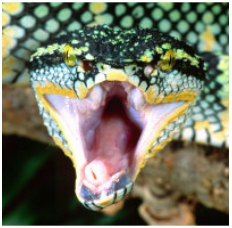


Lab 7: agc_complex.hdl

HDL Application Worker
using HDL Primitive Library

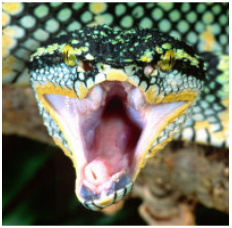
Objective

- 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: Ettus E310 (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/



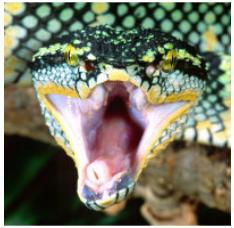
Step 0 – Create HDL Primitive Library Flow



- 1) IDE: File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard
 - Asset Type: **HDL Primitive Library**
 - Primitive Library Name: **prims**
 - Add to Project: **ocpi.training**
- 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



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

- 1) Refresh the OpenCPI Projects panel
- 2) 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
- 6) Refresh the Project Explorer panel to observe new artifacts under prims/: "target-xsim/" and "target-zynq/"

Configuration:

```
build ocpi.training.primitives HDL: xsim zynq
```

```
[ocpidev -d /home/training/training_project build hdl primitives --hdl-target xsim --hdl-target zynq]
```

No HDL platforms specified. No HDL assets will be targeted.

Possible HdIPlatforms are: alst4 alst4x e3xx isim matchstiq_z1 ml605 modelsim

picoflexor_s1t6a picoflexor_s3t6a xsim zed zed_ise.

make[1]: Entering directory `/home/training/training_project'

make[1]: Leaving directory `/home/training/training_project'

make[1]: Entering directory `/home/training/training_project/hdl/primitives'

===== For library prims:

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

Tool "xsim" for target "xsim" succeeded. 0:00.45 at 08:27:11

Creating directory ../lib/prims for library prims

No previous installation for target-xsim/prims in ../lib/prims/xsim.

Installing target-xsim/prims into ../lib/prims/xsim

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

Tool "vivado" for target "zynq" succeeded. 0:21.88 at 08:27:33

No previous installation for target-zynq/prims.libs in target-zynq/prims.

Installing target-zynq/prims.libs into target-zynq/prims

No previous installation for target-zynq/prims.sources in target-zynq/prims.

Installing target-zynq/prims.sources into target-zynq/prims

No previous installation for target-zynq/prims in ../lib/prims/zynq.

Installing target-zynq/prims into ../lib/prims/zynq

Installation suppressed for target-xsim/prims in ../lib/prims/xsim. Destination is identical.

Installation suppressed for target-zynq/prims.libs in target-zynq/prims. Destination is identical.

Installation suppressed for target-zynq/prims.sources in target-zynq/prims. Destination is identical.

Installation suppressed for target-zynq/prims in ../lib/prims/zynq. Destination is identical.

make[1]: Leaving directory `/home/training/training_project/hdl/primitives'

make[1]: Entering directory `/home/training/training_project'

make[1]: Leaving directory `/home/training/training_project'

Updating project metadata...

== > Command completed. Rval = 0

App Worker Development Flow



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

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

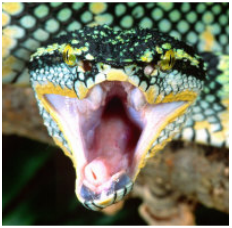


- Reference the "Component Spec Properties and Ports" table in the Component Datasheet

Via the IDE:

- 1) **File** → **New** → **Other** → **ANGRYVIPER** → **OpenCPI Asset Wizard**
- 2) Asset Type: **Component**
- 3) Component Name: **agc_complex**
- 4) Add To Project: **ocpi.training**
- 5) Location: **components Library** (default)
- 6) Finish
- 7) Refresh the Project Explorer
- 8) Located the OCS in the Project Explorer and open for editing in the OCS Editor ("Design" tab)
 - **"Add a Port"** named **"in"**, Protocol=**"iqstream_protocol"**
 - **"Add a Port"** named **"out"**, Protocol=**"iqstream_protocol"** and check the box for **"Producer"**

