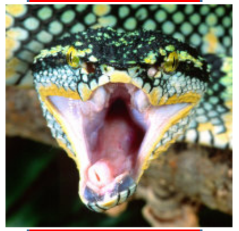
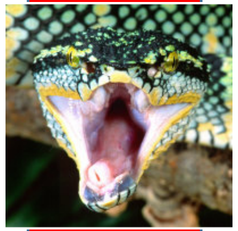


Lab 6: peak_detector.hdl

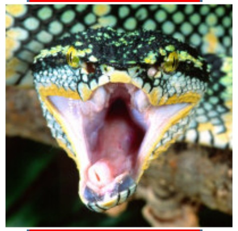
Simple HDL Application Worker



Objective

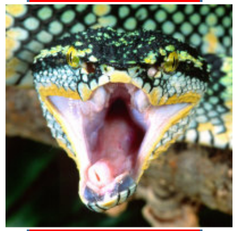
- Design, Build and Test: **HDL** Application Worker
- Function: Peak detection of a data stream (peak_detector.hdl)
- Volatile Properties: Signed Min/Max (OWD)
- Configure Data Port Interfaces: Interleaved to Parallel (OWD)
- Routes data/messages through the worker "pass-thru" (VHDL)
- Automated Unit Testing on multiple platforms (XML, Python)
  - Simulator: Xilinx XSIM
  - Hardware: Ettus E310 (Xilinx Zynq-7020)
- "Work-alike" to Simple RCC Worker (Lab 3)

What's Provided



- Component Datasheet
  - /home/training/provided/doc/Peak_Detector.pdf
- Scripts for generating and validating data
  - /home/training/provided/lab3/peak_detector.test/
  - **REUSE!**
  - **Recall: One {component}.test/ per OCS**
  - **Same as implemented in Simple RCC Application Worker (Lab3)**
    - **"Work-alike"**
- Script for plotting I/Q data
  - /home/training/provided/scripts/plotAndFft.py
- Worker VHDL with "commented" instructions
  - /home/training/provided/lab6/peak_detector.vhd

App Worker Development Flow

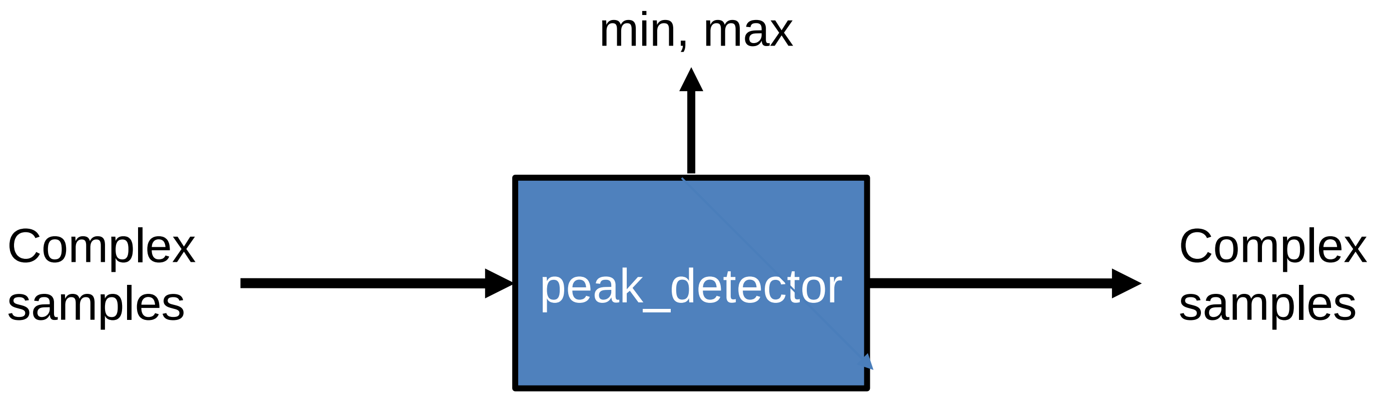


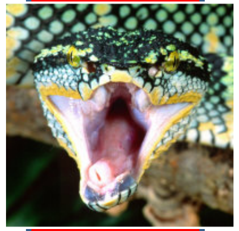
1) Protocol (OPS): Use pre-existing or create new
2) Component (OCS): Use pre-existing or create new
3) Create new App Worker (Modify OWD, Makefile, and source HDL/RCC code)
4) Build the App Worker for target device(s)
5) Create Unit Test ({component}-test.xml, generate, verify and view scripts)
6) Build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

