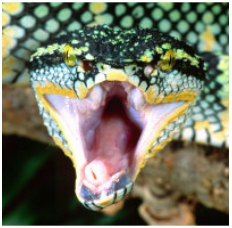


Lab 7: HDL Application Worker using HDL Primitive Library

agc_complex.hdl

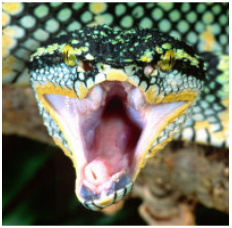
Objective

- Design, Build and Test: **HDL** Application worker
- Function: Automatic Gain Control (agc_complex.hdl)
- Create OpenCPI HDL Primitive Library (Used by Worker!)
- Create OpenCPI Component Spec (OCS)
- Properties Parameters to set VHDL generics unique to Worker (OWD)
- Convert Data Port Interface: Interleaved to Parallel (OWD)
- Routes data/messages through the worker “pass-thru” (VHDL)
- Automated Unit Testing on multiple platforms (XML, Python)
 - Simulator: Xilinx XSIM
 - Hardware: Matchstiq-Z1 (Xilinx Zynq-7020)
- Not a “Work-alike”



What's Provided

- Component Datasheet
 - /home/training/provided/doc/AGC_Complex.pdf
- Scripts for generating and validating test data
 - /home/training/provided/lab7/agc_complex.test/
- Worker VHDL with “commented” instructions
 - /home/training/provided/lab7/agc_complex.vhd
- HDL Primitive and Package VHDL
 - /home/training/provided/lab7/prims/



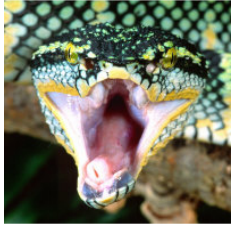
Step 0 – Create HDL Primitive Library Flow



- 1) IDE: File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard
 - Asset Type: **HDL Primitive Library**
 - Primitive Library Name: **prims**
 - Add to Project: **training_project**
- 2) Copy provided primitive source file agc.vhd into the primitive library
 - `$ cd /home/training/training_project/hdl/primitives/prims`
 - `$ cp -r /home/training/provided/lab7/prims/agc/ .`
- 3) Copy the provided primitive package prims_pkg.vhd into the primitive library
 - `$ cp /home/training/provided/lab7/prims/prims_pkg.vhd .`
- 4) Modify the HDL primitive library Makefile to list source files
 - Edit “/home/training/training_project/hdl/primitives/**prims**/Makefile” to include, ensuring it is located **PRIOR** to the include statement

SourceFiles=prims_pkg.vhd agc/src/agc.vhd

Step 0 – Build HDL Primitive Library



- Build the HDL primitive library for both XSIM and Zynq targets

- 1) **IDE:** “**Refresh**” the OpenCPI Projects panel

- 2) Use the IDE to “**Add**” the top-level “primitives” library to the Project Operations panel

- 3) **Check** the HDL Targets box and **highlight** “xsim” and “zynq”, under “xilinx”

- 4) **Click** “Build”

- 5) Review the Console window messages to ensure this step is error free

- Review Build Logs within the Console

```
[ocpidev -d /data/training/training_project build hdl primitive library prims --hdl-target xsim --hdl-target zynq]
```

```
make: Entering directory `/data/training/training_project/hdl/primitives/prims'
```

```
Building the prims library for xsim (target-xsim/prims) 0:()
```

```
Tool "xsim" for target "xsim" succeeded. 0:00.44 at 13:00:53
```

```
Building the prims library for zynq (target-zynq/prims) 0:()
```

```
Tool "vivado" for target "zynq" succeeded. 0:20.46 at 13:01:14
```

```
make: Leaving directory `/data/training/training_project/hdl/primitives/prims'
```

- **IDE:** “**Refresh**” the Project Explorer panel to observe new artifacts under prims/: “target-xsim/” and “target-zynq/”

App Worker Development Flow



- 1)OPS: Use pre-existing or create new
- 2)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

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 \Rightarrow 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 \Rightarrow Create new OCS