Step 3 - Create App Worker



- Via the IDE:
 - 1) **File** → **New** → **Other** → **ANGRYVIPER** → **OpenCPI Asset Wizard**
 - 2) Asset Type: **Worker**
 - 3) Worker Name: **agc_complex**
 - 4) Add To Project: **ocpi.training**
 - 5) Model: **HDL**
 - 6) Programming Language: **VHDL**
 - 7) Add To Library: **components**
 - 8) Component: **agc_complex (ocpi.training)**
 - 9) Finish
 - 10) Refresh the Project Explorer, then Refresh the OpenCPI Projects
 - 11) 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 (WHY!?)

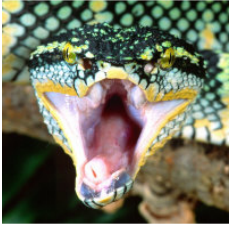


- Because...
 - Although not very exciting, this step proves the skeleton source code is build-able and the build engine is functional
- Build Generated Skeleton Code
 - 1) In the IDE, add the App Worker to the Project Operations panel
 - 2) Check the HDL Targets box and **highlight** "xsim" and "zynq"
 - 3) Check "Assets" Radio Button
 - 4) Click "Build"
 - 5) Review the Console window messages to ensure this step is error free
 - 6) Refresh the Project Explorer panel and review the newly generated build artifacts in **target-xsim/** and **target-zynq/**

Step 3 – Add Properties to Worker



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



Step 3 – Configure Data Port for 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: HDL App Worker OWD HDL Editor ("Design" tab)
 - 1) Highlight "Ports" from the "Outline"
 - 2) Click "Add a StreamInterface" and fill in the name "**in**" and set "DataWidth" to "**32**"
 - 3) Highlight "Ports" from the "Outline"
 - 4) Click "Add a StreamInterface" and fill in the name "**out**" and set "DataWidth" to "**32**"
- Expanding the data interface to 32 bit allows 16 bit I and 16 bit Q data to be transmitted simultaneously, i.e. parallel

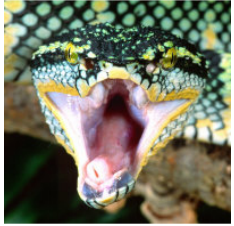
Step 3 – Configure for Version 2 HDL API



- The v1.5 release of IDE does not provide a field for declaring Version 2 HDL API, so the XML must be manually modified
 - 1) Edit OWD via the IDE: HDL App Worker OWD HDL Editor
 - 2) Switch the tab from “Design” to “Source”
 - 3) Add the top-level attribute Version=”2”
 - 4) Add InsertEOM=”1” to the “out” port

```
<HdlWorker language='vhdl' spec='agc_complex-spec' Version="2">  
  <Property Name="data_width" Type="uShort" Default="16" Parameter="true"></Property>  
  ...  
  <StreamInterface Name="in" DataWidth="32"></StreamInterface>  
  <StreamInterface Name="in" DataWidth="32" InsertEOM="1"></StreamInterface>  
</HdlWorker>
```

Step 3 – Configure for Version 2 HDL API



- After the OWD has been configured for Version 2, the name of the architecture in the generated VHDL file must also be changed

- 1) Open the VHDL in a text editor
- 2) Change the name from “agc_complex_worker” to “worker”

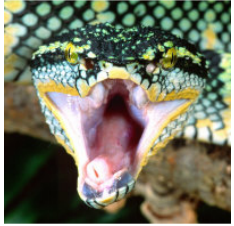
```
library IEEE; use IEEE.std_logic_1164.all; use ieee.numeric_std.all;
library ocpi; use ocpi.types.all; -- remove this to avoid all ocpi name collisions
architecture rtl of worker is
begin
    ctl_out.finished <= btrue; -- remove or change this line for worker to be finished when appropriate
    -- workers that are never "finished" need not drive this signal
    -- Skeleton assignments for agc_complex's volatile properties
end rtl;
```

