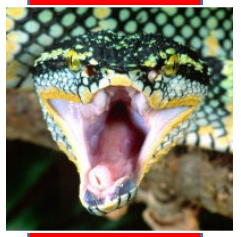
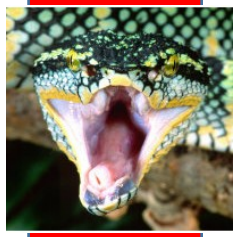
Lab 2
OpenCPI Application Development & Integration

Objectives



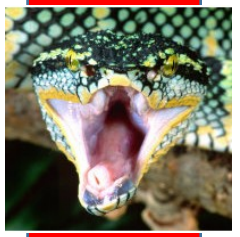
1. Create application using pre-existing workers from an imported project
2. Identify which components of the application can be deployed on the FPGA
3. Create and build two different HDL Assemblies which can be used by the application
4. Deploy the application with ocpirun using both of the HDL Assemblies

Overview

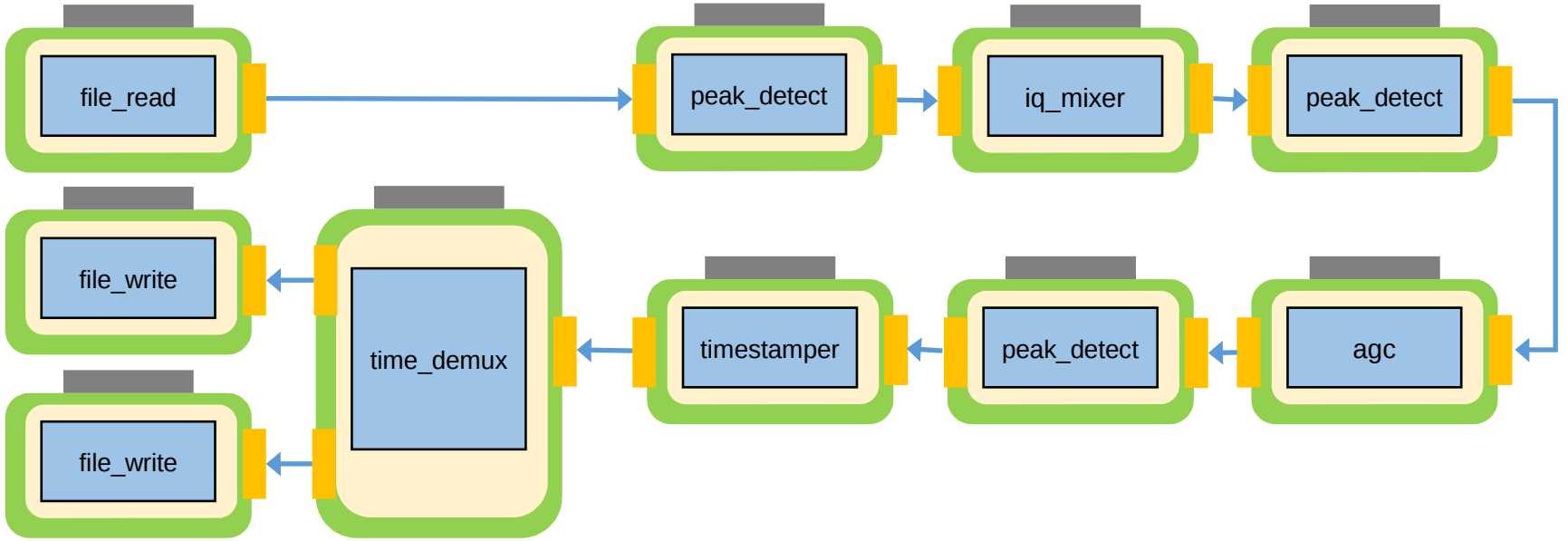


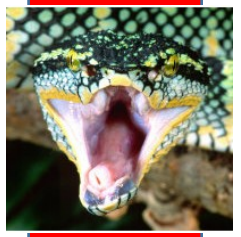
- A common use case for OpenCPI is the reuse of components from multiple libraries to construct applications, that are deployed onto heterogeneous systems.
- Once the required components for an application have been determined, the subset of components which will execute in the FPGA must be identified, 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 the OpenCPI 1.3 release
- 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

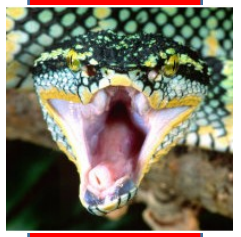




Part 1

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/ ~

- Register training_project:
  - In a terminal window:

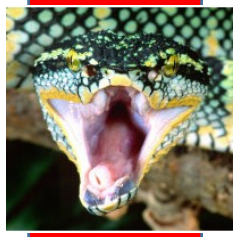
    cd ~/training_project/

    ocpidev register project

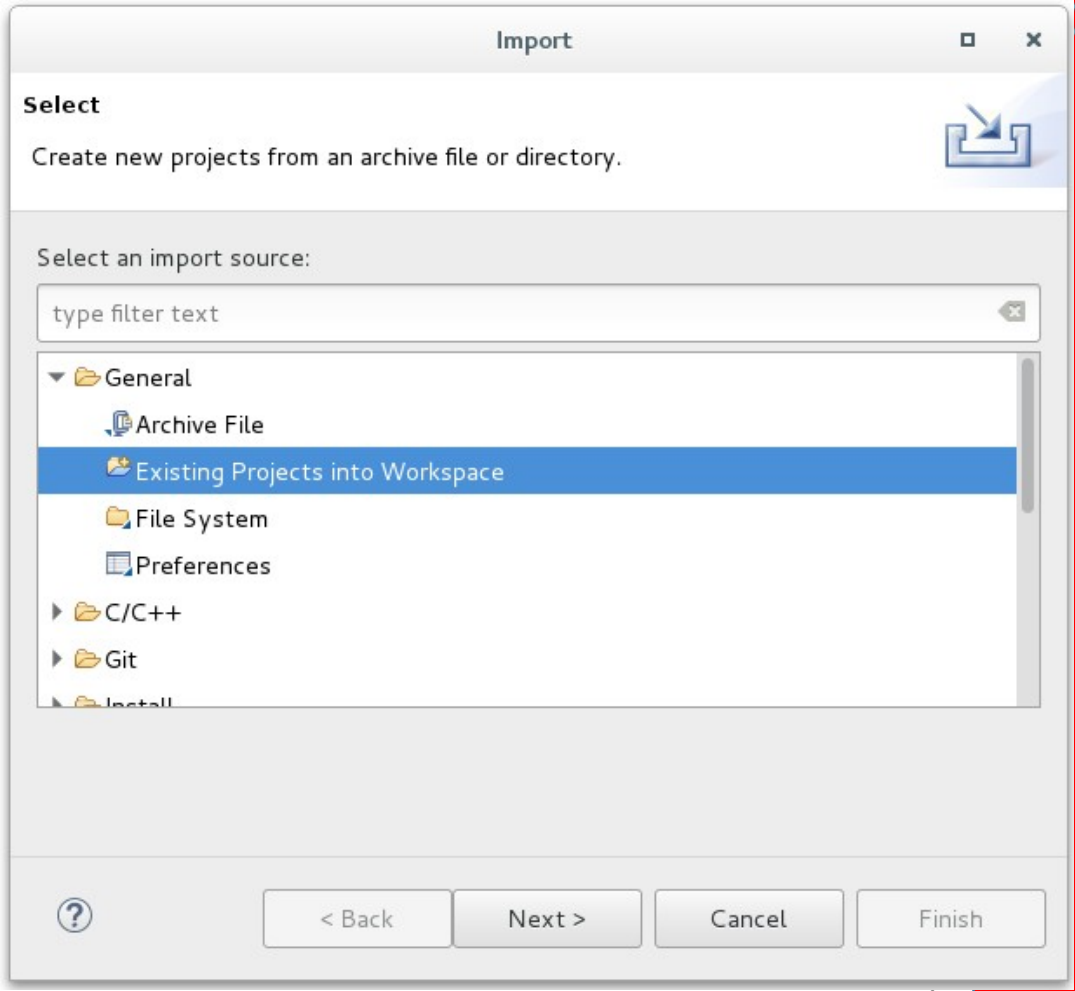
    ocpidev show projects --table

```
Project registry is located at: /opt/opencpi/project-registry
-------------------------------------------------------------------------------------
| Project Package-ID/Link | Path to Project                  | Valid/Exists | Registered |
| ----------------------- | -------------                    | ------------ | ---------- |
| ocpi.core               | /home/training/ocpi_core         | True         | True       |
| ocpi.training           | /home/training/training_project  | True         | True       |
| ocpi.cdk                | /opt/opencpi/cdk                 | True         | True       |
| ocpi.assets             | /home/training/ocpi_assets       | True         | True       |
-------------------------------------------------------------------------------------
```