Overview

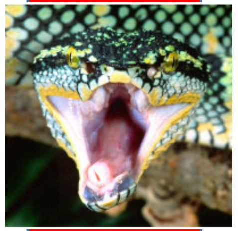
- What are the requirements of this component?
  - Given an input of complex numbers, the "peak_detector" component is meant to find the biggest I or Q sample and the smallest I or Q sample. The basic idea is:

    current_biggest = max(current_biggest,max(I,Q))
    current_smallest = min(current_smallest,min(I,Q))
  - Pass input of complex numbers to the output ports





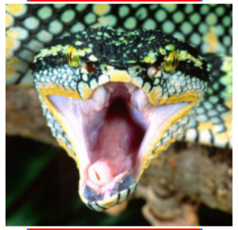
Step 1 – OPS: Use pre-existing or create new



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 ⇨ Create new OPS

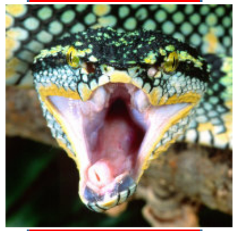
ANSWER: REUSE! OPS XML file is available from framework

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 OCS that satisfies the requirements 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 ⇨ Create new OCS

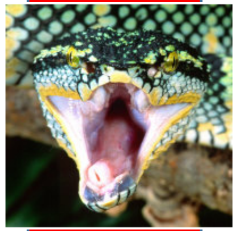
ANSWER: REUSE! OCS XML file is available from training_project



Step 3 - Create App Worker

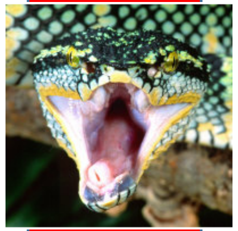
- Via the IDE:
  1) **File → New → Other → ANGRYVIPER → OpenCPI Asset Wizard**
  2) Asset Type: **Worker**
  3) Worker Name: **peak_detector**
  4) Add To Project: **ocpi.training**
  5) Model: **HDL**
  6) Programming Language: **VHDL**
  7) Add To Library: **components**
  8) Component: **peak_detector (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!?)



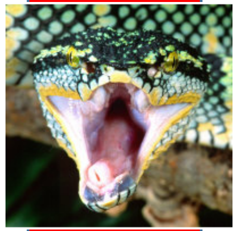
- Because...
  - Although not very exciting, this step proves the skeleton source code is build-able and the build engine is functional
- Build Generated Skeleton Code
  1) In the IDE, add the App Worker to the Project Operations panel
  2) Check the HDL Targets box and **highlight** "xsim" and "zynq", under "xilinx"
  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 – 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

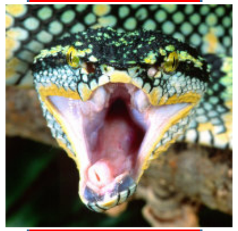
Step 3 – Configure for Version 2 HDL API