Step 3 – Rebuild App Worker

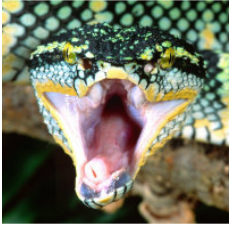


- IDE: Build Worker with changes from OWD to update gen/*
 - 1) Refresh the OpenCPI Projects panel
 - 2) Add the App Worker to the Project Operations panel
 - 3) Check the HDL Targets box and **highlight** "xsim" and "zynq"
 - 4) Check "Assets" Radio Button
 - 5) Click "Build"
 - 6) Review the Console window messages to ensure this step is error free
 - 7) Examine gen/{worker}-impl.vhd to observe that the new OWD properties and ports modifications are now available

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 (look for “TODO”). A high-level 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 (if it does not exist) the Library.mk file in:

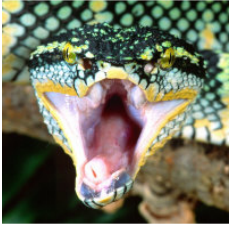
```
$ touch /home/training/training_project/components/Library.mk
```

2) Edit the Library.mk file to include:

```
HdlLibraries+=prims
```

Step 4 - Build the App Worker

- Build HDL App Worker for XSIM and Zynq Targets
 - 1) In the IDE, 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

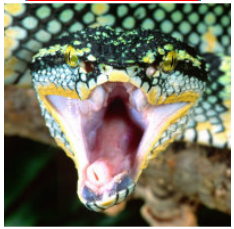
- End of build log should resemble the following, if free of errors:

```
Configuration:
build ocpi.training.components.agc_complex.hdl HDL: xsim zynq

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

make: Entering directory `/home/training/training_project/components/agc_complex.hdl'
Building the agc_complex worker for xsim (target-xsim/agc_complex) 0:(agc_complex_ocpi_max_opcode_out=0
agc_complex_ocpi_max_bytes_in=8192 agc_complex_data_width=16 agc_complex_ocpi_max_opcode_in=0
agc_complex_ocpi_max_bytes_out=8192 agc_complex_avg_window=16 agc_complex_ocpi_debug=false agc_complex_ocpi_version=2
agc_complex_ocpi_max_latency_out=256 agc_complex_ocpi_endian=little)
Tool "xsim" for target "xsim" succeeded. 0:02.61 at 09:44:47
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:(agc_complex_ocpi_max_opcode_out=0 agc_complex_ocpi_max_bytes_in=8192
agc_complex_data_width=16 agc_complex_ocpi_max_opcode_in=0 agc_complex_ocpi_max_bytes_out=8192 agc_complex_avg_window=16
agc_complex_ocpi_debug=false agc_complex_ocpi_version=2 agc_complex_ocpi_max_latency_out=256 agc_complex_ocpi_endian=little) target-
zynq/agc_complex.edf
Tool "vivado" for target "zynq" succeeded. 0:41.82 at 09:45:30
make[1]: Entering directory `/home/training/training_project/components'
make[1]: Leaving directory `/home/training/training_project/components'
make: Leaving directory `/home/training/training_project/components/agc_complex.hdl'
Updating project metadata...
== > 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

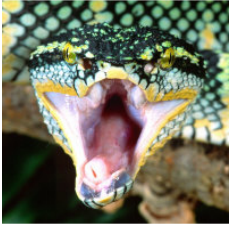


- Create a unit test for the "agc_complex" component, which results in generation of the "agc_complex.test/" directory. From the top of the project, execute:
- Via IDE:
 - 1) **File** → **New** → **Other** → **ANGRYVIPER** → **OpenCPI Asset Wizard** → **Unit Test**
 - 2) Add to Project: **ocpi.training**
 - 3) Add to Library: **components**
 - 4) Component Spec: **agc_complex-spec.xml**
- OR via Command-line:
 - \$ **ocpidev create test agc_complex**
 - Review contents of the <OCS>.test/
 - Stub files: Makefile, agc_complex-test.xml, generate.py, verify.py, view.sh
- Via IDE: Refresh the Project Explorer, then Refresh the OpenCPI Projects

Step 5(a) - Create Unit Test

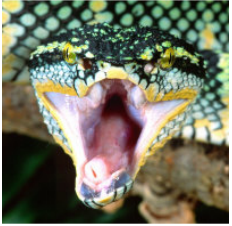
- Copy the generate.py, verify.py, and view.sh scripts from provided:

```
$ 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) – Edit Unit Test Description XML

- Final XML result is shown on later slide
- Open "agc_complex-test.xml" in the Unit Test Editor (GUI)
 - Direct the simulator to use HDL implementations for file operations
 - Click "Top Level Test Element" and Check "HDL File IO"
- Add and configure Input port
 - Click "Inputs" and press "Add"
 - From the component specification add the port "in" to the "Port" text box
 - Add the generate script with 32768 samples as an argument
 - In the "Script" text box add the value "generate.py 32768"



Step 5(a) – Edit Unit Test Description XML



- Add Output readers
 - Click "Outputs" and press "Add"
 - From the component specification add the "out" port
 - In the "**Port**" text box add the port name "out"
 - Add the verify script with 32768 samples as an argument
 - In the "Script" text box add the value "verify.py 32768"
 - Set the View script to view.sh
 - In the "View" text box add the value "view.sh"

Step 5(a) – Edit Unit Test Description XML



- Set property values
 - Add "ref" property with value "0x1B26"
 - Click "Properties" and click "Add"
 - Set "Name" to "ref"
 - In "Source of Test Values" section
 - Set "Value" to "0x1B26"
 - Add "mu" property with value "0x144E"
 - Click "Properties" and click "Add"
 - Set "Name" to "mu"
 - In "Source of Test Values" section
 - Set "Value" to "0x144E"

Step 5(a) – Unit Test Description XML (Solution)



- The resulting agc_complex-test.xml should look like this:

```
<Tests UseHDLFileIo='true'>
```

```
  <Input Port="in" Script="generate.py 32768"></Input>
```

```
  <Output Port="out" Script="verify.py 32768" View="view.sh"></Output>
```

```
  <Property Name="ref" Values="0x1B26"></Property>
```

```
  <Property Name="mu" Values="0x144E"></Property>
```

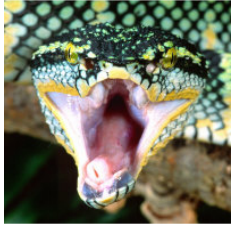
```
</Tests>
```

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



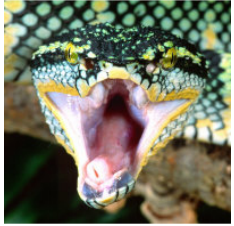
- Build Unit Test Suite for target simulation platform
 - 1) In the IDE, add the Unit Test to the Project Operations panel
 - 2) Highlight "xsim" the HDL Platforms panel (HDL Targets box unchecked)
 - 3) Check "Tests" radio button
 - 4) Click "gen + build"
 - 5) Review the Console window messages and address any errors
- Observe new artifacts in {OCS}.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)