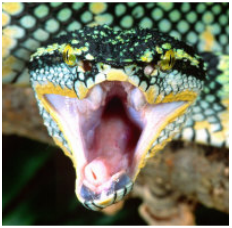
ANSWER: Must create a new OCS XML file

Step 2 – Create/Edit OCS



- Reference the “Component Spec Properties and Ports” table in the Component Datasheet
- **IDE: File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard**
 - **Asset Type: Component**
 - Component Name: **agc_complex**
 - Add To Project: **training_project**
 - Location: **components Library (default)**
- Edit the OCS via the IDE: OCS Editor (“Design” tab)
 - “**Add a Port**” named “in”, Protocol=“**iqstream_protocol**”
 - “**Add a Port**” named “out”, “**Producer**” checked, Protocol=“**iqstream_protocol**”

Step 3 - Create App Worker



- **IDE: File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard**
 - **Asset Type: Worker**
 - Worker Name: **agc_complex**
 - Add To Library: **components (default)**
 - Component: **agc_complex(ocpi.training)**
 - Model: **HDL**
 - Programming Language: **VHDL**
- **IDE: Refresh** the Project Explorer, then **Refresh** the OpenCPI Projects
- Use the Project Explorer to examine the auto-generated directories and files
 - components/{worker}.hdl/ - Worker directory with Author Model suffix (.hdl)
 - components/{worker}.hdl/Makefile - Includes a standard makefile fragment from the OCPI CDK
 - components/{worker}.hdl/{worker}.xml - OWD XML file
 - components/{worker}.hdl/{worker}.vhd - VHDL (architecture) skeleton file
 - components/{worker}.hdl/gen/ - OCPI worker build artifacts ({worker}-impl.vhd contains the Entity!)

Step 3 – Build Skeleton Code

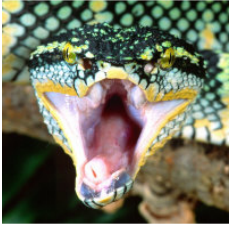


- Generated skeleton code can be built!?
 - 1) “**Refresh**” the OpenCPI Projects panel
 - 2) Use the IDE to “**Add**” the App Worker to the Project Operations panel
 - 3) **Check** the HDL Targets box and **highlight** “xsim” and “zynq”
 - 4) **Check** “Assets” Radio Button
 - 5) **Click** “Build”
 - 6) Review the Console window messages to ensure this step is error free
- New Build Artifacts in **target-xsim/** and **target-zynq/**
- Although not very exciting, this step proves the skeleton source code is build-able and the build engine is functional

Step 3 – Add Properties to Worker



- Reference the “Worker Properties” table in the Component Datasheet
- Using IDE: OWD HDL Editor (“Design” tab), add properties:
 - DATA_WIDTH_p
 - AVG_WINDOW_p
 - hold
 - ref
 - mu
 - messageSize
- Ensure all attributes and default values are set for each property/parameter



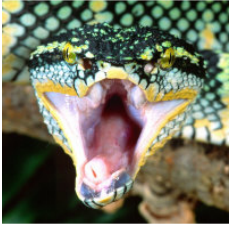
Step 3 – Convert Data Port to Parallel

- Convert data port interface from Interleaved to Parallel
 - Note: iqstream_protocol is default interleaved 16bit I/Q sample data
- Determine port configuration settings by examining the “Worker Interfaces” table in the Component Datasheet
- Edit OWD via the IDE: OWD HDL Editor (“Design” tab)
 - 1) “**Add a StreamInterface**” port named “**in**” and set “DataWidth” to **32**
 - 2) “**Add a StreamInterface**” port named “**out**” and set “DataWidth” to **32**
- Expanding the data interface to 32 bit allows 16bit I and 16bit Q data to be transmitted simultaneously, *i.e.* parallel

Step 3 – Rebuild App Worker

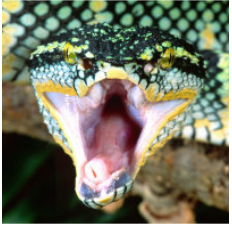


- Build Worker with changes from OWD.
 - 1) “**Refresh**” the OpenCPI Projects panel
 - 2) Use the IDE to “**Add**” the App Worker to the Project Operations panel
 - 3) **Check** the HDL Targets box and **highlight** “xsim” and “zynq”
 - 4) **Check** “Assets” Radio Button
 - 5) **Click** “Build”
 - 6) Review the Console window messages to ensure this step is error free
- Notice that the new properties and ports modifications are now added in the gen/agc_complex-impl.vhd.



