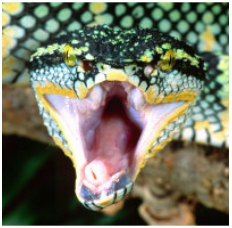


Lab 9: counter.hdl

Debugging an HDL App Worker on a hardware platform

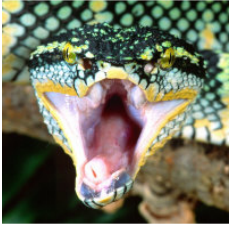
Objectives

- Learn to debug OpenCPI applications
- Observe property values (**ocpiview** and **ocpihdl**)
- Use **ocpi_debug** to enable/disable debugging
- **Step** through an application
- Use ocpihdl to manage the **states/properties** of workers



HDL Application Worker Debugging

1. Build the HDL App Worker for a simulator
2. Build the HDL Assembly for a simulator
3. Test the HDL App Worker and view the results
4. Add debug functionality to the worker
5. Rebuild the HDL App worker for target device(s)
6. Rebuild the HDL Assembly for target platform(s)
7. Debug the HDL App worker using ocpihdl



What's Provided? Everything!

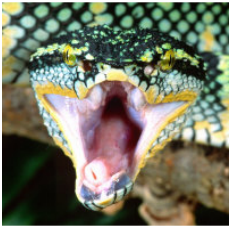
- **counter.hdl/**

- This includes the counter HDL worker
 - Plain version
 - Debug version

- **counter.test/**

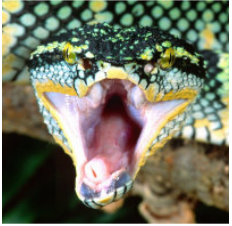
- This includes the Unit Test Description XML for the "counter" component

- **counter-spec.xml**



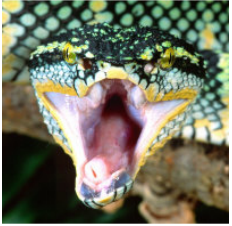
Prerequisites

- Copy "**counter.hdl**/" directory to the components library
 - From: ~/training/provided/lab9/counter.hdl/
 - To: ~/training/training_project/components
- Copy "**counter.test**/" directory to the components library
 - From: ~/training/provided/lab9/counter.test/
 - To: ~/training/training_project/components
- Copy "**counter-spec.xml**" to components/specs
 - From: ~/training/provided/lab9/counter-spec.xml
 - To: ~/training/training_project/components/specs



Counter – HDL App Worker

- To simplify this lab we will use a single "counter" HDL application worker
- The purpose of this worker is to increment a counter vector until reaching a max value
- We will discover that something is wrong with the worker!



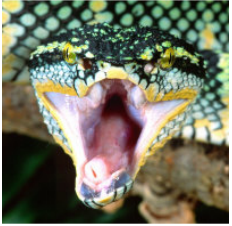
Steps 1 - Build the HDL Worker (Xilinx XSIM)

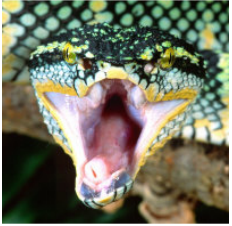


- 1) IDE: "Refresh" the Project Explorer panel
- 2) IDE: "Refresh" the OpenCPI Projects panel
- 3) In the IDE, add the App Worker to the Project Operations panel
- 4) Check the "HDL Targets" box and highlight "xsim" under "xilinx"
- 5) Check "Assets" Radio button
- 6) Click "Build"
- 7) Review the Console window messages to ensure this step is error free

Steps 2 - Build the Unit Test (Xilinx XSIM)

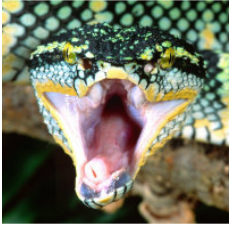
- 1) In the IDE, remove the App Worker from the Project Operations panel
- 2) Add the Unit Test to the Project Operations panel
- 3) Highlight "xsim" the HDL Platforms panel
 "HDL Targets" box unchecked
- 4) Check "Tests" Radio button
- 5) Click "gen + build"
- 6) Review the Console window messages and address any errors