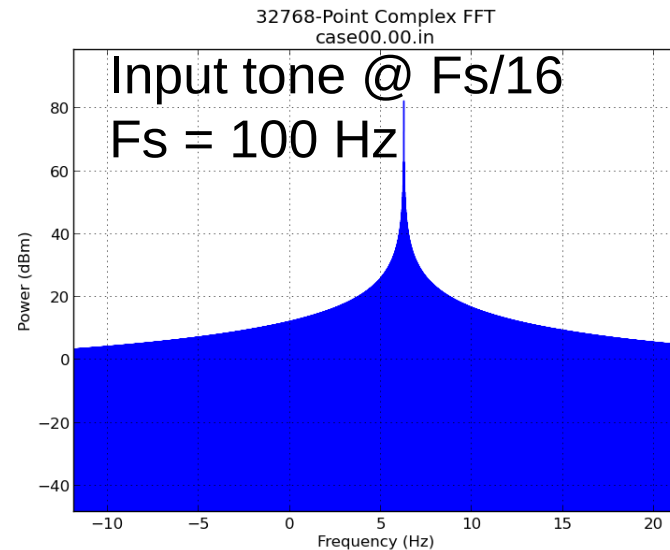
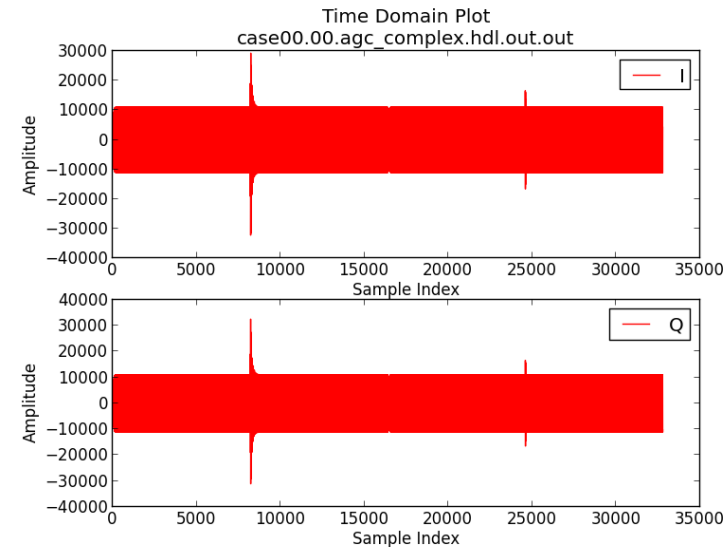
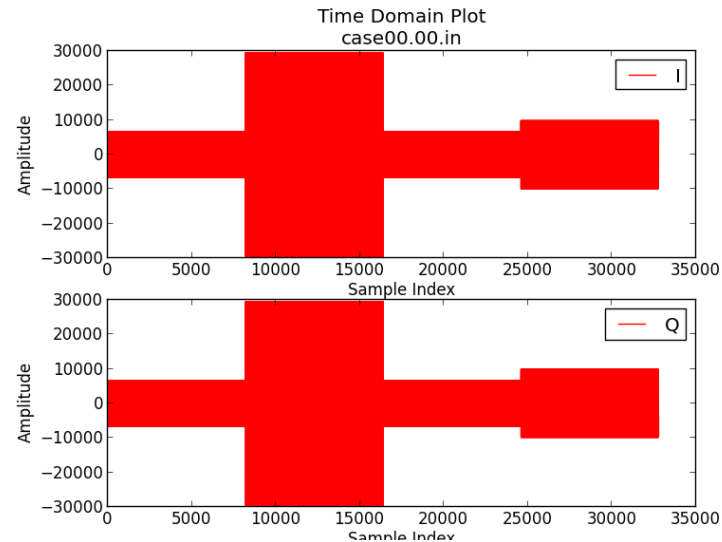
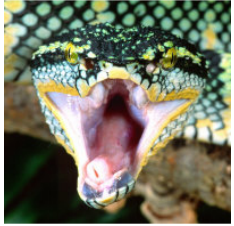
- Observe new artifacts in `agc_complex.test/gen/assemblies/agc_complex_0_frw/`
 - **agc_complex_0_frw.xml** – generated assembly xml (OHAD)
 - `gen/` - artifacts generated/used by framework
 - `lib/` - artifacts generated/used by framework
 - `target-xsim/` - artifacts generated/used by framework and 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)

