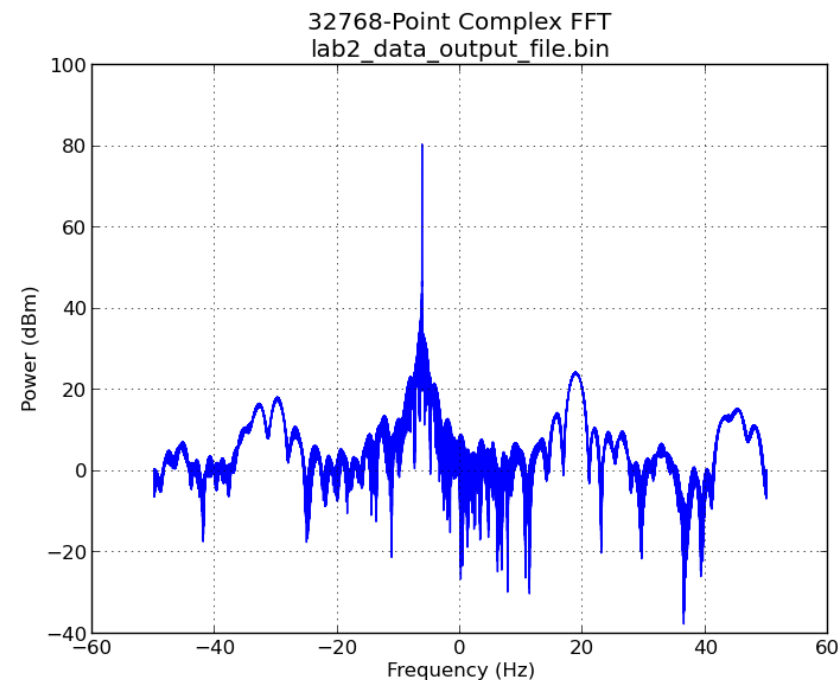
```
"cd ~/training_project/applications"
```

```
"python ../scripts/plotAndFft.py odata/lab2_data_output_file.bin complex 32768 100"
```



Time Domain

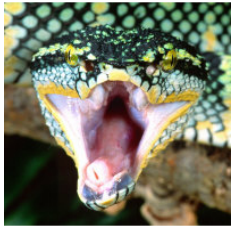
Single input level – minus settling from AGC



Frequency Domain

Tone at $F_s/16$ minus 12.5 – Mixer shifted 12.5 Hz

Expected Result – Output Timestamps



On Host:

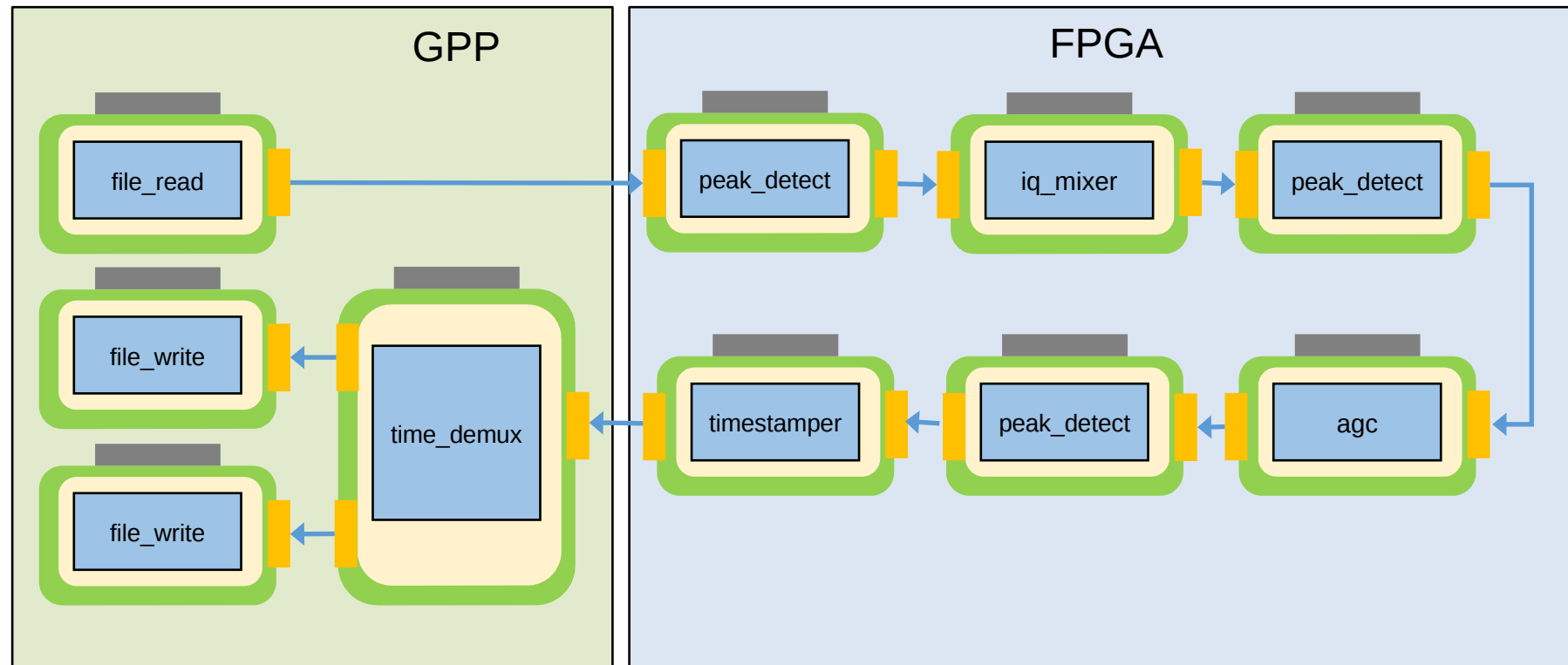
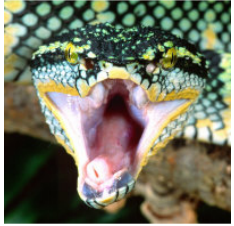
```
"cd ~/training_project/applications"
```

```
"python scripts/print_timestamps.py odata/lab2_time_output_file.bin"
```

```
*****
*** Python: Prints Timestamps ***
Pass: File is not all zeros
Timestamp is: 0.3454791 ( Seconds: 0x0 Fraction: 0x5871516a )
Timestamp is: 0.3553820 ( Seconds: 0x0 Fraction: 0x5afa50f7 ) Delta: 0.0099029
Timestamp is: 0.3652552 ( Seconds: 0x0 Fraction: 0x5d815c8d ) Delta: 0.0098731
Timestamp is: 0.3751009 ( Seconds: 0x0 Fraction: 0x60069c47 ) Delta: 0.0098457
Timestamp is: 0.3849273 ( Seconds: 0x0 Fraction: 0x628a990b ) Delta: 0.0098265
Timestamp is: 0.3947605 ( Seconds: 0x0 Fraction: 0x650f06e3 ) Delta: 0.0098332
Timestamp is: 0.4045932 ( Seconds: 0x0 Fraction: 0x67936b00 ) Delta: 0.0098326
Timestamp is: 0.4144343 ( Seconds: 0x0 Fraction: 0x6a185db8 ) Delta: 0.0098411
Timestamp is: 0.4242827 ( Seconds: 0x0 Fraction: 0x6c9dc9e7 ) Delta: 0.0098484
Timestamp is: 0.4341280 ( Seconds: 0x0 Fraction: 0x6f230295 ) Delta: 0.0098453
Timestamp is: 0.4439507 ( Seconds: 0x0 Fraction: 0x71a6c0f0 ) Delta: 0.0098227
Timestamp is: 0.4537747 ( Seconds: 0x0 Fraction: 0x742a93c2 ) Delta: 0.0098240
Timestamp is: 0.4636022 ( Seconds: 0x0 Fraction: 0x76aea24f ) Delta: 0.0098275
Timestamp is: 0.4734202 ( Seconds: 0x0 Fraction: 0x79321077 ) Delta: 0.0098180
Timestamp is: 0.4832367 ( Seconds: 0x0 Fraction: 0x7bb566f8 ) Delta: 0.0098166
Timestamp is: 0.4930727 ( Seconds: 0x0 Fraction: 0x7e3a039f ) Delta: 0.0098360
*** End ***
*****
```

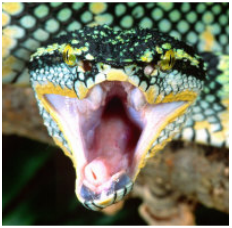
Timestamps are incrementing and *roughly* uniformly spaced
(but affected by software bottlenecks because stamped at **end** of processing)