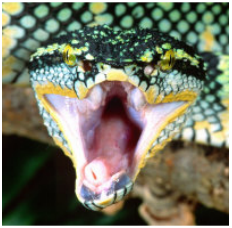
Step 3(a) – Run Unit Test on (Xilinx XSIM)

- Run 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) Click "prep + run" to Run Tests
 - 4) Review the Console window messages and address any errors
- Validate the Output:
 - Open the log file (run/xsim/case00.00.counter-1.hdl.log) and confirm the property value:
 - counter.counter = "10"
 - The "max" property is currently set to "10" by counter-test.xml
 - The "counter" property reached its maximum value of "10" as expected



Step 3(a) – Run Unit Test: Change count

- Edit counter.test/counter-test.xml to test the value "9" for the "max" property
- Regenerate the application XML:
 - 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
 - 6) Review gen/applications/*.xml(s) to ensure the max value has been updated



Step 3(a) – Run Unit Test: Change count

- IDE: Rerun the Unit Test with the new maximum value
 - 1) Check "keep simulations" so that we can see the waveform output later
 - 2) Click "prep + run" to run the tests
- OR in a Terminal window, browse to counter.test/

```
$ ocpidev run test counter.test -d /home/training/training_project -l components --mode prep_run --only-platform xsim --only-platform centos7 --keep-simulations
```

 - “--keep-simulations” so that we can view the simulation output
- Open the log file (run/xsim/case00.00.counter-1.hdl.log) and review the property value
 - Output:
 - counter.counter = "10"
 - The counter PASSED the maximum value of "9"! HOW IS THIS POSSIBLE?

Step 3(b) - Viewing Simulation Signals



- To view the simulation results, in a terminal window
`$ ocpiview run/xsim/case00.00.counter.hdl.simulation &`
- Navigate to the counter object (add the “worker” signals to wave window)

The screenshot displays two panels from the OpenCPI GUI. The left panel, titled 'Scope', shows a hierarchical tree of simulation objects. The right panel, titled 'Objects', shows a list of selected signals and their current values.

Name	Design
counter_0_xsim_base	counter_0_xsim_base
ftop	counter_0_xsim_base
pfconfig_i	counter_0_xsim_base
counter_0_i	counter_0_xsim_base
assy	counter_0_xsim_base
uut_counter	counter_0_xsim_base
rv	counter_0_xsim_base
wci	counter_0_xsim_base
worker	counter_0_xsim_base
sdp_unoc2cp_i	counter_0_xsim_base

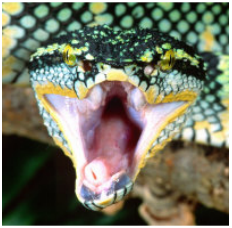
Name	Value
ctl_in	
ctl_out	
props_in	
props_out	
enable	
counter[15:0]	
finished	
step_counter	
ocpi_debug	
ocpi_endian	

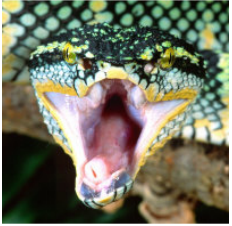
Step 3(b) – Observe the Counter Signal

- Note the change in the counter value



- Counting by two, not one.

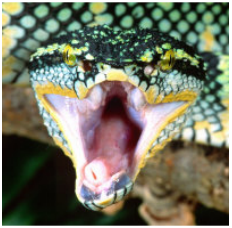




Step 4 - Adding Debugging Functionality

- In counter.hdl:


```
$ cp counter-debug.vhd counter.vhd  
$ cp counter-debug.xml counter.xml
```
- Observe what is done to add debugging/stepping functionality
 - counter.xml
 - New "step" property
 - "max" property is extended from OCS so that we can read it using ocpihdl
 - counter.vhd
 - When the "ocpi_debug" parameter is true, "enable" depends on "step"

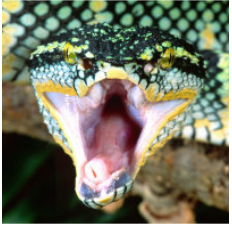


Step 4 - Adding Debugging Functionality (cont.)