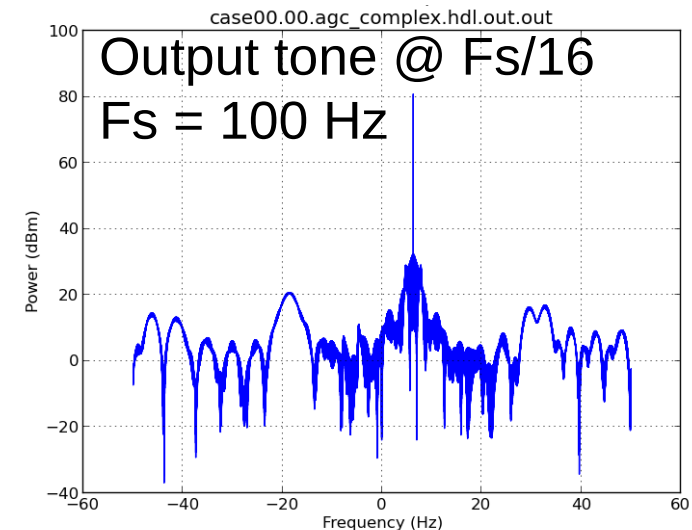


- Via IDE: Run Unit Test Suite for target simulation platform
 - 1) In the IDE, add the Unit Test (.test) to the Project Operations panel
 - 2) Highlight "xsim" the HDL Platforms panel (HDL Targets box unchecked)
 - 3) Check "keep simulations"
 - 4) Check "run view script"
 - 5) Click "prep+run+verify" to Run Tests
Simulation takes approximately 1 minute to complete.
 - 6) Review the Console window messages and address any errors
Completion of each test case is reported in Console with a "PASSED".
- Via Command-line terminal:
 - In a terminal window, execute within the {component}.test/
`$ ocpidev run --mode prep_run_verify --only-platform xsim --keep-simulations --view`

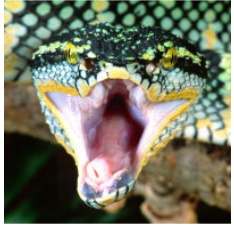
Step 7(a) – Unit Test I/O Plots (Xilinx XSIM)



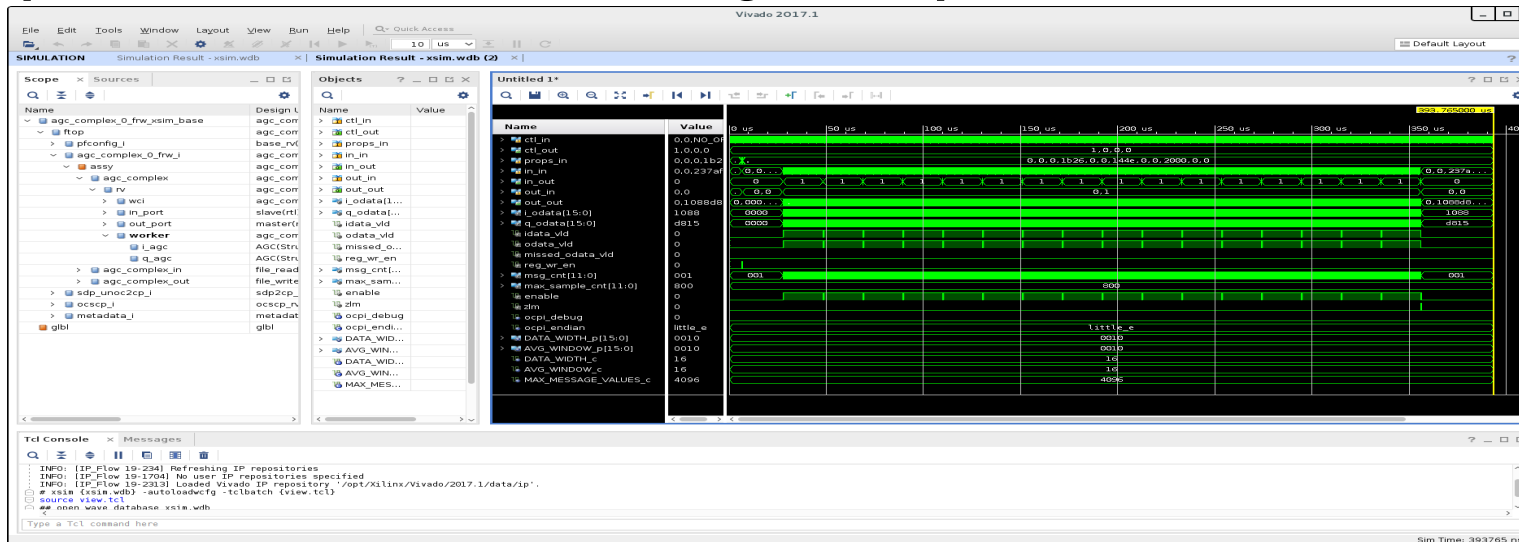
ref = 0x1B26
mu = 0x144E



Step 7(a) – View Simulation Waveforms (Xilinx XSIM)