Second Implementation: Most Possible HDL Workers

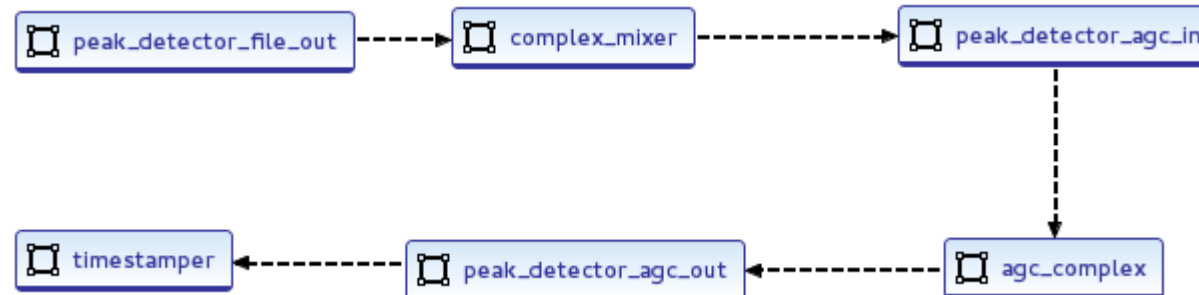


Repeat Parts 3 and 4 for this implementation

Most Possible HDL Workers: Hints



- Part 3
 - Sample assembly name: "lab2_most_hdl_assy"
 - Don't forget to delete "nothing" worker
 - Uniquely name instances of peak detect
 - Specify External connections for input and output ports of assembly
 - Make sure to save before building!!!
- Part 4
 - Command to run the application:
"ocpirun -t 1 -d -v -mcomplex_mixer=hdl lab2_app.xml"



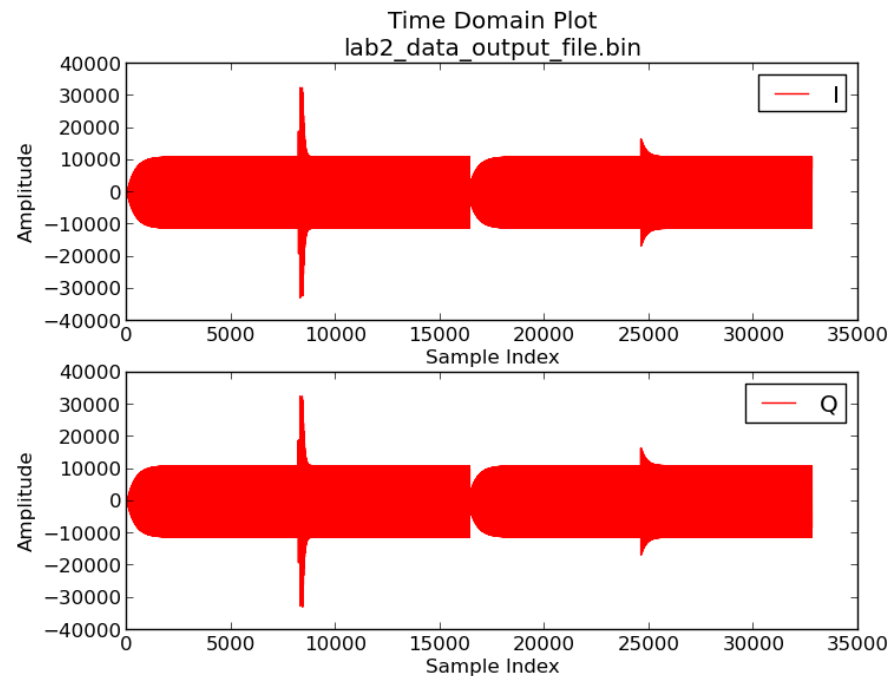
Expected result – Output Data



On Host:

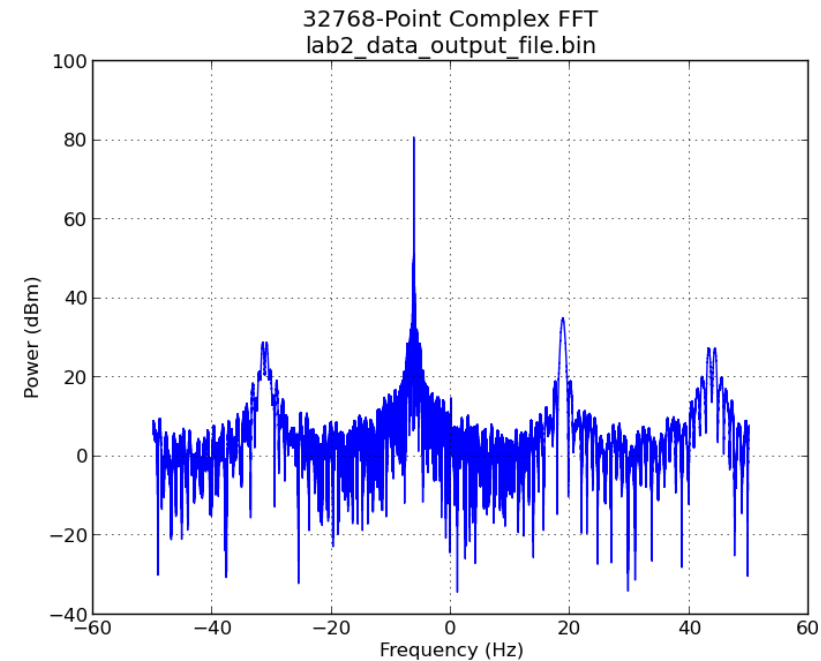
```
"cd ~/training_project/applications"
```

```
"python ../scripts/plotAndFft.py odata/lab2_data_output_file.bin complex 32768 100"
```