Step 1.2



- Launch IDE
  - Ensure IDE is launched after project registry is modified (last step)
- Import training_project:
  - To import project into eclipse:
    - File -> Import…
      - “Existing Projects into Workspace”



Step 1.3

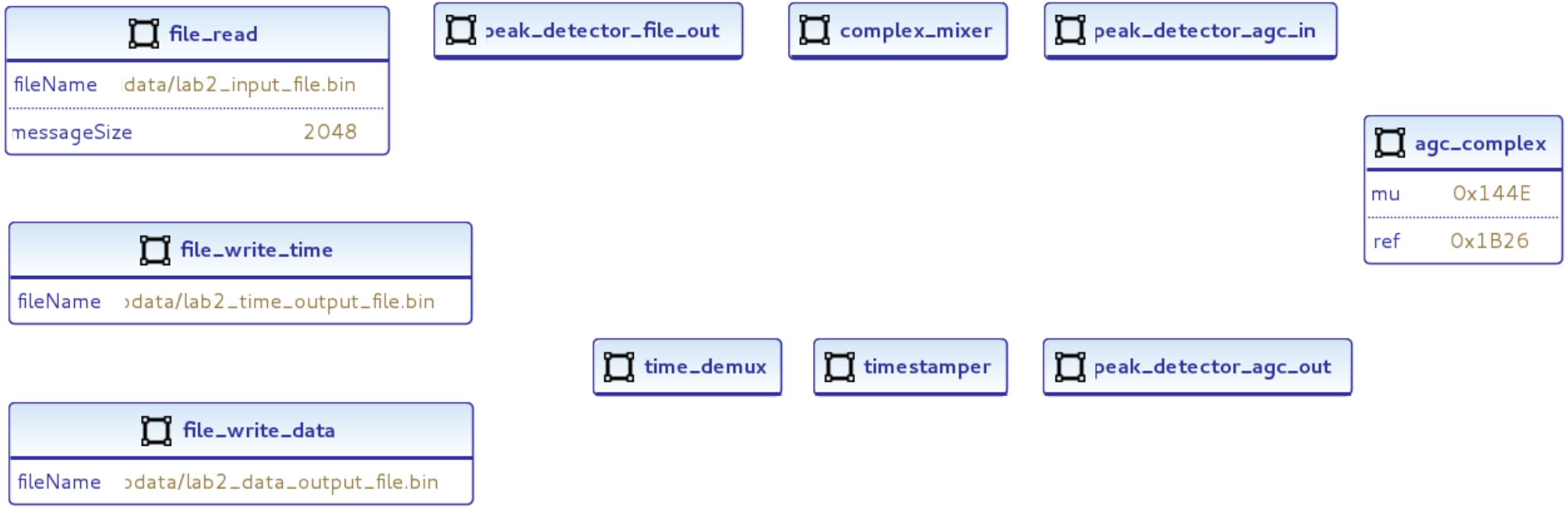
- Create new Application using IDE called lab2_app
  - XML only
- Add the components in this table to the application
  - Do so in the order they are listed here
- Hints
  - Remember to name components with multiple instances uniquely

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

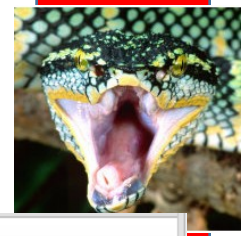
Step 1.4

- Specify property values
  - When not specified in the XML, the properties of a component assume a default value
  - **No ValueFiles in this lab**

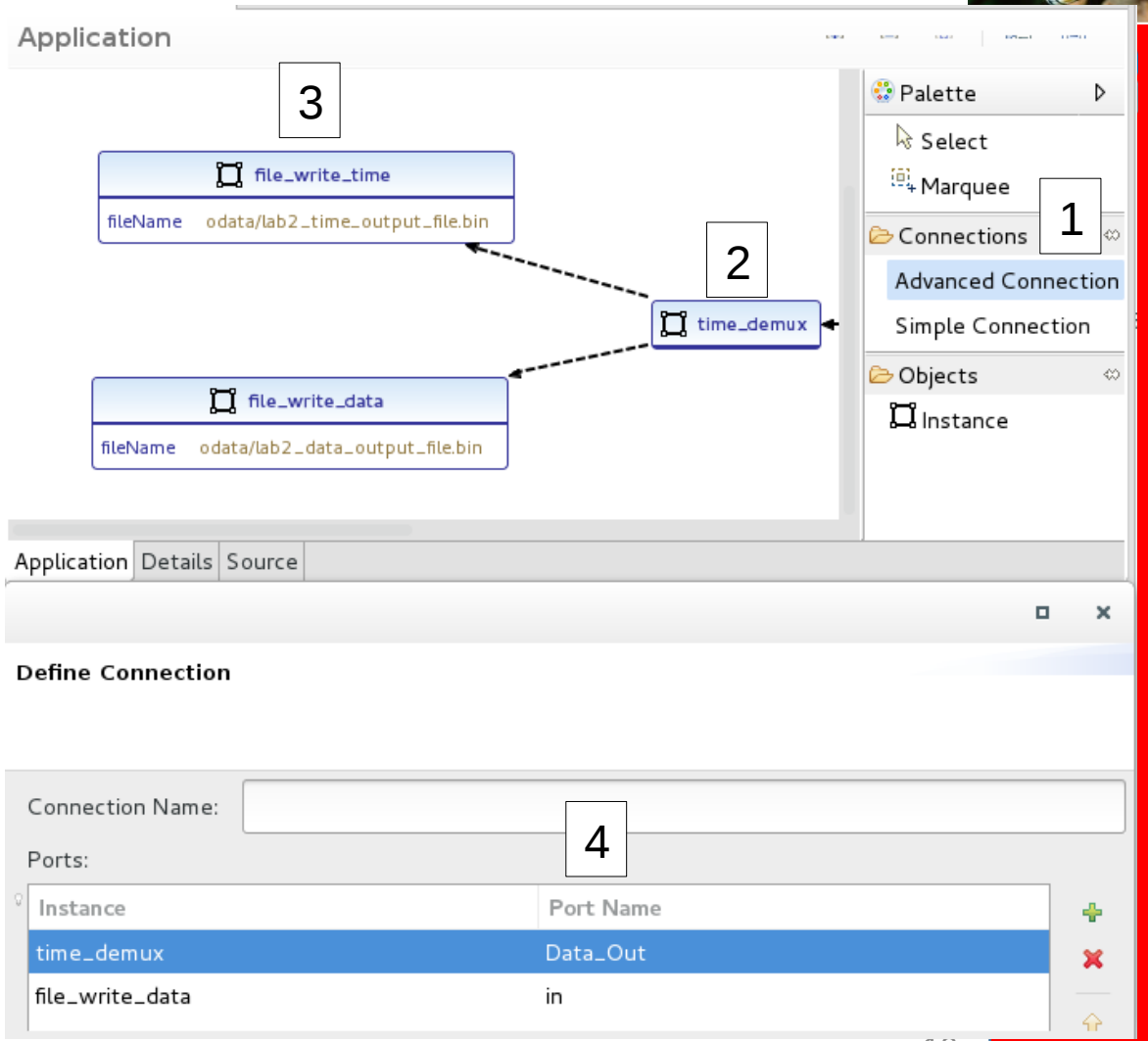
Property Values Required		
Component	**Property Name**	**Value**
file_read	fileName	idata/lab2_input_file.bin
file_read	messageSize	2048
agc_complex	mu	0x144E
agc_complex	ref	0x1B26
file_write_time	fileName	odata/lab2_time_output_file.bin
file_write_data	fileName	odata/lab2_data_output_file.bin



Step 1.5



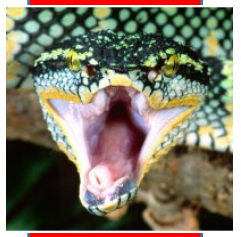
- Make connections
  1. Click “Advanced Connection” on Palette Menu
  2. Click originating instance
  3. Click destination instance
  4. Populate “Port Name” fields for connections
     - **time_demux uses non-default Port Names**
       - timestamper:out->time_demux: Mux_In
       - time_demux: Data_Out->file_write_data:in
       - time_demux: Time_Out->file_write_time:in
     - All other components use default Port Names