- Must have checked "keep simulations" or ran
`$ ocpidev run --mode prep_run_verify --only-platform xsim --keep-simulations --view`
- In a terminal window, browse to `agc_complex.test/` and execute
`$ ocpiview run/xsim/case00.00.agc_complex.hdl.simulation/ &`



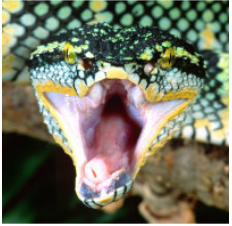
Step 5(b) – 7(b) - Unit Test Ettus E310



- 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 Ettus E310
 - 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 Ettus E310 (e3xx)
 - Input test data file(s) based on user provided scripts
 - Various scripts to manage the execution of the applications onto the target platform(s)

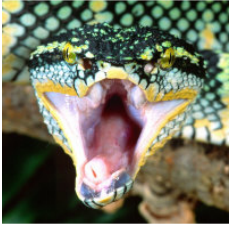
Step 5(b) - Create Unit Test

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



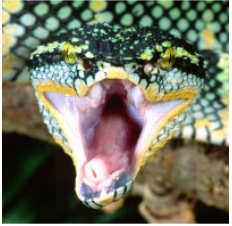
Step 6(b) - Build Unit Test (e3xx)

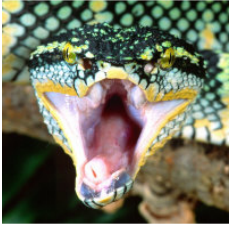
- Build Unit Test Suite for target hardware platform
 - 1) In the IDE, add the Unit Test to the Project Operations panel
 - 2) Highlight "e3xx" the HDL Platforms panel (HDL Targets box unchecked)
 - 3) Check "Tests" radio button
 - 4) Click "gen+build" to build tests
 - 5) Review the Console window messages and address any errors
- NOTE: The build process takes 5-10 mins to complete.



Step 6(b) – Review Build Logs (e3xx)

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





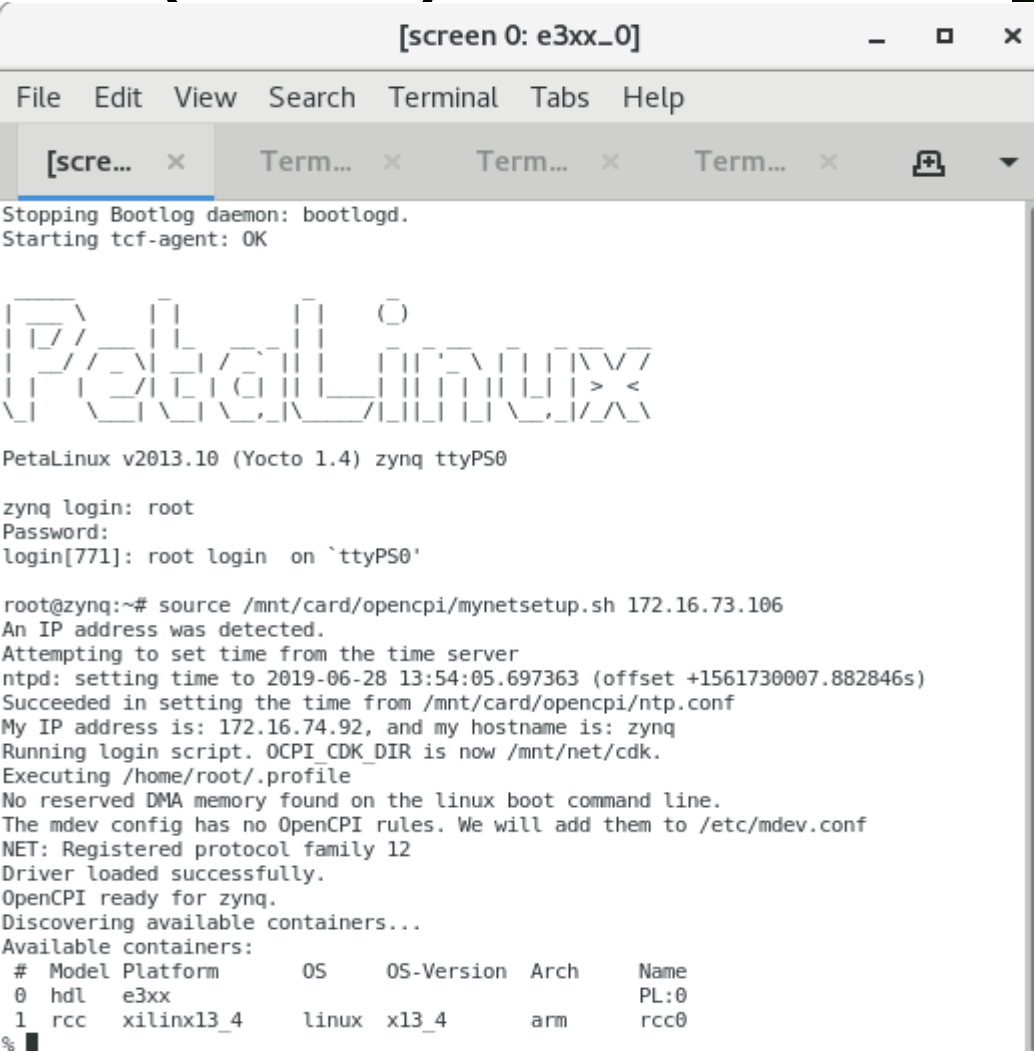
Step 6(b) - Review Build Artifacts (e3xx)