Step 3 – Modify App Worker Source Code

- The skeleton {worker}-skel.vhd file is an empty architecture
 - The entity is generated VHDL based on OCS and OWD, and located in the gen/{worker}-impl.vhd
- Replace skeleton .vhd file with *provided* .vhd
 - Contains instructions to modify the VHDL source code
 - 1) \$ `cd /home/training/training_project/components/agc_complex.hdl/`
 - 2) \$ `cp /home/training/provided/lab7/agc_complex.vhd .`
- Open the VHDL in a text editor. Starting at the top, modify the source code per the commented instructions. A summary of required modifications is provided:
 - **Initialize the constants with parameter values**
 - **Make general signal and data port assignments**
 - **Wire signals to the AGC primitive instantiations**



Step 4 – Include HDL Primitive Library in search path

- Two methods to include local HDL Primitive Libraries in the search path
 - Library.mk \Leftarrow Common to all workers in a Library (**Implement this method!**)
 - Worker's Makefile \Leftarrow Localized to specific Worker

1) Create the Library.mk file in:

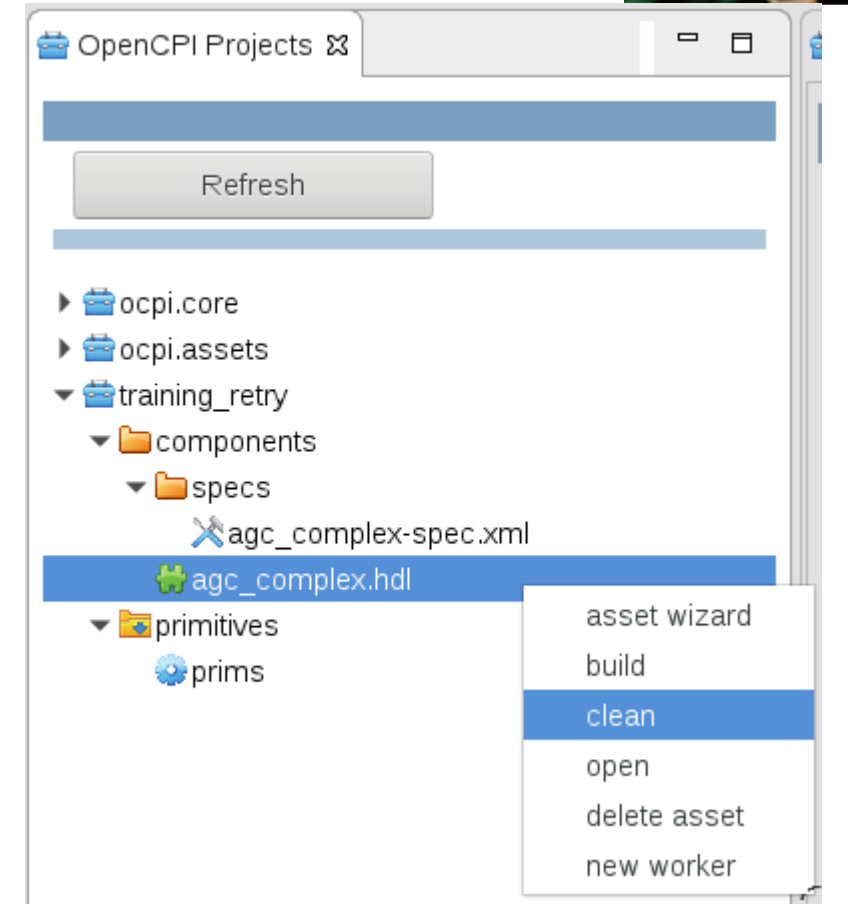
`/home/training/training_project/components/Library.mk`

2) Edit the Library.mk file to include:

`HdlLibraries+=prims`

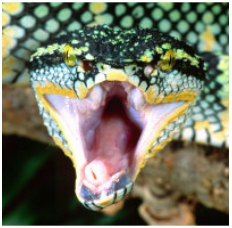
Supplement: Temporary Fix for 1.4

- There is an issue where a previously built HDL worker will not recognize new changes to the .vhd source file.
- This will be fixed in a later release.
- To address this issue, right click the worker in the OpenCPI Projects panel and click clean.
- OR touch the worker OWD file.