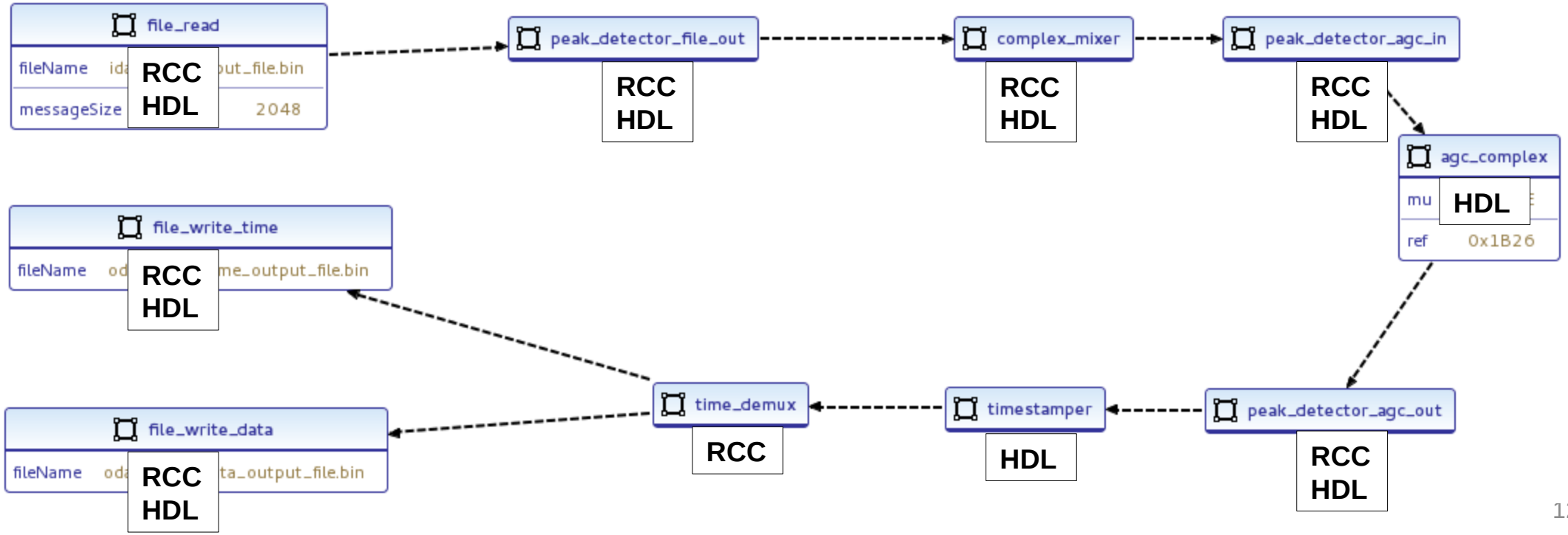
Part 2

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

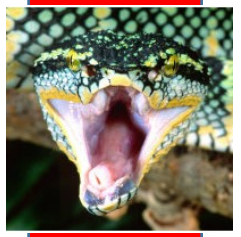
Which components have HDL implementations?



- 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

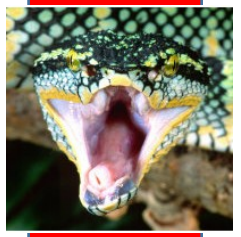


Which components make sense to use on FPGA?



- 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 two implementations
  - Most possible HDL workers
  - Most possible RCC workers
- Application is the same for both!

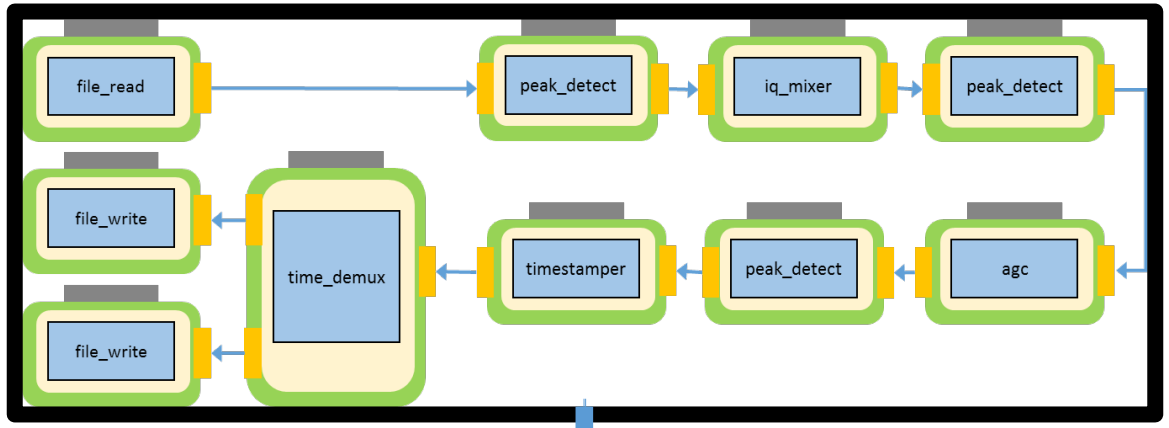
Most possible HDL workers	Most possible RCC workers
file_read.rcc	file_read.rcc
peak_detector.hdl	peak_detector.rcc
complex_mixer.hdl	complex_mixer.rcc
peak_detector.hdl	peak_detector.rcc
agc_complex.hdl	agc_complex.hdl
peak_detector.hdl	peak_detector.hdl
timestamper.hdl	timestamper.hdl
time_demux.rcc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