- 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_e3xx_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 (e3xx)

- 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



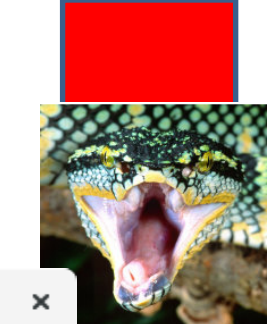
```
[screen 0: e3xx_0]
File Edit View Search Terminal Tabs Help
[scre... x Term... x Term... x Term... x
Stopping Bootlog daemon: bootlogd.
Starting tcf-agent: OK

Petalinux

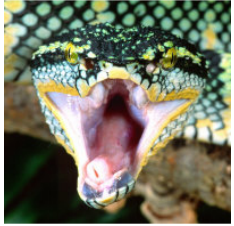
PetaLinux v2013.10 (Yocto 1.4) zynq ttyPS0

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 linux x13_4 arm PL:0
1 rcc xilinx13_4 linux x13_4 arm rcc0
% █
```



Step 7(b) - Run Unit Test (Ettus E310)



- Via IDE approach to running unit tests on remote platforms:
 - 1) In the “Project Operations” panel, select "remotes" radio button
 - 2) Click "+remotes"
 - 3) Change remote variable text to use Ettus E310's IP and point to the training project:
 - 4) {IP of Ettus E310}=root=root=/mnt/training_project
 - 5) Select the newly created remote. This will be the target remote test system. Unselected remotes will not be targeted
 - 6) Add the Unit Test to the Project Operations panel
 - 7) Highlight "e3xx" the HDL Platforms panel (HDL Targets box unchecked)
 - 8) Check "run view script"
 - 9) Click "prep + run + verify" to run tests
 - 10) Review the Console window messages and address any errors

Step 7(b) - Run Unit Test (Ettus E310)



- Via Command-line terminal:
 - In a terminal window, execute within the {component}.test/
`$ OCPI_REMOTE_TEST_SYSTEMS={IP of Ettus E310}=root=root/mnt_training \
ocpidev run --mode prep_run_verify --only-platform e3xx --keep-simulations --view`
- Perform an md5sum to ensure the output of the XSIM and E310 tests are the same

agc_complex.test\$ md5sum run/*/*.out

- 6c51985e57f22381baa71953166b8663 run/**xsim**/case00.00.agc_complex.hdl.out.out
- 6c51985e57f22381baa71953166b8663 run/**e3xx**/case00.00.agc_complex.hdl.out.out