- Enables the counting operation based on **ocpi_debug** flag
- When the **step** property is written as true, perform a single counter increment (single step)

```
-- If we ARE debugging, do nothing until the step property is written as true.  
-- Then, step (increment counter) and wait for another step_written pulse  
debug_gen : if its(ocpi_debug) generate  
    step_counter <= '1' when (its(props_in.step) and (props_in.step_written = '1')) else '0';  
    enable <= '1' when (its(ctl_in.is_operating) and step_counter = '1') else '0';
```

- Otherwise, proceed normally

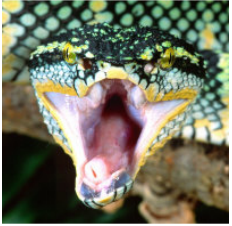


Step 4 - Adding Debugging Functionality (cont.)

- Note that the counter.hdl supports two build configurations, as defined in the counter-build.xml

```
<build>
  <configuration id='0'>
    <parameter name='ocpi_debug' value='false' />
  </configuration>
  <configuration id='1'>
    <parameter name='ocpi_debug' value='true' />
  </configuration>
</build>
```

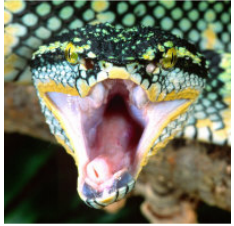
- The ocpi_debug parameter is provided by the framework, which allows the worker to build separate implementations, both with and without debugging behavior



Step 5-6 – Setup Unit Test for different Worker configurations

- Edit the counter-test.xml to set ocpi_debug to false and true
`<Property Name='ocpi_debug' Values='false,true'></Property>`
- Use the IDE to "clean"
 - counter.hdl
 - counter.test

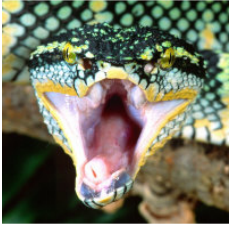
Step 5-6 - Build the HDL Worker and Assembly for HW (cont)



- Build HDL App Worker for Zynq Targets
 - 1) In the IDE, add the App Worker to the Project Operations panel
 - 2) Check the HDL Targets box and highlight "zynq"
 - 3) Check "Assets" Radio Button
 - 4) Click "Build"
 - 5) Review the Console window messages and address any errors
- Build Unit Test Suite for target hardware platform
 - 1) In the IDE, remove the App Worker from the Project Operations panel
 - 2) Add the Unit Test to the Project Operations panel
 - 3) Highlight "e3xx" 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
- NOTE: The build process takes 5-10 mins to complete.

Step 7(a-d) - Debug the HDL App worker using ocpihdl

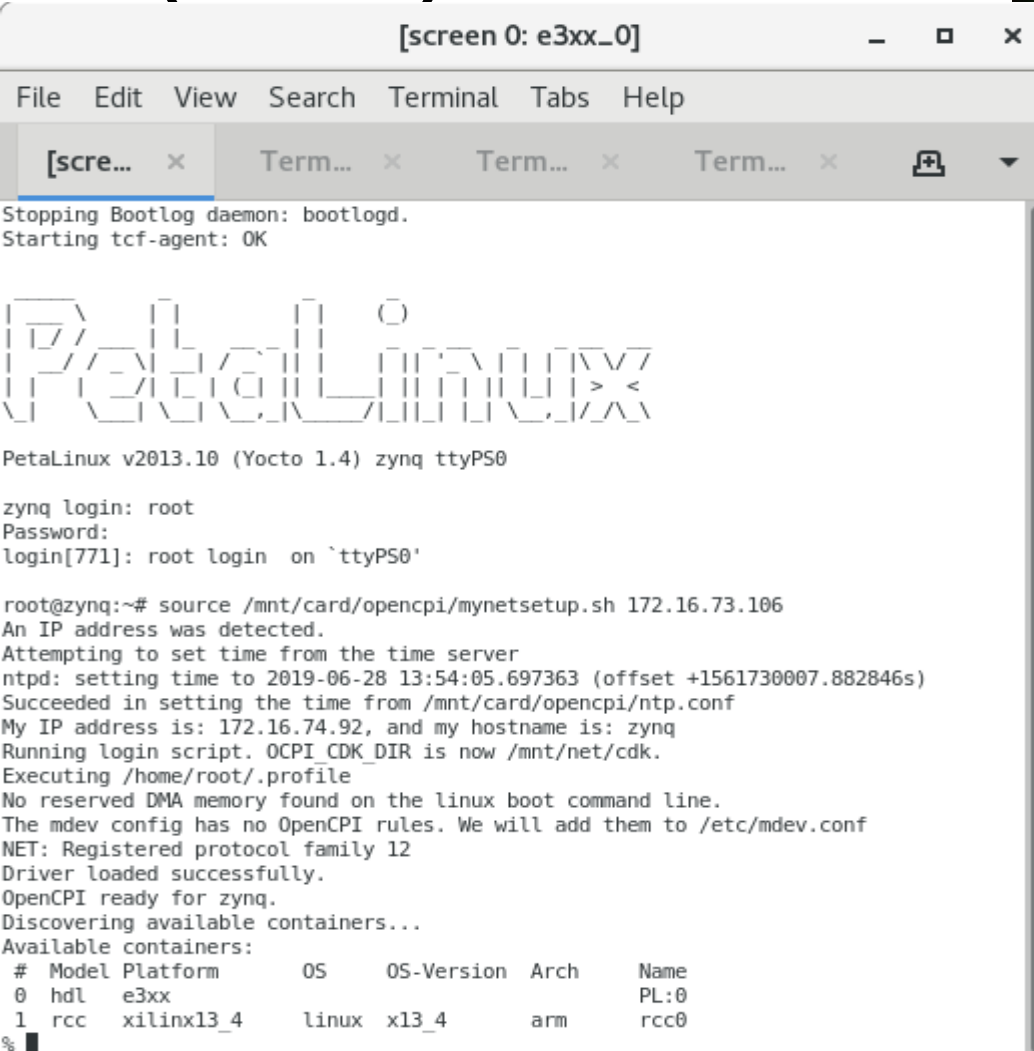
- The remainder of the lab will involve the following steps:
 - a) Execute on the target HW platform
 - b) Confirm Debug Mode
 - c) Collect HDL Worker/Assembly Information
 - d) Debug the HDL App Worker using ocpihdl