Time Domain

Single input level – minus settling from AGC



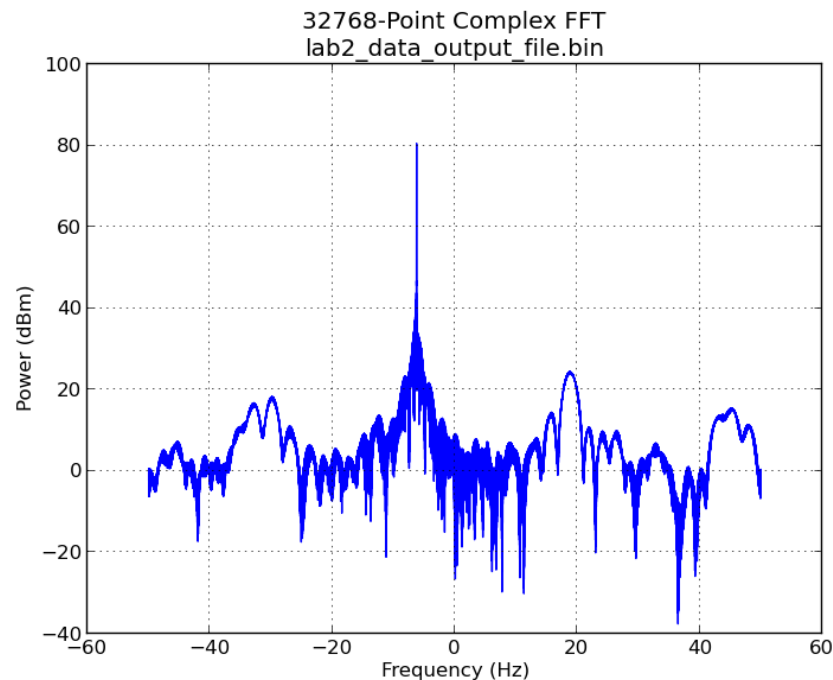
Frequency Domain

Tone at $F_s/16$ minus 12.5 – Mixer shifted 12.5 Hz

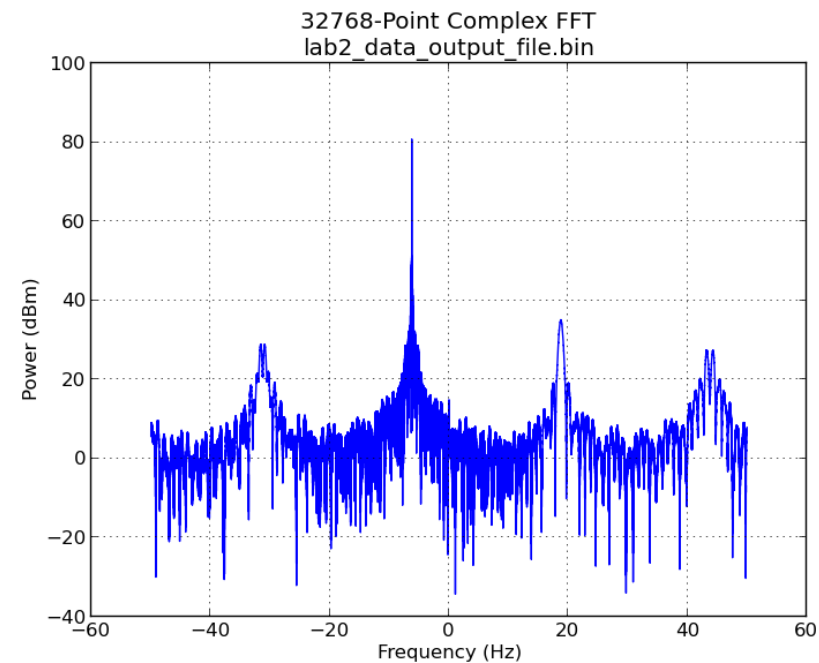
Expected result – Delta Explained



The differences observed in the plots of the output data between the mostly RCC vs mostly HDL applications, is primarily a result of the precision of the math computations performed by the different Authoring Model implementations of the Complex Mixer. Specifically, the RCC worker utilizes the 3rd party LiquidDSP library which performs floating-point computations (truncated to int16), where as the HDL worker performs all fixed-point (16 bit) computations. While the difference is enough to be observed in the plots below, the system designer must determine if the performance is sufficient for a given target application.