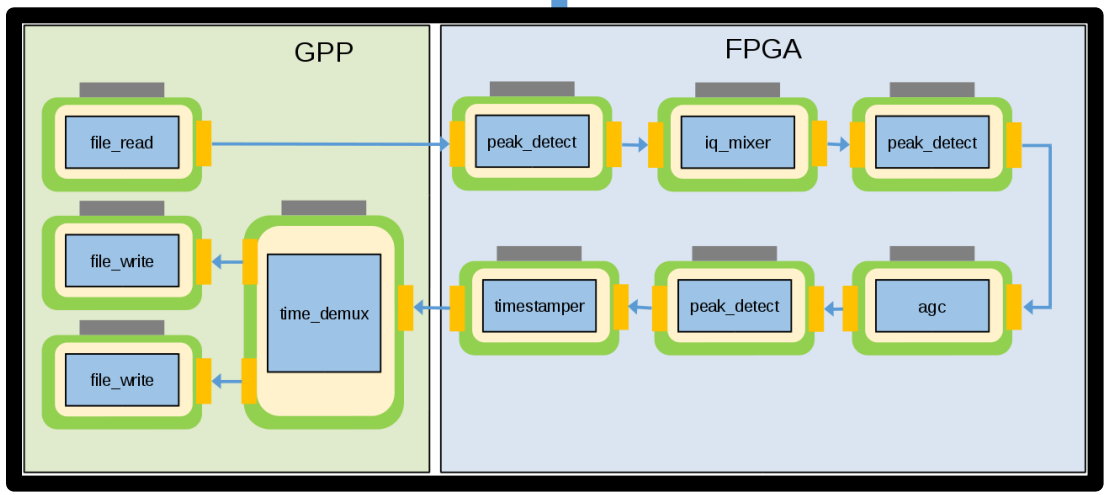
Part 3

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

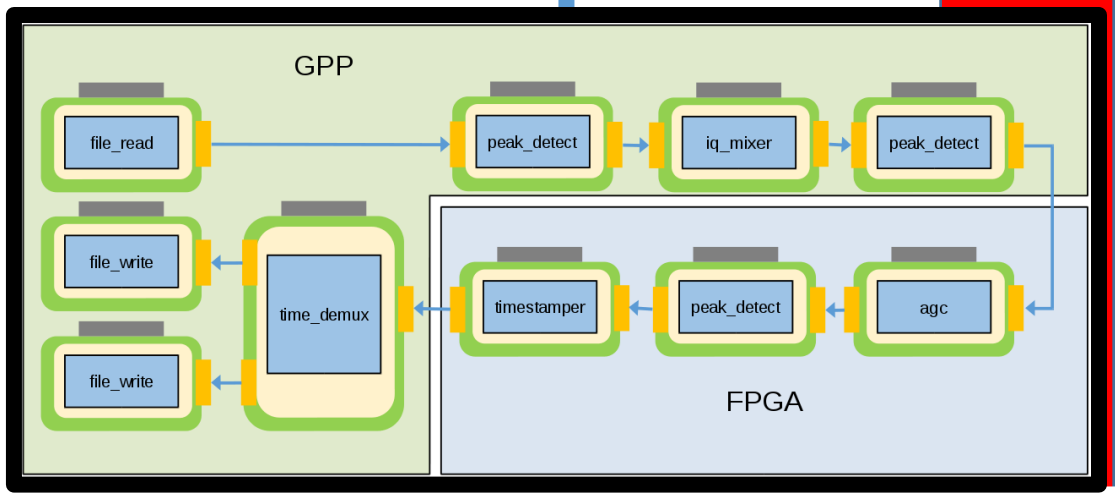
One Application



Multiple Implementations



Most possible HDL workers

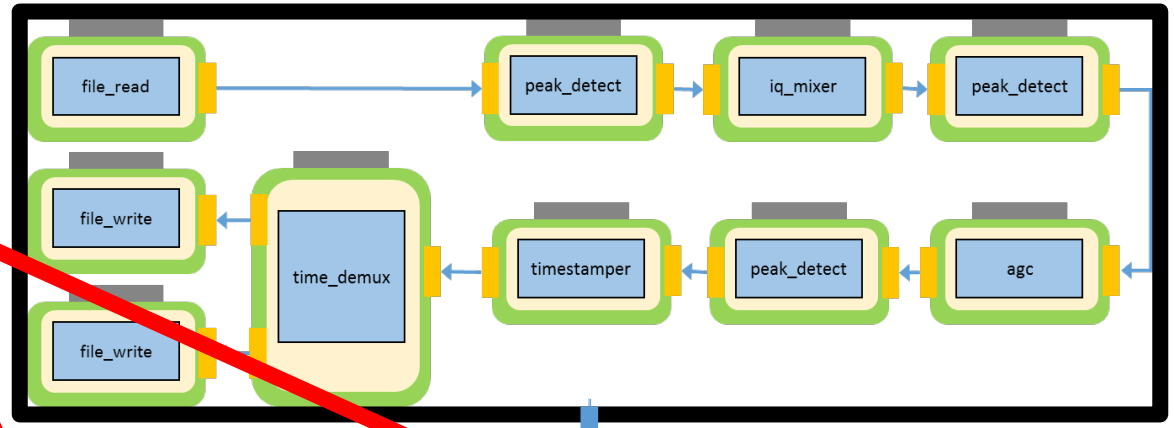


Most possible RCC workers

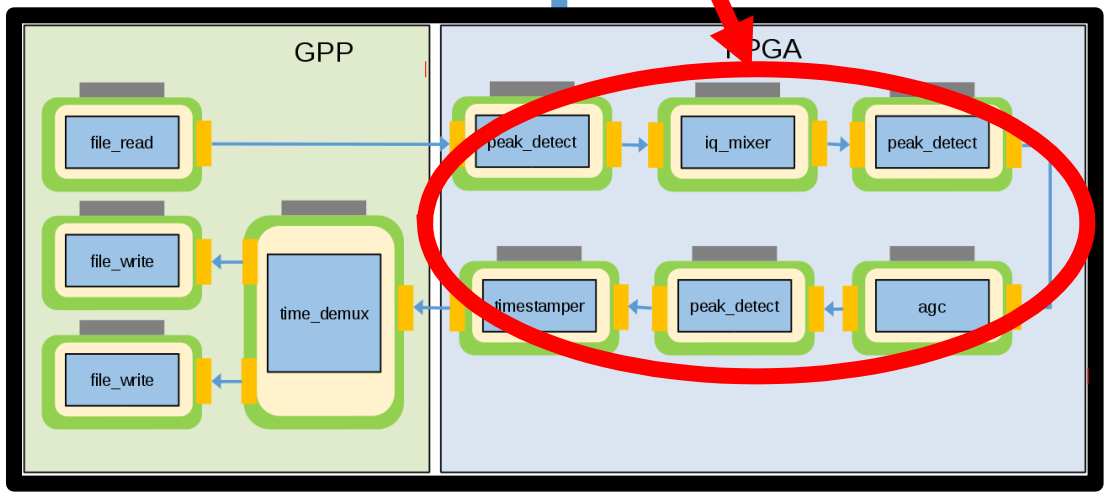
OpenCPI

One Application

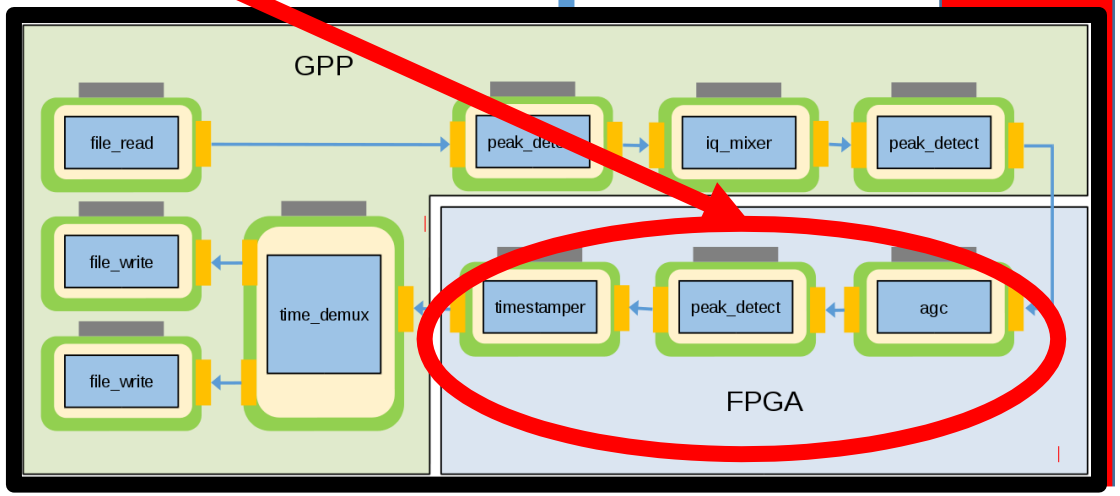
Need to specify and build this piece



Multiple Implementations



Most possible HDL workers

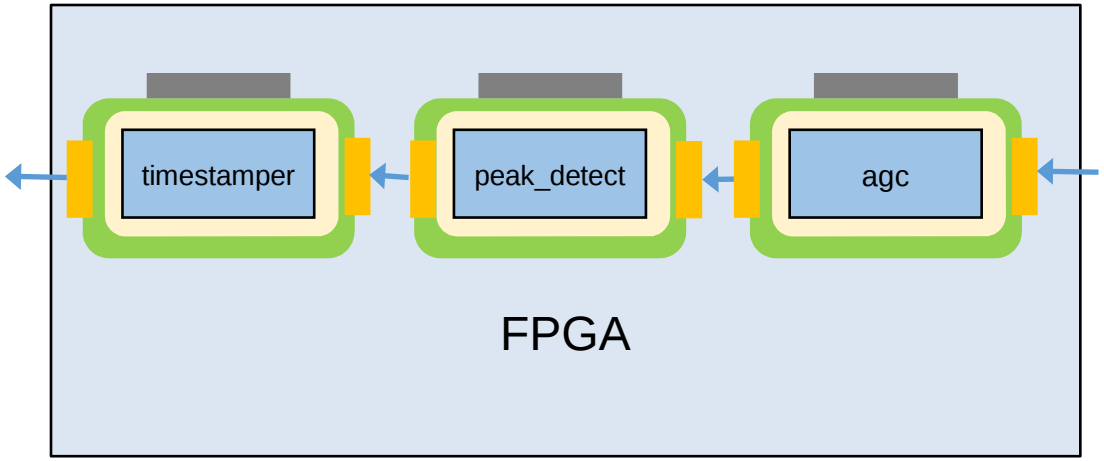


Most possible RCC workers

OpenCPI

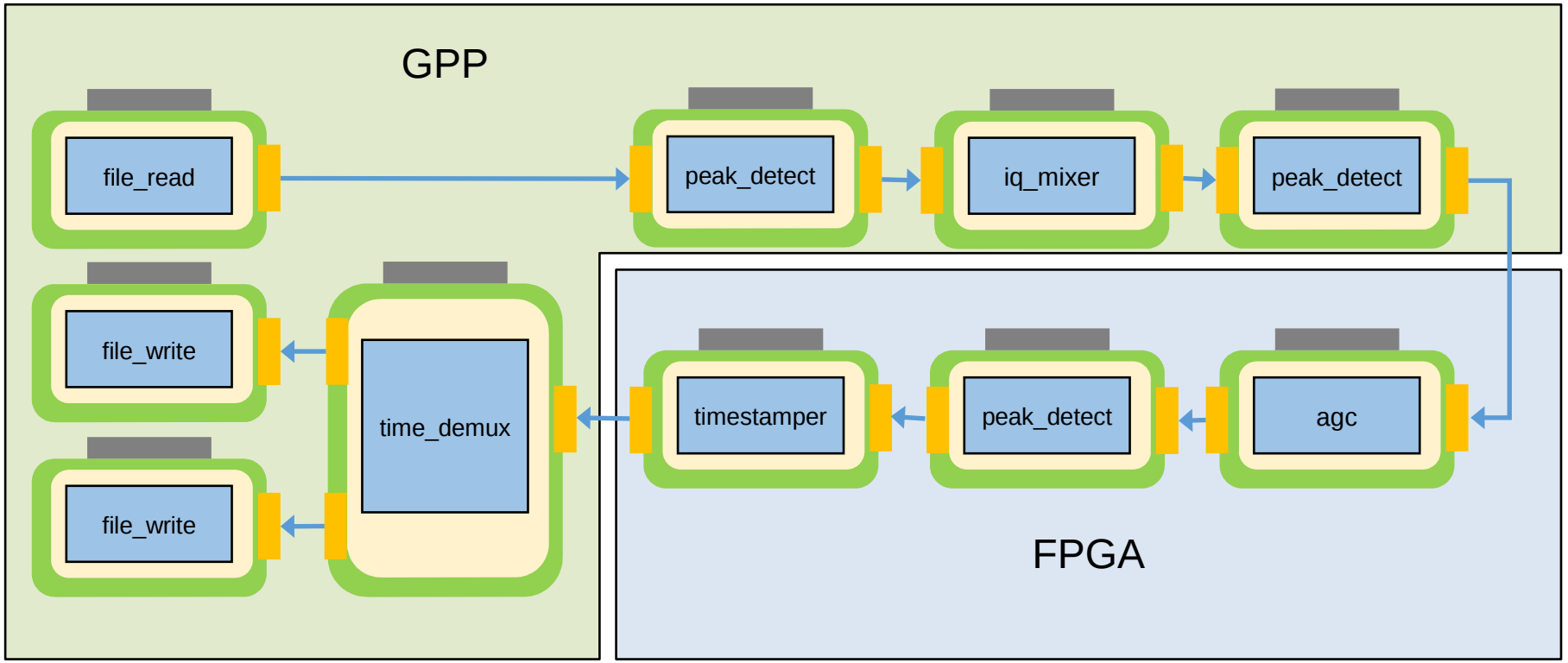
What is specified in an HDL Assembly?