- 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='peak_detector-spec' Version=”2”>
    <StreamInterface Name=”in” DataWidth=”32”></StreamInterface>
    <StreamInterface Name=”in” DataWidth=”32” InsertEOM=”1”></StreamInterface>
</HdlWorker>
```

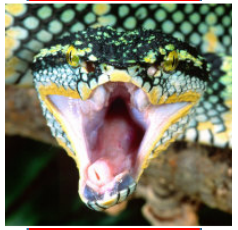
Step 3 – Modify App Worker Source Code



- The skeleton .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

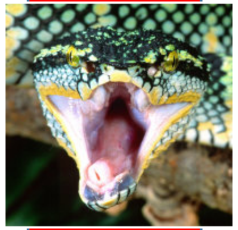
  1) $ **cd /home/training/training_project/components/peak_detector.hdl/**
  2) $ **cp -f /home/training/provided/lab6/peak_detector.vhd .**
- Open the VHDL in a text editor. Starting at the top, modify the source code per the commented instructions (look for “TODO”). A high-level summary of required modifications is provided:
  - **Define the constant**
  - **Make general signal and data port assignments**
  - **Assign the volatile output properties**

Step 4 - Build 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", under "xilinx"
  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

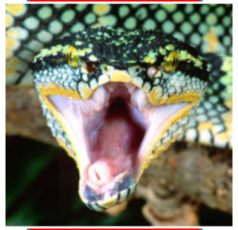


- End of build log should resemble the following, if free of errors:

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

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

make: Entering directory `/home/training/training_project/components/peak_detector.hdl'
Generating the definition file: gen/peak_detector-defs.vhd
Generating the implementation header file: gen/peak_detector-impl.vhd from peak_detector.xml
Generating the implementation skeleton file: gen/peak_detector-skel.vhd
Generating the VHDL constants file for config 0: target-xsim/generics.vhd
Generating the opposite language definition file: gen/peak_detector-defs.vh
Generating the Verilog constants file for config 0: target-xsim/generics.vh
Building the peak_detector worker for xsim (target-xsim/peak_detector) 0:(peak_detector_ocpi_endian=little peak_detector_ocpi_max_bytes_out=8192 peak_detector_ocpi_max_latency_out=256 peak_detector_ocpi_version=2
peak_detector_ocpi_max_opcode_in=0 peak_detector_ocpi_max_opcode_out=0 peak_detector_ocpi_max_bytes_in=8192 peak_detector_ocpi_debug=false)
 Tool "xsim" for target "xsim" succeeded.  0:02.79 at 09:10:26
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 "peak_detector" for target "zynq" 0:(peak_detector_ocpi_endian=little peak_detector_ocpi_max_bytes_out=8192 peak_detector_ocpi_max_latency_out=256 peak_detector_ocpi_version=2
peak_detector_ocpi_max_opcode_in=0 peak_detector_ocpi_max_opcode_out=0 peak_detector_ocpi_max_bytes_in=8192 peak_detector_ocpi_debug=false) target-zynq/peak_detector.edf
 Tool "vivado" for target "zynq" succeeded.  0:37.06 at 09:11:04
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/peak_detector.hdl'
Updating project metadata...
== > Command completed. Rval = 0
```

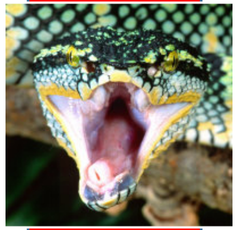
- Review artifacts {worker}.hdl/: "target-xsim/" and "target-zynq/"
  - These directories contain output files from their respective FPGA vendor tools (in this case, simply Xilinx Vivado) in addition to a framework auto-generated file, generics.vhd
  - 'generics.vhd' contains parameter configuration 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)

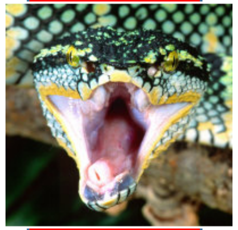
Step 5(a) - Create Unit Test



- **REUSE** from "Work-alike" Simple RCC Worker (Lab3)!
- Located in "peak_detector.test/" directory
- No changes required to Test XML

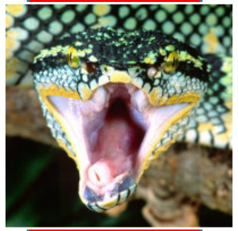
```
<Tests useHDLFileIo='true'>
  <Input Port='in' Script='generate.py 32768'/>
  <Output Port='out' Script='verify.py 32768' View='view.sh'/>
</Tests>
```

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



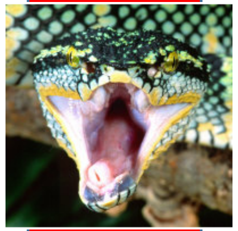
- Build Unit Test Suite for target simulation platform
  1) In the IDE, add the Unit Test to the Project Operations panel
  2) Highlight "xsim" the HDL Platforms panel (HDL Targets box unchecked)
  3) Check "Tests" radio button
  4) Click "gen + build"
  5) Review the Console window messages and address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



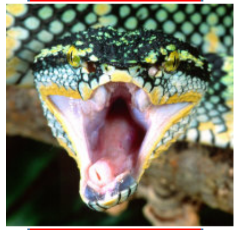
- Observe new artifacts in peak_detector.test/gen/assemblies/peak_detector_0_frw
  - **peak_detector_0_frw.xml** – generated assembly XML (OHAD)
  - gen/ - artifacts generated/used by framework
  - lib/ - artifacts generated/used by framework
  - target-xsim/ - artifacts generated/used by framework and FPGA tools
  - container-**peak_detector_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 launching onto a simulation platform