Step 7(a) – 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

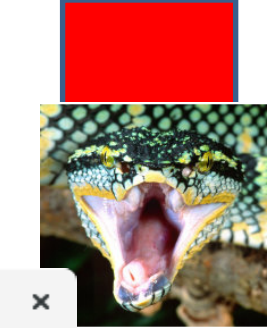


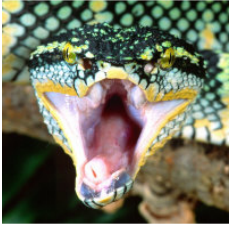
```
[screen 0: e3xx_0]
File Edit View Search Terminal Tabs Help
[scre... x Term... x Term... x Term... x +
Stopping Bootlog daemon: bootlogd.
Starting tcf-agent: OK

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)

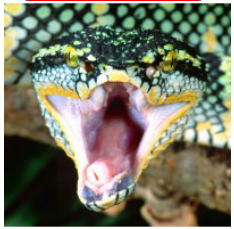
- IDE approach to running unit tests on remote platforms:
 - 1) Add counter.test to the Project Operations Panel
 - 2) Check "Tests"
 - 3) Select "remotes" radio button
 - 4) Click "+remotes"
 - 5) Change remote variable text to use E310's IP and point to the training project
{IP of Ettus E310}=root=root=/mnt/training_project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

Step 7(b) - Confirm Debug Mode

- On the Ettus E310, the application is now running
- We have the application "hanging"
 - It is in debug mode and waiting for our input
 - Lets see what the state of the worker is by going to the logs.
 - Open the log file (run/e3xx/case00.00.counter-1.hdl.log)

```
Dump of all initial property values:
Property 0: counter.counter = "0"
Property 1: counter.max = "9" (cached)
Property 2: counter.ocpi_debug = "true" (parameter)
Property 3: counter.ocpi_endian = "little" (parameter)
Property 4: counter.ocpi_version = "0" (parameter, hidden, worker)
Property 5: counter.step = "false" (cached, debug)
Application started/running
Waiting for application to finish (no time limit)
% █
```

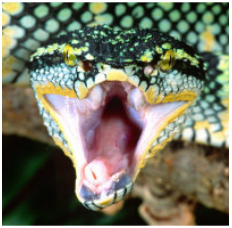
- On the Ettus E310, we can now use **ocpihdl** to control the worker



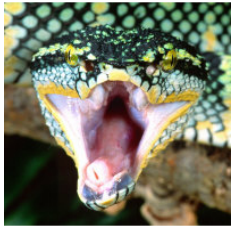
For reference (Output of: ocpihdl)

Major commands/modes:

```
search [-i <interface>]      # search for OpenCPI HDL devices, limit ethernet to <interface>
emulate [-i <interface>]     # emulate an HDL device on ethernet, on first or specified interface
ethers                       # list available ethernet interfaces
probe <hdl-dev>              # see if a specific HDL device is available
get [<instance> [<property>]] # get info from bitstream
set <instance> <property> <value> # set property value for worker instance
control <instance> <operation> # perform reset, unreset, or control operation
status <instance>           # show status of worker/instance
testdma                     # test for DMA memory setup
admin <hdl-dev>             # dump admin information (reading only) for <hdl-device>
wadmin <hdl-dev> <offset> <value> # write admin word <value> for <hdl-device> at <offset>
radmin <hdl-dev> <offset>      # read admin word for <hdl-device> at <offset>
settime <hdl-dev>           # set the GPS time of the device to system time
deltatime <hdl-dev>         # measure round trip and difference between host and device
dump <hdl-dev>              # dump all state/status of <platform> including all workers
reset <hdl-dev>             # reset platform
flash <hdl-dev>             # flash load platform
```

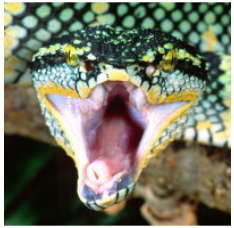


For reference (Output of: ocpihdl)



```
bram <infile> <outfile>          # create a BRAM file from in input file
unbram <infile> <outfile>         # recreate the original file from a BRAM file
uuid -p <platform> -c <part> <outputfilename>
wclear <hdl-dev> <worker>         # clear worker status errors
wdump <hdl-dev> <worker>          # dump worker's control plane registers
wop <hdl-dev> <worker> <op>       # perform control operation on worker
    ops are: initialize, start, stop, release, after, before
wread <hdl-dev> <worker> <offset>[/size] # perform config space read of size bytes at offset, default 4
wreset <hdl-dev> <worker>          # assert reset for worker
wunreset <hdl-dev> <worker>        # deassert (enable) worker
wwctl <hdl-dev> <worker> <val>     # write worker control register
wwpage <hdl-dev> <worker> <val>    # write worker window register
wwrite <hdl-dev> <worker> <offset>[/size] <value> # perform config space write of size bytes at offset
                                     # generate UUID verilog file
load <hdl-dev> <file>              # load bitstream from file
unload <hdl-dev>                   # revert device to unloaded state: no bitstream
getxml <hdl-dev> <file>            # Extract the xml metadata from the device into the file
```

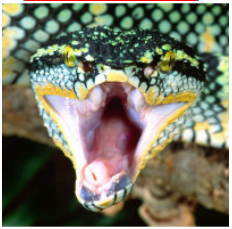

For reference (Output of: ocpihdl)



```
simulate                                # run simulator inside created sim: device
Options: (values are either directly after the letter or in the next argument)
-l <level>                             # set log levels
-i <interface>                         # set ethernet interface to use
-d <hdl-device>                       # identify a specific hdl device to use
-p <hdl-platform>                    # specify a particular hdl platform (e.g. ml605)
-c <hdl-part>                        # specify a particular part/chip (e.g. xc6vlx240t
-s <spin-clocks>                     # clocks to credit/run the sim between control messages
-t <sleep-usecs>                     # delay time letting sim run between credits
-T <sim-time>                        # total simulation time before terminating
-D                                    # turn off simulation dumping
-e <sim-executable>                  # simulator executable "bitstream" file
-A                                    # make the simulator publically available on the LAN
-v                                    # be verbose
-x                                    # print numeric values in hex rather than decimal

<worker> can be multiple workers such as 1,2,3,4,5. No ranges.
<hdl-dev> examples: 3                # PCI device 3 (i.e. 0000:03:00.0)
                                0000:04:00.0    # PCI device 0000:04:00.0
                                PCI:0001:05:04.2 # fully specified PCI device
                                a0:00:b0:34:55:67 # ethernet MAC address on first up+connected interface
                                eth0/a0:00:b0:34:55:67 # ethernet address on a specific interface
                                Ether:eth1/a0:00:b0:34:55:67 # fully specified Ethernet-based device
```

Step 7(c) - Collect HDL Worker/Assembly Information



- View the currently loaded workers and their indices

\$ ocpihdl get