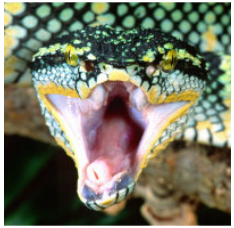


Step 4 - Build the App Worker

- Build HDL App Worker for XSIM and Zynq Targets
 - 1) Use the IDE to “**Add**” the App Worker to the Project Operations panel
 - 2) **Check** the HDL Targets box and **highlight** “xsim” and “zynq”
 - 3) **Check** “Assets” Radio Button
 - 4) **Click** “Build”
 - 5) Review the Console window messages and address any errors



Step 4 – Review Build Logs and Artifacts



- End of build log should resemble the following, if free of errors:

```
[ocpidev -d /data/training/training_project build worker agc_complex.hdl -l components --hdl-target xsim --hdl-target zynq]
```

```
make: Entering directory `/data/training/training_project/components/agc_complex.hdl'
Generating the VHDL constants file for config 0: target-xsim/generics.vhd
Generating the opposite language definition file: gen/agc_complex-defs.vh
Generating the Verilog constants file for config 0: target-xsim/generics.vh
Building the agc_complex worker for xsim (target-xsim/agc_complex) 0:(ocpi_debug=false ocpi_endian=little)
Tool "xsim" for target "xsim" succeeded. 0:01.85 at 16:04:33
Creating link to export worker binary: ../lib/hdl/xsim/agc_complex -> target-xsim/agc_complex
Creating link from ../lib/hdl/xsim/agc_complex.vhd -> target-xsim/agc_complex-defs.vhd to expose the definition of worker agc_complex.
Creating link from ../lib/hdl/xsim/agc_complex.v -> target-xsim/agc_complex-defs.vh to expose the other-language stub for worker agc_complex.
Generating the VHDL constants file for config 0: target-zynq/generics.vhd
Generating the Verilog constants file for config 0: target-zynq/generics.vh
Building worker core "agc_complex" for target "zynq" 0:(ocpi_debug=false ocpi_endian=little) target-zynq/agc_complex.edf
Tool "vivado" for target "zynq" succeeded. 0:32.58 at 16:05:06
Creating link to export worker binary: ../lib/hdl/zynq/agc_complex.edf -> target-zynq/agc_complex.edf
Creating link from ../lib/hdl/zynq/agc_complex.vhd -> target-zynq/agc_complex-defs.vhd to expose the definition of worker agc_complex.
Creating link from ../lib/hdl/zynq/agc_complex.v -> target-zynq/agc_complex-defs.vh to expose the other-language stub for worker agc_complex.
make: Leaving directory `/data/training/training_project/components/agc_complex.hdl'
==> Command completed. Rval = 0
```

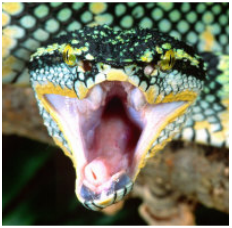
- Observe new artifacts {worker}.hdl/: “target-xsim/” and “target-zynq/”
 - These directories contain output files from their respective FPGA vendor tools (in this case, simply Xilinx Vivado) in addition to a framework auto-generated file, generics.vhd
 - ‘generics.vhd’ contains parameter configuration settings of the HDL worker for the particular “target-”

Step 5(a) - 7(a) Unit Test - Simulation



- Employ the framework's Unit Test Suite to generate:
 - OAS (OpenCPI Application Specification) XML file(s)
 - Used by the framework for running the executable on a simulation platform
 - In this case, the target simulation platform is Xilinx XSIM Simulation Server
 - OHAD (OpenCPI HDL Assembly Description) XML file(s)
 - Used by the framework to build an executable for the target simulation platform
 - In this case, the target simulation platform is Xilinx XSIM (xsim)
 - Input test data file(s) based on user provided scripts
 - Various scripts to manage the execution of the applications onto the target platform(s)

Step 5(a) - Create Unit Test

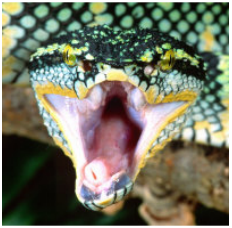


- Create a unit test for the “agc_complex” component, which results in generation of the “agc_complex.test/” directory. From the top of the project, execute:
- Create Unit Test
 - **File** → **New** → **Other** → **ANGRYVIPER** → **OpenCPI Asset Wizard** → **Unit Test**
 - **Add to Project:** *training_project*
 - **Add to Library:** *components*
 - **Component Spec:** *agc_complex-spec.xml*
- **OR** `$ ocpidev create test agc_complex`
 - Review contents of the <OCS>.test/
 - Stub files: Makefile, agc_complex-test.xml, generate.py, verify.py, view.sh
 - **IDE: Refresh** the Project Explorer, then **Refresh** the OpenCPI Projects