Mostly RCC



Mostly HDL

(Intentionally Blank)



Remove training project



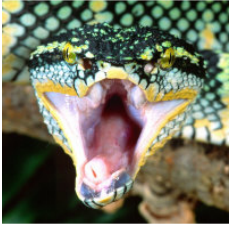
The remainder of the labs will be recreating the project and application that was just built one component at a time

1. Remove the training project

1. Right-click project in OpenCPI Projects pane
2. "delete asset"
3. Right-click project in Project Explorer pane
4. "Delete" and check box to remove from disk

2. Reboot target platform before lab 3

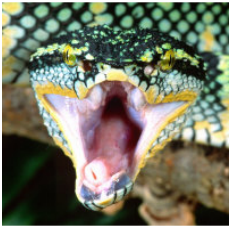




Lab 3 Preparation

Objectives

- Create an OpenCPI Project using the IDE
- Modify the Project's Namespace
- Update Project Registry to reflect changes
- Copy "provided" scripts into the new Project
- Create Component Library in preparation for the next lab



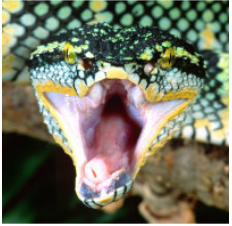
Project Creation using ocpidev



- Create "training_project" project in /home/training/
 1. `$ cd ~`
 2. `$ ocpidev create project training_project`
 3. Edit ~/training_project/Project.mk
 1. "PackagePrefix=ocpi"
 2. "PackageName=training" (defaults to name of directory)
 3. "ProjectDependencies=ocpi.assets"
 4. Save and Close

Add project to registry using ocpidev

1. `$ cd ~/training_project/`
2. `$ ocpidev register project`



Check Project Registry



- Examine the current state of the Project Registry
 - `$ ocpi dev show registry`
 - Are the paths correct?
- If not:
 - Un-register a "specific" project from registry
 - `$ ocpi dev unregister project ocpi.training`
 - Re-register (from within Project's directory)
 - `$ ocpi dev register project`

```
$ ocpi dev show registry
Project registry is located at: /opt/opencpi/project-registry
```

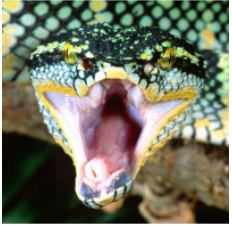
Project	Package-ID	Path to Project	Valid/Exists
ocpi.assets		/home/training/ocpi_projects/assets	True
ocpi.assets_ts		/home/training/ocpi_projects/assets_ts	True
ocpi.core		/home/training/ocpi_projects/core	True
ocpi.bsp.bsp_e310		/home/training/ocpi_projects/bsp_e310	True
ocpi.training		/home/training/training_project	True

Edit “.project” file

1. \$ cd ~/training_project

2. Edit .project file

1. Replace “local” with “ocpi”
2. Save and Close



Import training_project into AV IDE

1. Launch IDE

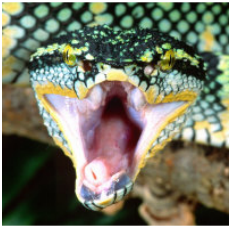
2. Import training_project:

1. To import project into eclipse:

1. File → Import...

1. "Existing Projects into Workspace"

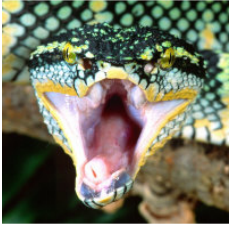
2. Refresh "OpenCPI Projects"



Copy "provided" scripts into Project

- Copy the provided "scripts" directory into the top-level of the training Project

```
$ cp -rf /home/training/provided/scripts/ \
/home/training/training_project/
```



Preparation for the next lab

- Create a Component Library
 - Right-Click “training_project” → Asset Wizard → Library
 - (Verify “ocpi.training” is target)
 - Library Name: “components”