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

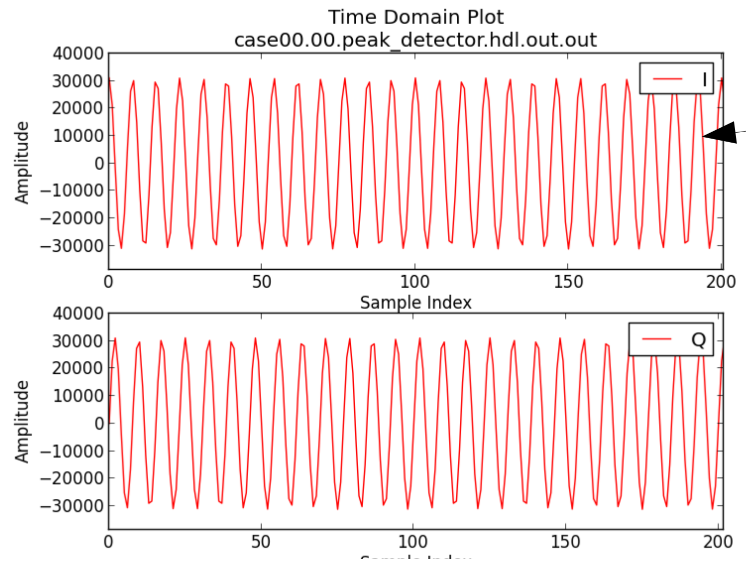


- Via IDE: Run Unit Test Suite for target simulation platform
  1) In the IDE, add the Unit Test (.test) to the Project Operations panel
  2) Highlight "xsim" the <u>HDL Platforms</u> panel (HDL Targets box unchecked)
  3) Check "keep simulations"
  4) Check "run view script"
  5) Click "prep+run+verify" 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" along with final values for the min/max peaks.
- Via Command-line terminal:
  - In a terminal window, execute within the {component}.test/

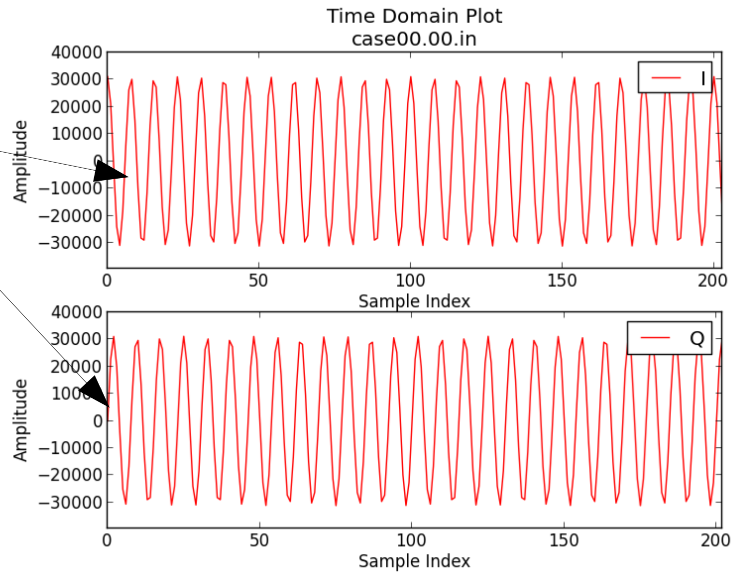
```
$ ocpidev run --mode prep_run_verify --only-platform xsim --keep-simulations --view
```

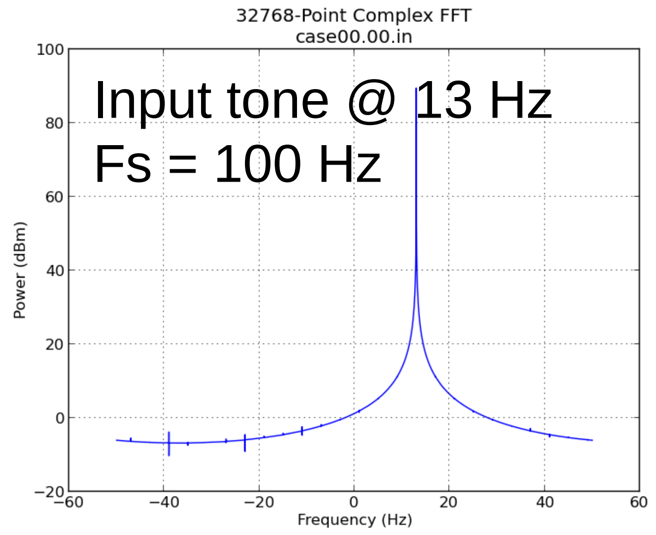


Step 7(a) - Unit Test I/O Plots (Xilinx XSIM)