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



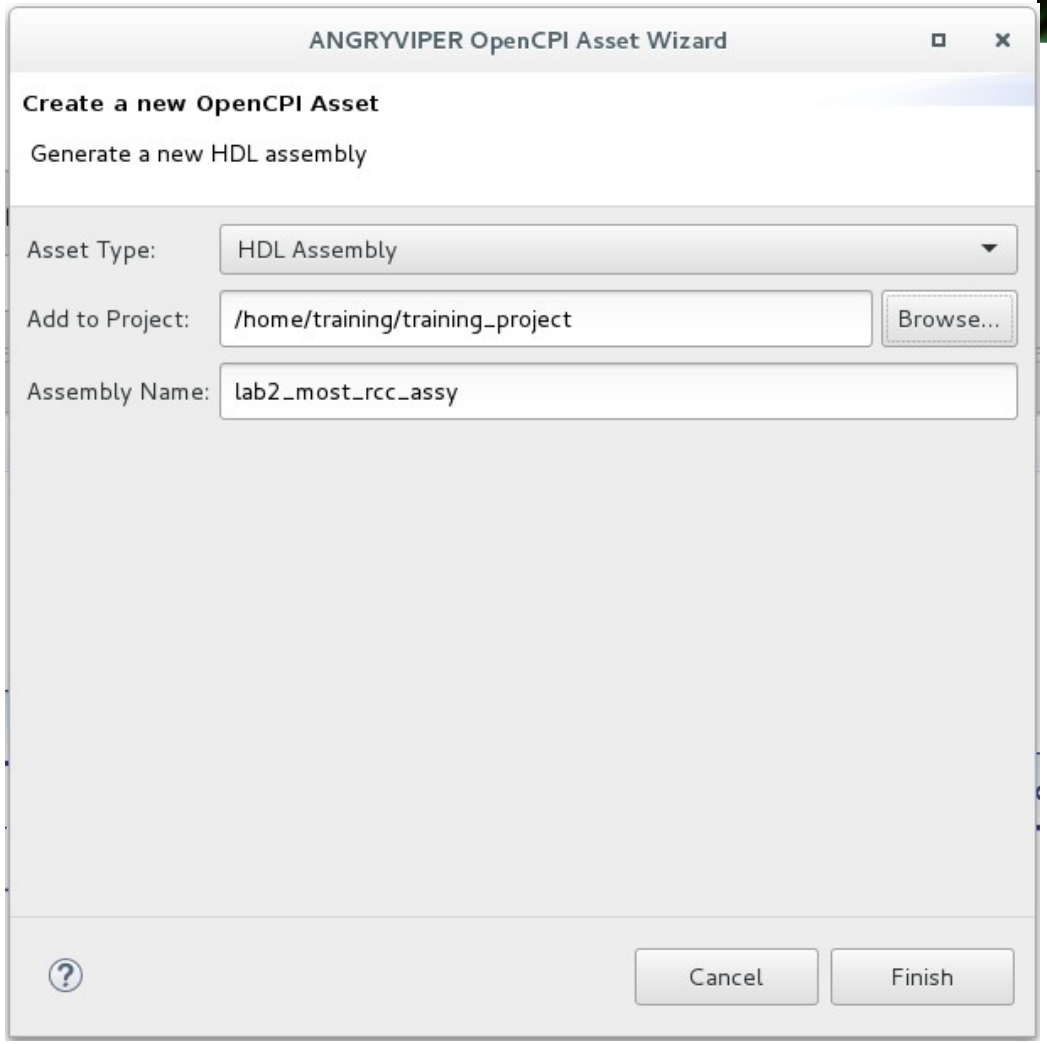

```
<HdlAssembly>
    <Instance worker="agc_complex" name="agc_complex" external="in"></Instance>
    <Instance worker="timestamper" name="timestamper" external="out"></Instance>
    <Instance worker="peak_detector" name="peak_detector"></Instance>
    <Connection>
        <Port instance="agc_complex"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

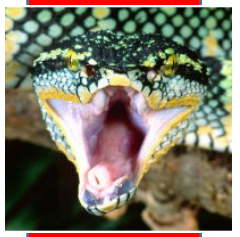


Step 3.1

- Create new HDL Assembly in an existing project
  - In Project Explorer, right click training_project:
    - New -> OpenCPI Asset Wizard
  - Asset Type: HDL Assembly
  - Add to Project: training_project
  - Assembly Name: lab2_most_rcc_assy

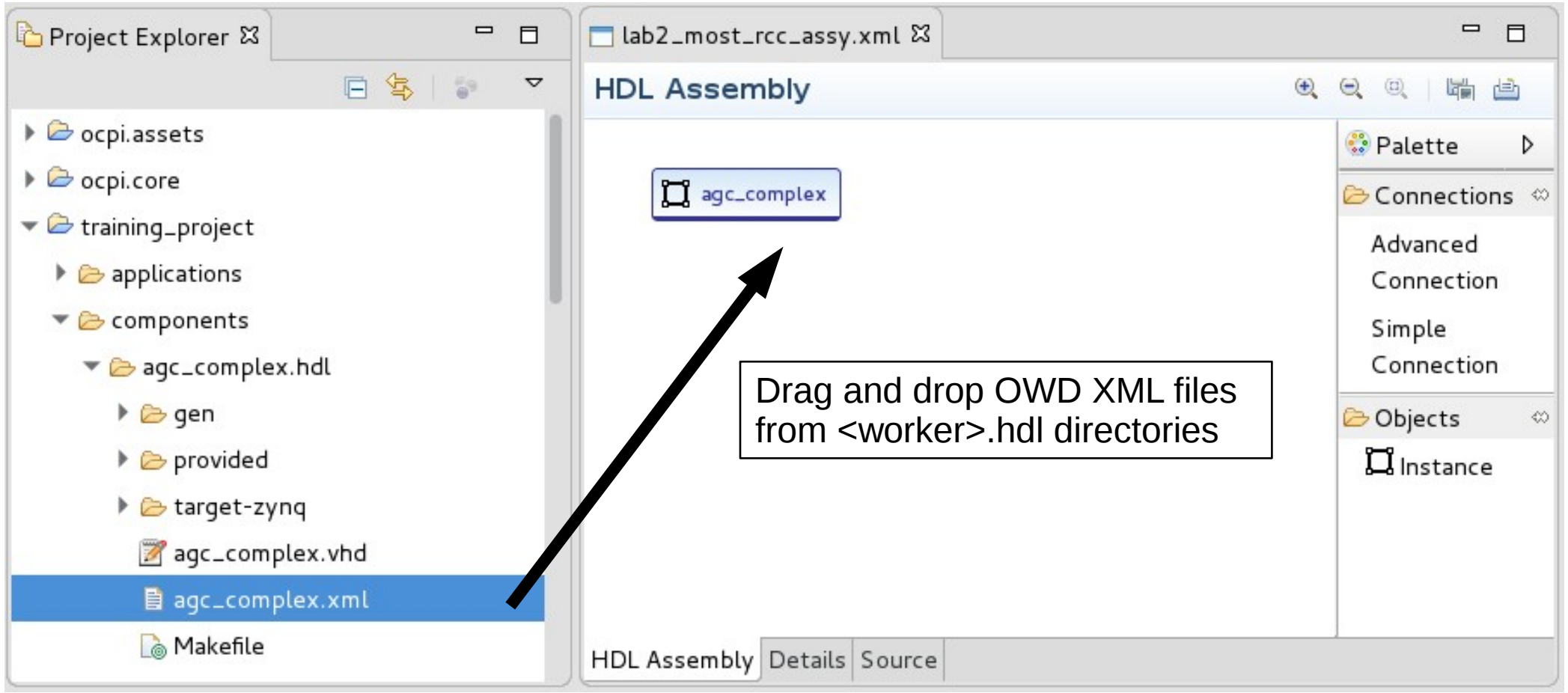


Step 3.2



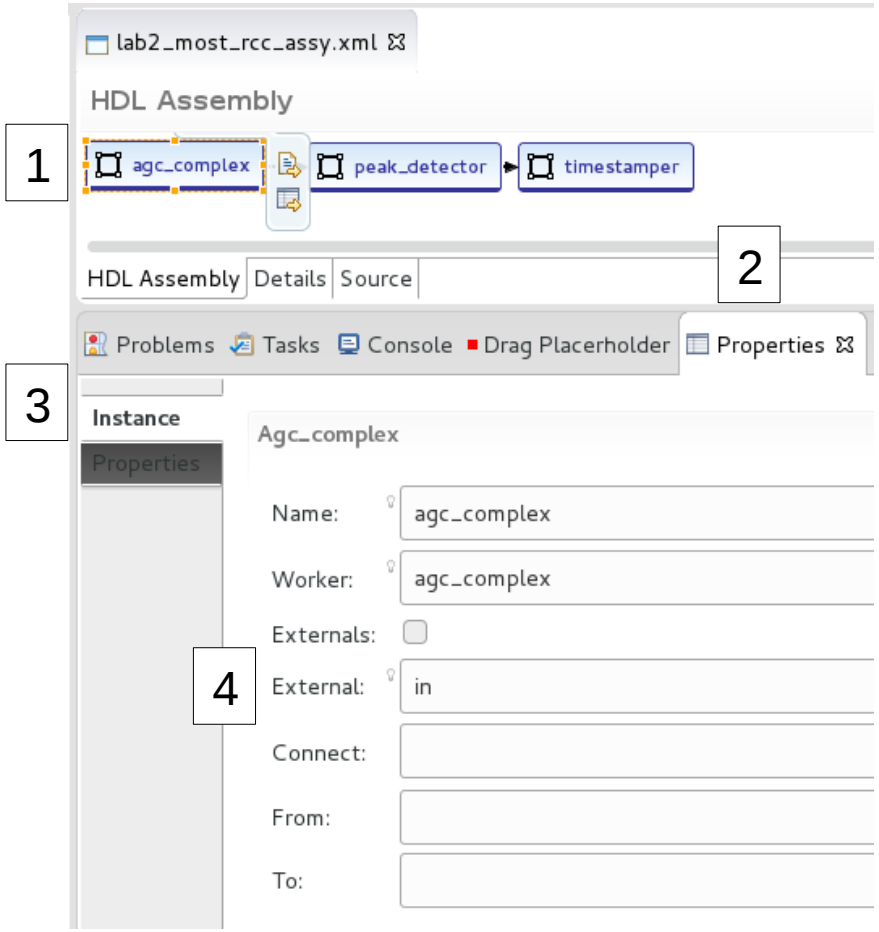
- Delete nothing worker
  - This worker is automatically placed by the framework to ensure the generated HDL assembly can be executed without editing the generated file
- Generate HDL Assembly XML using IDE
  1. Drag and drop workers into assembly **(diagram on next slide)**
     - Workers not specs. Located in .hdl directories
  2. Make connections in between workers

Adding Workers to HDL Assembly



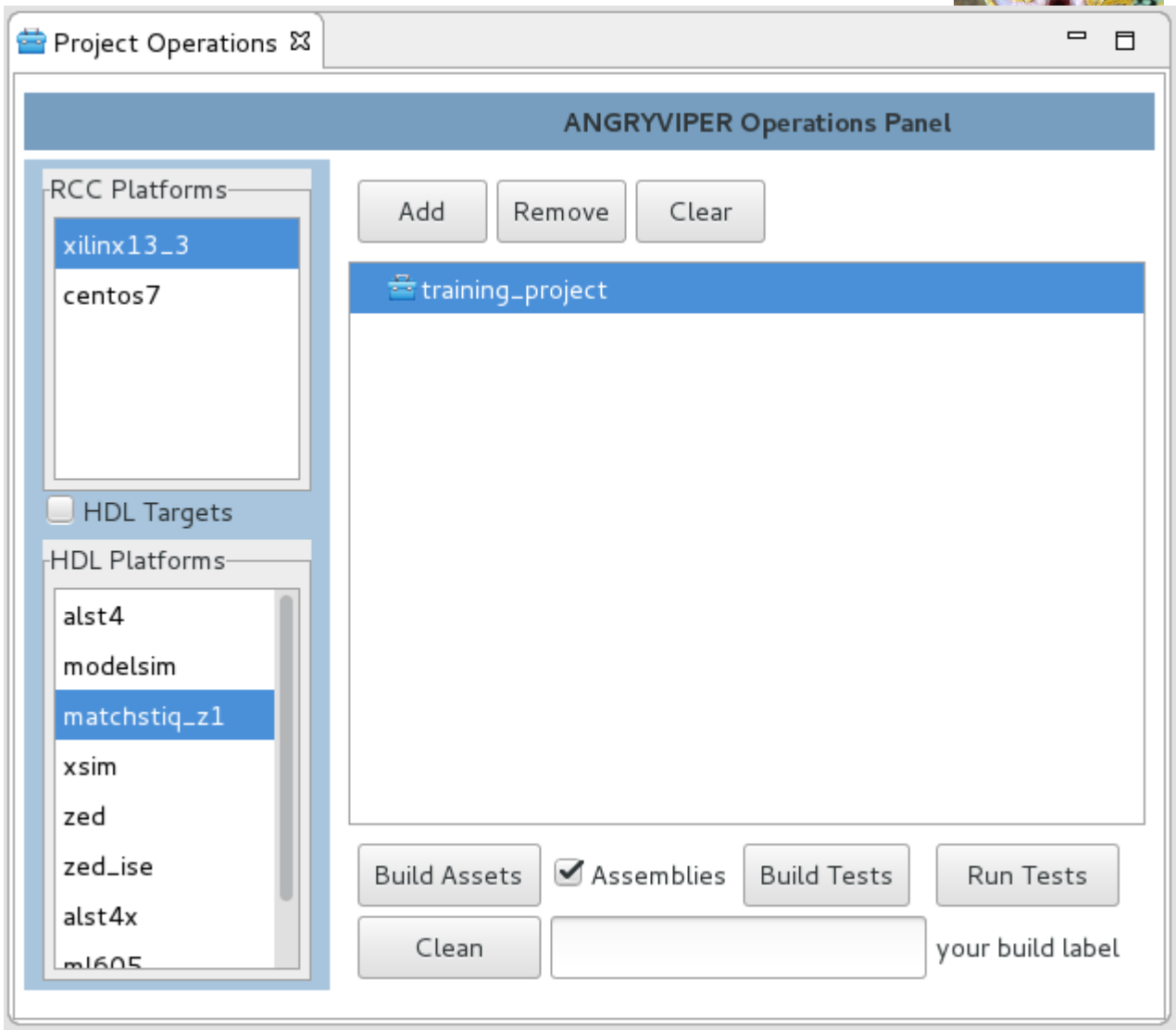
Step 3.3

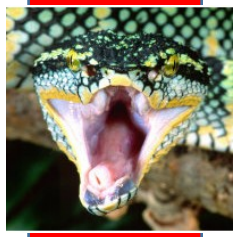
- Specify External connections for input and output ports of assembly
  - agc_complex -> in
  - timestamper -> out
- To specify external port for worker