```
% ocpihdl get
```

```
HDL Device: 'PL:0' is platform 'e3xx' part 'xc7z020' and UUID 'b32c808c-a267-11e9-9b24-038f0fc4dbc4'
```

```
Platform configuration workers are:
```

```
Instance p/e3xx of io worker e3xx (spec ocpicore.platform) with index 0
```

```
Instance p/time_server of io worker time_server (spec ocpicore.devices.time_server) with index 1
```

```
Container workers are:
```

```
Instance c/ocscp of normal worker ocscp (spec ocpicore.ocscp)
```

```
Instance c/unoc_term0_0 of io worker sdp_term (spec ocpicore.devices.sdp_term)
```

```
Instance c/unoc_term1_0 of io worker sdp_term (spec ocpicore.devices.sdp_term)
```

```
Instance c/unoc_term2_0 of io worker sdp_term (spec ocpicore.devices.sdp_term)
```

```
Instance c/unoc_term3_0 of io worker sdp_term (spec ocpicore.devices.sdp_term)
```

```
Instance c/metadata of normal worker metadata (spec ocpicore.metadata)
```

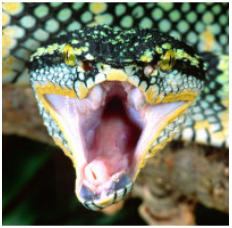
```
Application workers are:
```

```
Instance a/uut_counter of normal worker counter-1 (spec ocpitraining.counter) with index 2
```

```
% █
```

- Use this index (or "counter-1") to identify the worker from now on

Step 7(d) - Debug the HDL App Worker using ocpihdl



- ocpihdl get -v 2 ("-v" to list property values)

- Note property values:

- counter: 0
- max: 9
- ocp_debug: true
- step=false

```
counter: 0
max: 9
ocpi_debug: true
ocpi_endian: little
step: false
```

- Also try: ocpihdl get -v counter-1
- ocpihdl set 2 step true
- ocpihdl get -v 2
 - Note : counter : 2

```
counter: 2
max: 9
ocpi_debug: true
ocpi_endian: little
step: true
```

Step 7(d) - Debug the HDL App Worker using ocpihdl (cont.)



- Continue setting "step" to true
 - Note how the value of counter changes
 - When counter reaches "max", step one last time and the application will complete.
 - You will see the following output back on the Development Host

```
make: Leaving directory `/home/training/training_project/components/counter.test'  
== > Command completed. Rval = 0
```

- Open the log file to see the final output
(run/e3xx/case00.01.counter-1.hdl.log)

```
Application finished  
Dump of all final property values:  
Property 0: counter.counter = "10"
```

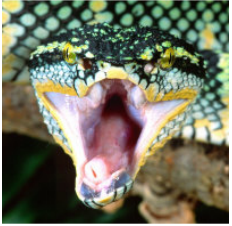
Found the Bug



- By now you have discovered the bug using one or more of the following methods:
 - Viewing the log files and observing the output for different max values
 - Viewing the simulated counter signals using **ocpiview**
 - Using debug properties and **ocpihdl** to step through the application
- The bug:
 - We are incrementing the counter by 2 each step
 - You can fix the bug in counter.vhd, disable debugging, rebuild, and rerun
- This was a very simple example, but **ocpihdl** can be very helpful for debugging and controlling OpenCPI applications

One Alternative

- Use a *property* (instead of a parameter) to toggle debugging
 - Instead of (or in conjunction with) `ocpi_debug`
 - This allows you to turn off debugging using run-time `ocpihdl` commands
- If time permits:
 - Replace `ocpi_debug` with a property (*not* parameter)
 - Can no longer use the "generate" statements!
 - Rebuild
 - Run
 - Step a few times
 - Set the debug flag to false
 - Let the application complete



BACKUP SLIDES



Using ocpihdl on a simulator

NOT RECOMMENDED

- **MUST** specify the HDL device with every ocpihdl command
 - For example: `$ ocpihdl status -d sim:... -v status 2`
- To make this easier, save the string:
 - `$ ocpihdl search (to identify the device)`
 - `$ SIM="-d sim:127.0.0.1:49876"`
 - This will be different every time a simulator starts
 - Then for example: `$ ocpihdl status $SIM -v status 2`