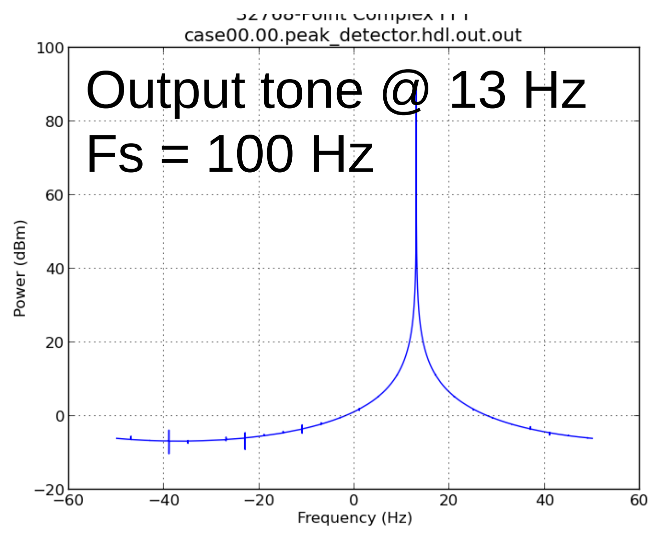
Note:
Zoomed-In





Input tone @ 13 Hz
Fs = 100 Hz

min_peak = -31129
max_peak = 31129



Output tone @ 13 Hz
Fs = 100 Hz

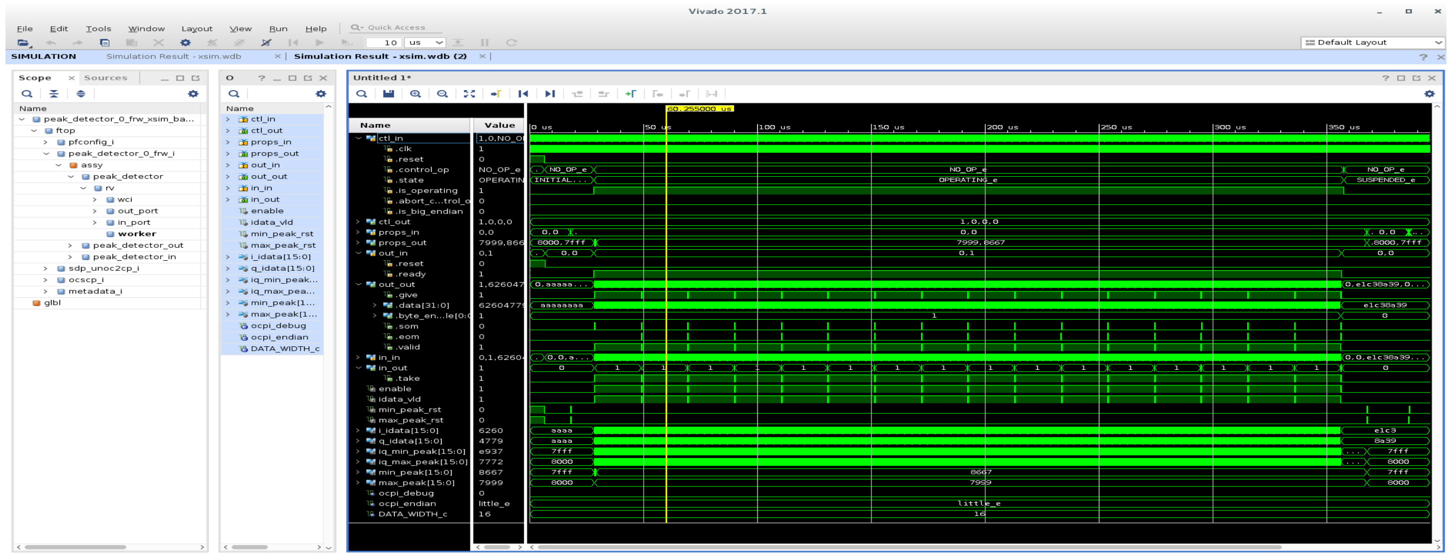
Step 7(a) – View Simulation Waveforms (Xilinx XSIM)