  1. Click worker in IDE
  2. Navigate to Properties tab
  3. Click the Instance subtab
  4. Enter name of port in ‘External’ field



Step 3.4

- Build HDL Assembly
  - **IDE: “Refresh” the OpenCPI Projects panel**
  - Use the IDE to “**Add**” the training project to the ANGRVIPER Operations Panel
  - **Highlight** RCC Platforms “xilinx13_3” and HDL Platforms “matchstiq_z1”
  - **Check** the Assemblies RCC Platforms “xilinx13_3” and HDL Platforms “matchstiq_z1”
  - **Click** “Build Assets”
    - This command will build a FPGA image for the Matchstiq Z1 and compile all RCC workers for the ARM architecture
    - This step takes ~20 minutes to complete

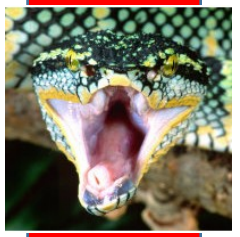




Part 4

Deploy the application on the Matchstiq-Z1

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>

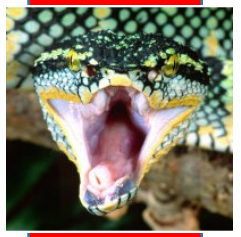
More detail on ocpirun can be found in the **OpenCPI Application Development Guide** document

Step 4.1

- Setup deployment platform
  1. Connect to serial port via USB on rear of Matchstiq Z1 on Host
     - ‘screen /dev/ttyUSB0 115200’
  2. Boot and login into Petalinux on Matchstiq Z1
     - 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>’

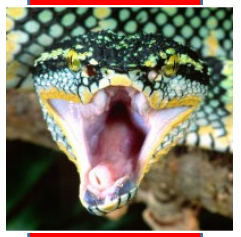
More detail on this process can be found in the **Matchstiq_Z1 Getting started Guide** document

Step 4.2



- Setup environment on Matchstiq Z1
  - The OCPI_LIBRARY_PATH environment variable is used to locate deployable artifacts