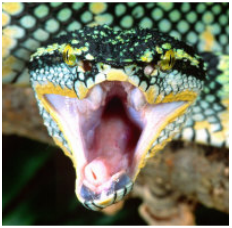
Step 5(a) - Create Unit Test

- Copy the generate.py, verify.py, and view.sh scripts from provided:

```
$ cd /home/training/training_project/components/agc_complex.test/  
$ cp -a /home/training/provided/lab7/agc_complex.test/generate.py .  
$ cp -a /home/training/provided/lab7/agc_complex.test/verify.py .  
$ cp -a /home/training/provided/lab7/agc_complex.test/view.sh .
```



Step 5(a) – Unit Test Description XML



- Update the agc_complex-test.xml to :

```
<tests useHDLFilelo='true'>
```

```
  <input port='in' script='generate.py 32768' messagesize='8192'/>
```

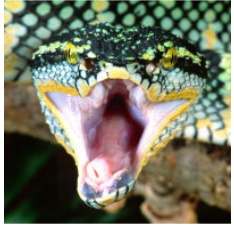
```
  <output port='out' script='verify.py 32768' view='view.sh'/>
```

```
  <property name='ref' value='0x1B26'/>
```

```
  <property name='mu' value='0x144E'/>
```

```
</tests>
```

Step 6(a) - Build Unit Test (Xilinx XSIM)



- Build Unit Test Suite for target simulation platform
 - 1) **IDE: Refresh** the Project Explorer, then **Refresh** the OpenCPI Projects
 - 2) Use the IDE to “**Add**” the Unit Test to the Project Operations panel
 - 3) **Highlight** “xsim” the HDL Platforms panel (HDL Targets box unchecked)
 - 4) **Check** “Tests” Radio Button
 - 5) **Click** “gen + build”
 - 6) Review the Console window messages and address any errors
- Observe new artifacts in `agc_complex.test/gen/`
 - `cases.txt` – “Human-readable” file listing 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
 - `assemblies/` - Used by framework to build bitstreams

Step 6(a) – Review Build Artifacts (Xilinx XSIM)