- Must have checked "keep simulations" or ran

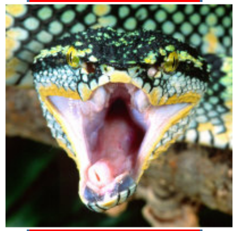
  $ ocpidev run --mode prep_run_verify --only-platform xsim --keep-simulations --view

- In a terminal window, browse to peak_detector.test/ and execute

  $ ocpiview run/xsim/case00.00.peak_detector.hdl.simulation &



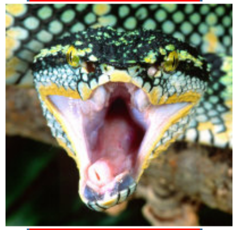
Name	Value
ctl_in	1,0,NO_O
.clk	1
.reset	0
.control_op	NO_OP_e
.state	OPERATIN
.is_operating	1
.abort_c...trol_o	0
.is_big_endian	0
ctl_out	1,0,0,0
props_in	0,0
props_out	7999,866
out_in	0,1
.reset	0
.ready	1
out_out	1,626047
.give	1
.data[31:0]	6260477
.byte_en...le[0:0	1
.som	0
.eom	0
.valid	1
in_in	0,1,6260
in_out	1
.take	1
enable	1
idata_vld	1
min_peak_rst	0
max_peak_rst	0
i_idata[15:0]	6260
q_idata[15:0]	4779
iq_min_peak[15:0]	e937
iq_max_peak[15:0]	7772
min_peak[15:0]	8667
max_peak[15:0]	7999
ocpi_debug	0
ocpi_endian	little_e
DATA_WIDTH_c	16



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

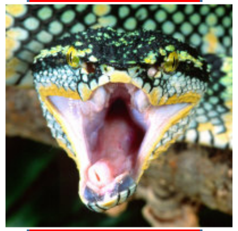
- Employ the framework's Unit Test Suite to generate:
  - OAS (OpenCPI Application Specification) XML file(s)
    - Used by the framework for running the bitstream on hardware platform
    - In this case, the target hardware platform is Ettus E310
  - OHAD (OpenCPI HDL Assembly Description) XML file(s)
    - 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

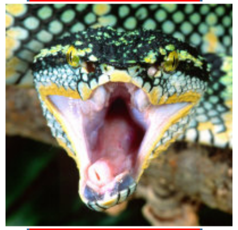


- **REUSE** from 'Work-alike' Simple RCC Worker (Lab3)!
- **REUSE** from Simulation portion of this lab!
- Located in "peak_detector.test/" directory
- No changes required Test XML

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



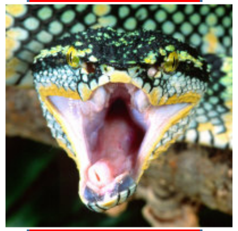
- Build Unit Test Suite for target hardware platform
  1) In the IDE, add the Unit Test to the Project Operations panel
  2) Highlight "e3xx" the HDL Platforms panel (HDL Targets box unchecked)
  3) Check "Tests" radio button
  4) Click "gen+build" to build tests
  5) Review the Console window messages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



Step 6(b) – Review Build Logs (e3xx)