    - To deploy this application, the needed artifacts should be in OCPI_LIBRARY_PATH
      - Software worker .so files
      - HDL container .bitz files
    - To set OCPI_LIBRARY_PATH on Matchstiq Z1
      - ‘export OCPI_LIBRARY_PATH=/mnt/ocpi_core/exports:/mnt/training_project/’

Step 4.3



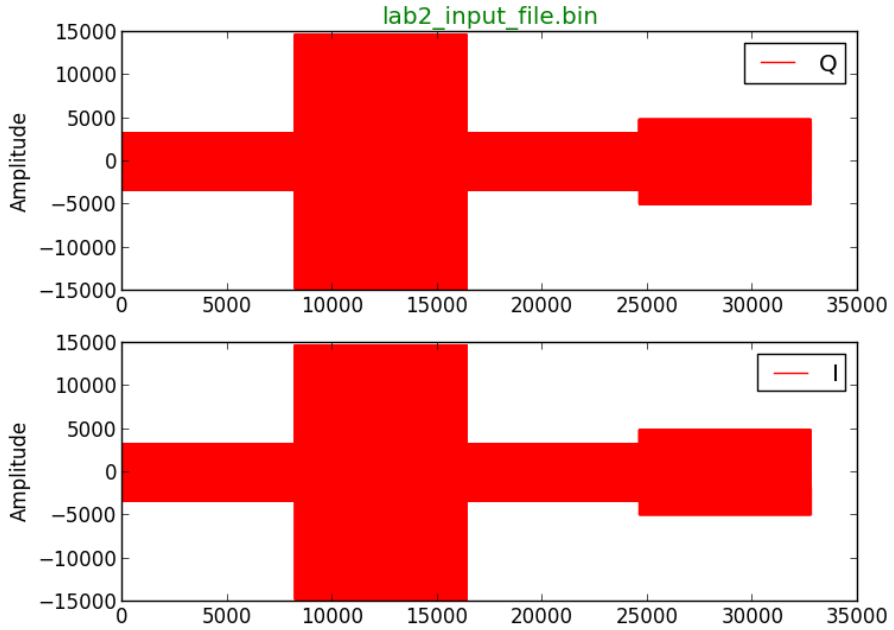
- Run application using ocpirun on Matchstiq Z1
- To run application on Matchstiq Z1:
  1. Navigate to OAS XML:
     - ‘cd /mnt/training_project/applications’
  1. Pass OAS XML to ocpirun:
     - ‘ocpirun -t 1 -d -v -mcomplex_mixer=rcc lab2_app.xml’

Expected result – Input Data

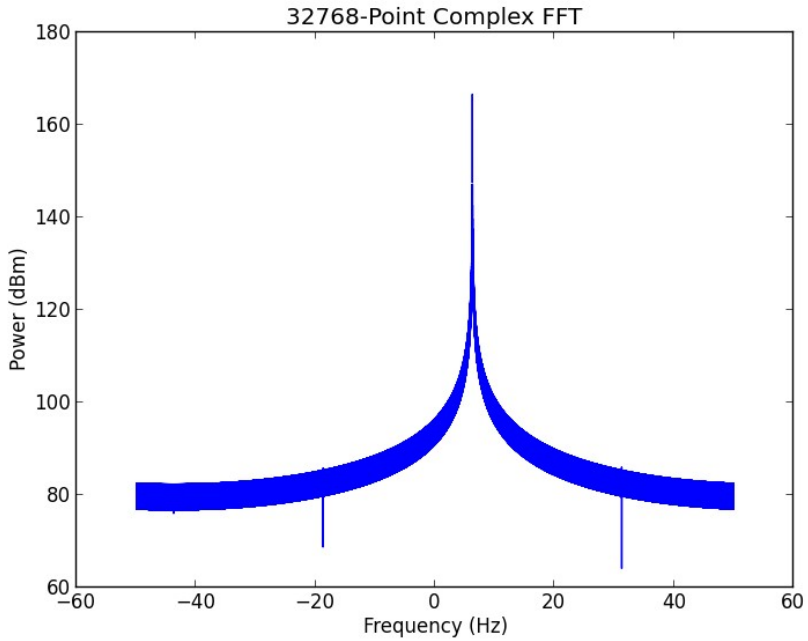
On Host:

‘cd ~/training_project/applications’

‘python scripts/plot.py idata/lab2_input_file.bin complex 32768’



Time Domain
3 distinct input levels – from AGC unit test lab



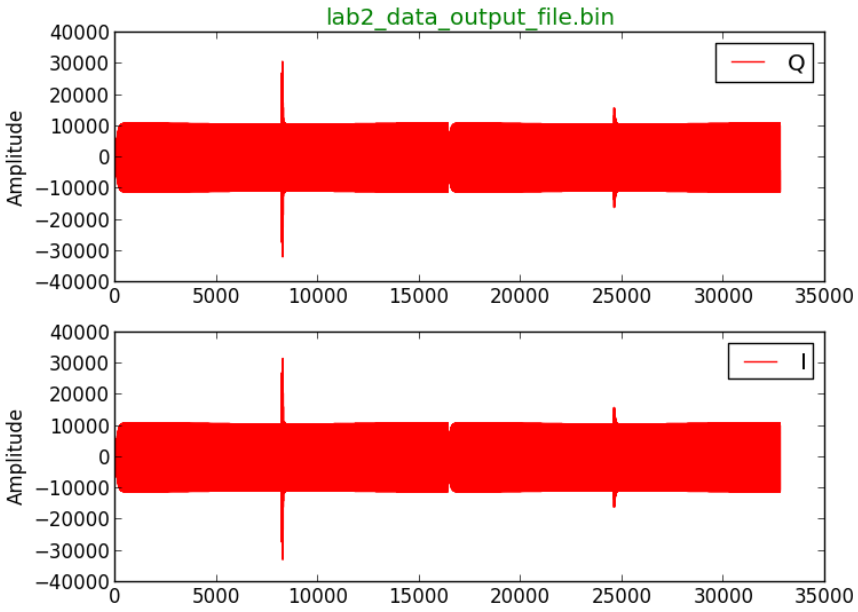
Frequency Domain
Tone at Fs/16 – from AGC unit test lab

Expected result – Output Data

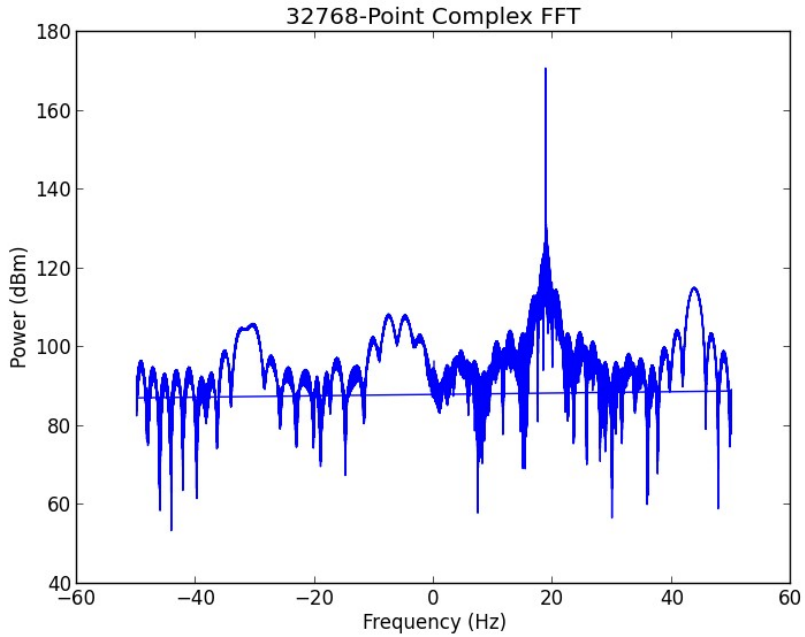
On Host:

‘cd ~/training_project/applications’

‘python scripts/plot.py odata/lab2_data_output_file.bin complex 32768’



Time Domain
Single input level – minus settling from AGC



Frequency Domain
Tone at Fs/16 minus 10 – Mixer shifted 10 Hz

Expected result – Output Timestamps

On Host:

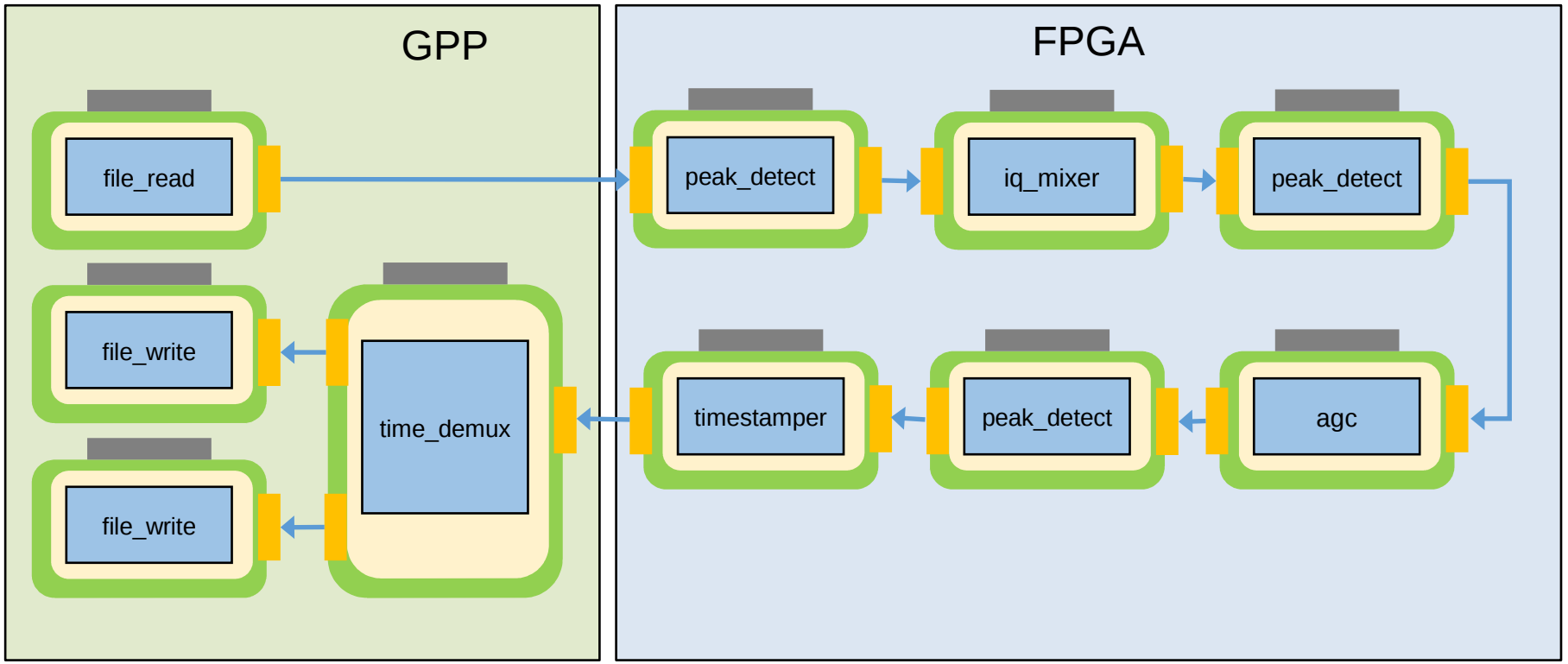
‘cd ~/training_project/applications’

‘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 ***
********************************************************************************
```