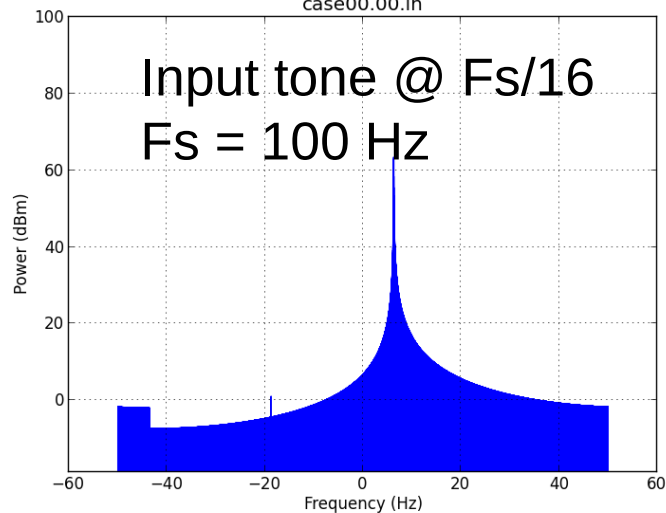
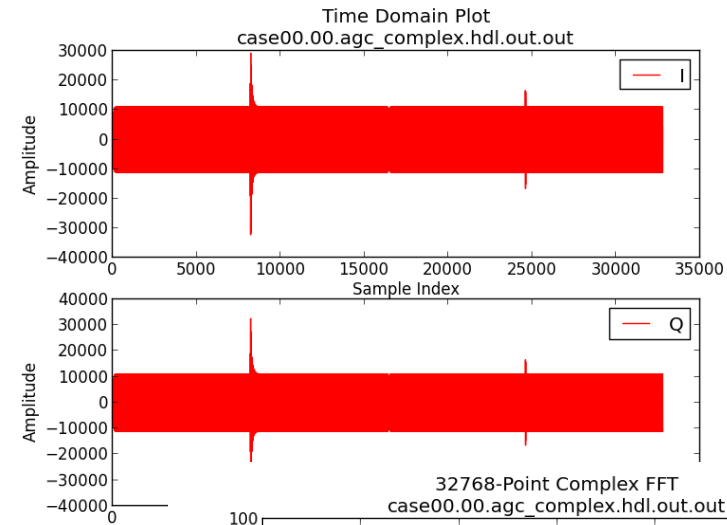
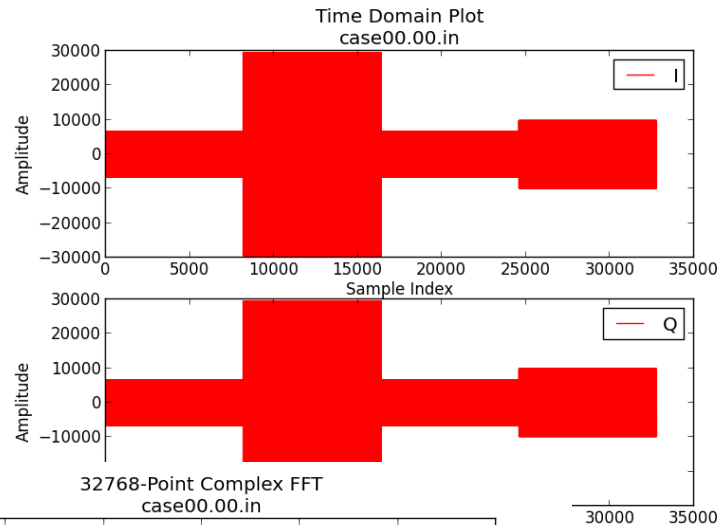
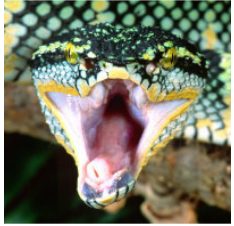
- Observe new artifacts in `agc_complex.test/gen/assemblies/agc_complex_0_frw/`
 - **agc_complex_0_frw.xml** – generated assembly xml (OHAD)
 - `gen/` - artifacts generated/used by framework
 - `lib/` - artifacts generated/used by framework
 - `target-xsim/` - artifacts generated/used by framework and FPGA tools
 - `container-agc_complex_0_frw_xsim_base/`
 - `gen/` - artifacts generated/used by framework
 - `target_xsim/`
 - artifacts generated/used by framework and output files from FPGA tools
 - Execution file for execution onto a Simulation platform

Step 7(a) - Run Unit Test (Xilinx XSIM)

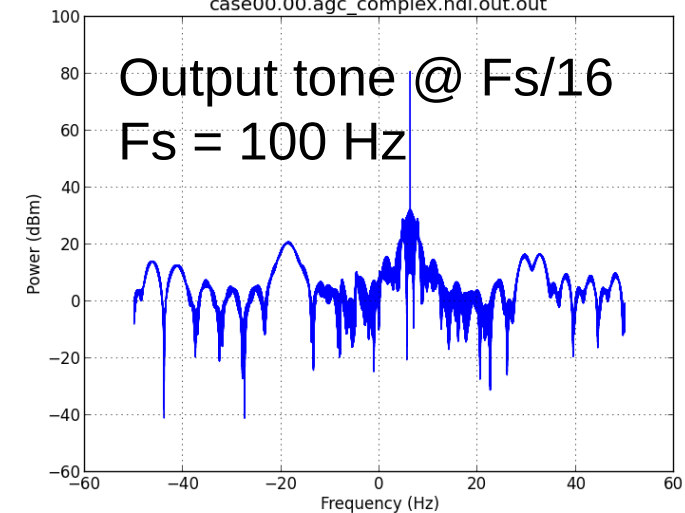


- Run Unit Test Suite for target simulation platform
 - 1) Use the IDE to “**Add**” the Unit Test to the Project Operations panel
 - 2) **Highlight** “xsim” the HDL Platforms panel (HDL Targets box unchecked)
 - 3) **Check** “Tests” Radio Button
 - 4) **Check** “run view script” to see the output of the view script
 - 5) **Check** “ keep simulations” so that we can see the waveform output later.
 - 6) **Click** “prep + run + verify” to run the tests
 - 7) Review the Console window messages and address any errors
- Simulation takes approximately 30 seconds to complete. Completion of each test case is reported in Console with a “PASSED” along with final values for the min/max peaks.
- 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=xsim KeepSimulations=1 View=1** (PLOTS ON NEXT PAGE)
 - \$ **make verify** {verify previous results}
 - \$ **make view** {plot previous results}