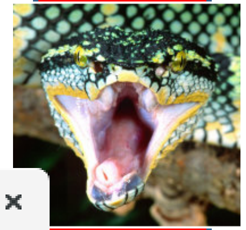
- Below is an example of a successful build
  - (only the end portion is shown)

```
Building container core "peak_detector_0_e3xx_z1_base" for target "zynq" 0:(sdp_width=1 ocpi_debug=false ocpi_endian=little) target-zynq/peak_detector_0_e3xx_base.edf
Generating UUID, artifact xml file and metadata ROM file for container peak_detector_0_e3xx_base (0).
Wrote bram file 'target-zynq/metadatarom.dat' (1876 bytes) from file 'target-zynq/peak_detector_0_e3xx_base-art.xml' (12690 bytes)
 Tool "vivado" for target "zynq" succeeded.  0:45.34 at 12:58:31
Creating link to export worker binary: ../../peak_detector_0/lib/hdl/zynq/peak_detector_0_e3xx_base.edf -> target-zynq/peak_detector_0_e3xx_base.edf
Creating link from ../../peak_detector_0/lib/hdl -> gen/peak_detector_0_e3xx_base.xml to expose the container-peak_detector_0_e3xx_base implementation xml.
Creating link from ../../peak_detector_0/lib/hdl/zynq/peak_detector_0_e3xx_base.vhd -> target-zynq/peak_detector_0_e3xx_base-defs.vhd to expose the definition of worker peak_detector_0_e3xx_base.
Creating link from ../../peak_detector_0/lib/hdl/zynq/peak_detector_0_e3xx_base.v -> target-zynq/peak_detector_0_e3xx_base-defs.vh to expose the other-language stub for worker peak_detector_0_e3xx_base.
For peak_detector_0 on e3xx using config base: creating optimized DCP file using "opt_design". Details in opt.out
Time: 0:38.66 at 12:59:10
 Tool "vivado" for target "zynq" succeeded on stage "opt".
For peak_detector_0 on e3xx using config base: creating placed DCP file using "place_design". Details in place.out
Time: 0:54.32 at 13:00:04
 Tool "vivado" for target "zynq" succeeded on stage "place".
For peak_detector_0 on e3xx using config base: creating routed DCP file using "route_design". Details in route.out
Time: 1:03.33 at 13:01:08
 Tool "vivado" for target "zynq" succeeded on stage "route".
Generating timing report (RPX) for peak_detector_0 on e3xx using base using "report_timing". Details in timing.out
Time: 0:28.55 at 13:01:36
 Tool "vivado" for target "zynq" succeeded on stage "timing".
For peak_detector_0 on e3xx using config base: Generating Xilinx Vivado bitstream file target-zynq/peak_detector_0_e3xx_base.bit. Details in bit.out
Time: 0:41.37 at 13:02:18
 Tool "vivado" for target "zynq" succeeded on stage "bit".
Making compressed bit file: target-zynq/peak_detector_0_e3xx_base.bit.gz from target-zynq/peak_detector_0_e3xx_base.bit and target-zynq/peak_detector_0_e3xx_base-art.xml
```



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

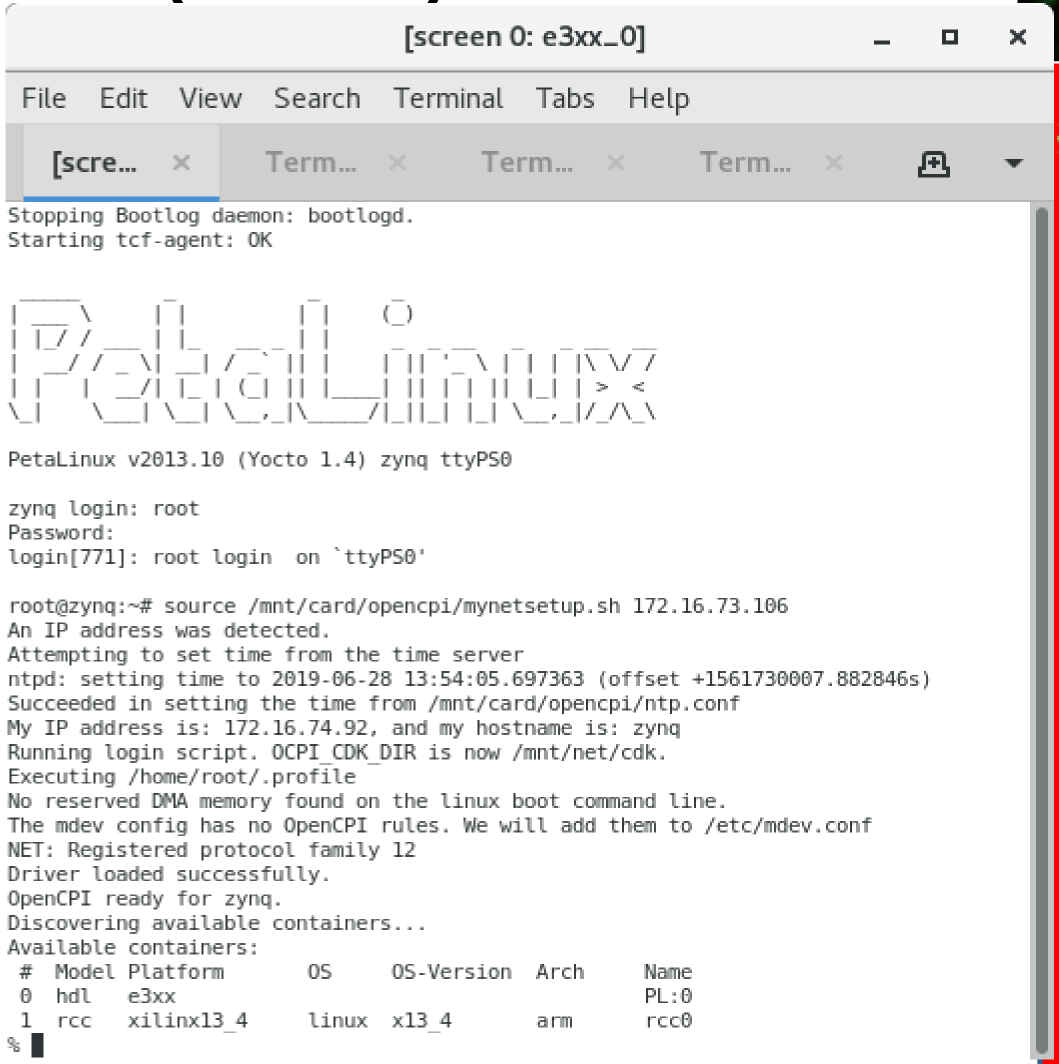
- Observe new artifacts in peak_detector.test/gen/assemblies/**peak_detector_0**/
  - **peak_detector_0.xml** – generated assembly XML (OHAD)
  - gen/ - artifacts generated/used by framework
  - lib/ - artifacts generated/used by framework
  - target-zynq/ - artifacts generated/used by framework and FPGA tools
  - container-**peak_detector_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)