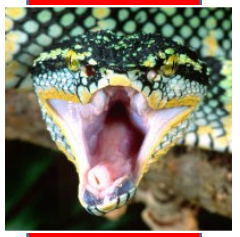
Timestamps are incrementing and uniformly spaced (approximately)

First Implementation: Most Possible HDL Workers

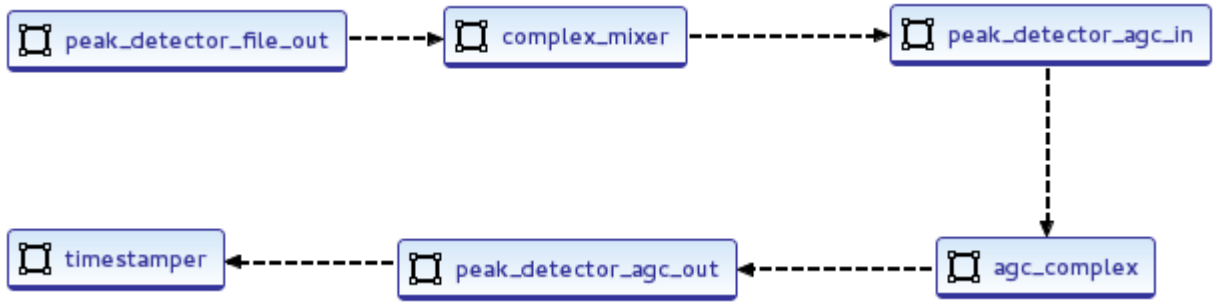


Repeat Parts 3 and 4 for this implementation

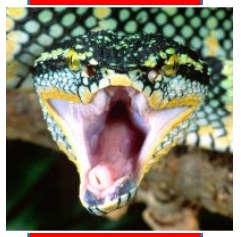
Most Possible HDL Workers: Hints



- 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
  - ocpirun command: ocpirun -t 1 -d -v -mcomplex_mixer=hdl lab2_app.xml



Remove training project



- The remainder of the labs will be recreating the project and application that was just built, one component at a time
- To remove the training project
  - cd ~
  - ocpidev delete project training_project/
- Reboot Matchstiq Z1 before lab 3