Step 7(a) – Unit Test I/O Plots (Xilinx XSIM)



ref = 0x1B26
mu = 0x144E



- 



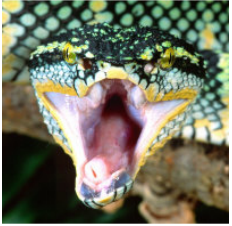
Step 5(b) – 7(b) - Unit Test Matchstiq-Z1



- Employ the framework's Unit Test Suite to generate:
 - OAS (OpenCPI Application Specification) XML file(s)
 - Used by the framework for running the bitstream on hardware platform
 - In this case, the target hardware platform is Matchstiq-Z1
 - OHAD (OpenCPI HDL Assembly Description) XML file(s)
 - Used by the framework to build an bitstream for the target hardware platform
 - In this case, the target hardware platform is Matchstiq-Z1 (matchstiq_z1)
 - Input test data file(s) based on user provided scripts
 - Various scripts to manage the execution of the applications onto the target platform(s)

Step 5(b) - Create Unit Test

- **REUSE** from Simulation portion of this lab!
- Located in “agc_complex.test/” directory
- No changes required Test XML



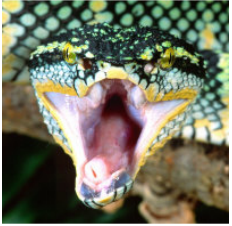
Step 6(b) - Build Unit Test (Matchstiq-Z1)



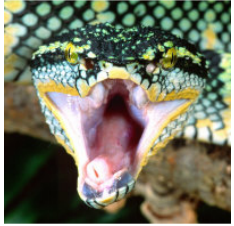
- Build Unit Test Suite for target hardware platform
 - 1) Use the IDE to “**Add**” the Unit Test to the Project Operations panel
 - 2) **Highlight** “matchstiq_z1” the HDL Platforms panel (HDL Targets box unchecked)
 - 3) **Check** “Tests” Radio Button
 - 4) **Click** “gen+build” to Build Tests
 - 5) Review the Console window messages and address any errors
- **NOTE:** The build process takes 5-10 mins to complete.

Step 6(b) – Review Build Logs (Matchstiq-Z1)

- Observe the console window to make sure the GUI completed successfully.



Step 6(b) - Review Build Artifacts (Matchstiq-Z1)



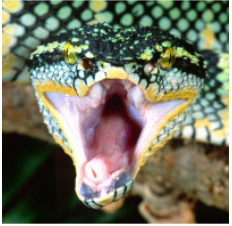
- Observe new artifacts in `agc_complex.test/gen/assemblies/agc_complex_0/`
 - **agc_complex_0.xml** – generated assembly xml (OHAD)
 - `gen/` - artifacts generated/used by framework
 - `lib/` - artifacts generated/used by framework
 - `target-zynq/` - artifacts generated/used by framework and FPGA tools
 - `container-agc_complex_0_matchstiq_z1_base/`
 - `gen/` - artifacts generated/used by framework
 - `target_zynq/`
 - artifacts generated/used by framework and output files from FPGA tools
 - **Bitstream** file for execution onto a hardware platform

Step 7(b) - Run Unit Test (Matchstiq-Z1)



- 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 (Matchstiq-Z1)

- Setup the remote test system:
 - In the Project Operations Panel
 - **Check** “Tests”
 - **Select** “remotes” radio button
 - **Click** “+remotes”
 - **Change** remote variable text to use matchstiq 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 (Matchstiq-Z1)



- Run Unit Test Suite for target hardware platform
 - 1) Use the IDE to “**Add**” the Unit Test to the Project Operations panel
 - 2) **Highlight** “matchstiq_z1” the HDL Platforms panel (HDL Targets box unchecked)
 - 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=matchstiq_z1 View=1**
 - \$ **make verify** {verify previous results}
 - \$ **make view** {plot previous results}