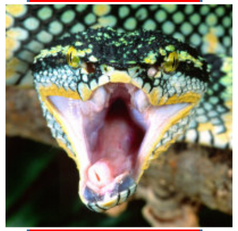
- Setup deployment platform
  1. Connect to serial port via USB on rear of Ettus E310 on Host
     - "screen /dev/e3xx_0 115200"
  2. Boot and login into Petalinux on E310
     - User/Password = root:root
  3. Verify Host and E310 have valid IP addresses
     - For training, they should both be on the 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



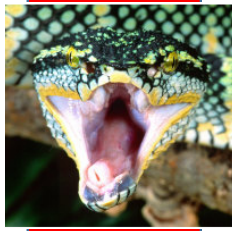
```
Stopping Bootlog daemon: bootlogd.
Starting tcf-agent: OK

 ______     _        _      _
|  ___ \   | |      | |    (_)
| |__/ /   | |_     | |
|  __/ / _ \| __| / _` || |    | || '_ \ | | | |\ \/ /
| |    |  __/| |_ | (_| || |____| || | | || |_| | >  <
\_|     \___| \__| \__,_|\_____/|_||_| |_| \__,_|/_/\_\

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   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

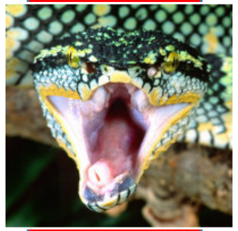
Step 7(b) - Run Unit Test (Ettus E310)



- 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
```

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



- Python script verifies output data from the Unit Test

```
============ Verifying test cases for platform xsim
Performing test cases for ocpi.training.peak_detector on platform xsim.  Functions are:  verify
  Verifying case case00.00 for worker peak_detector.hdl using script on output file: case00.00.peak_detector.hdl.out.out
('File to validate: ', 'case00.00.peak_detector.hdl.out.out')
('uut_min_peak = ', -31129)
('uut_max_peak = ', 31129)
('file_min_peak = ', -31129)
('file_max_peak = ', 31129)
    Verification for port out: PASSED
```

- Perform an md5sum to ensure the output of the XSIM and E310 tests are the same
  peak_detector.test$ md5sum run/*/*.out
  - 747a9fa9fbd73607c474e60782cb4cd3 run/**centos7**/case00.01.peak_detector.rcc.out.out
  - 747a9fa9fbd73607c474e60782cb4cd3 run/**xilinx13_4**/case00.01.peak_detector.rcc.out.out
  - 747a9fa9fbd73607c474e60782cb4cd3 run/**xsim**/case00.00.peak_detector.hdl.out.out
  - 747a9fa9fbd73607c474e60782cb4cd3 run/**e3xx**/case00.00.peak_detector.hdl.out.out