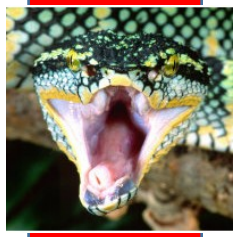
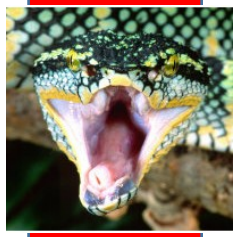


Lab 9:
Tools and Debugging for HDL Workers

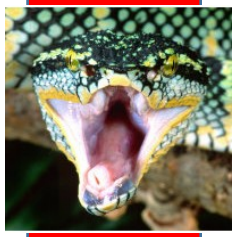
counter.hdl & counter.test

Objectives



- Learn to debug ANGRYVIPER 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

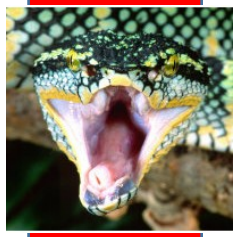


1. Build the HDL App Worker for a simulator
2. Build the HDL Assembly for a simulator
3. Test the HDL App Worker and view the results
4. Add debug functionality to the worker
5. Rebuild the HDL App worker for target device(s)
6. Rebuild the HDL Assembly for target platform(s)
7. Debug the HDL App worker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

What's Provided? Everything!

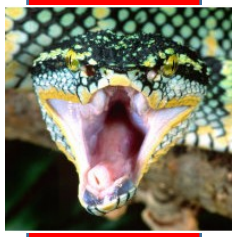
- **counter.hdl**
  - This includes the counter HDL worker
    - Plain version
    - Debug version
- **counter.test**
  - This includes the Unit Test Description XML for the "counter" component
- counter-spec.xml

Prerequisites



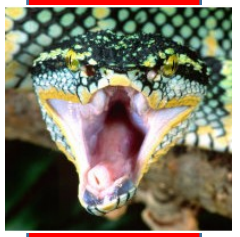
- Copy "**counter.hdl**/" directory to the components library
  - From: /home/training/provided/lab9/counter.hdl/
  - To: /home/training/training_project/components
- Copy "**counter.test**/" directory to the components library
  - From: /home/training/provided/lab9/counter.test/
  - To: /home/training/training_project/components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

Counter – HDL App Worker



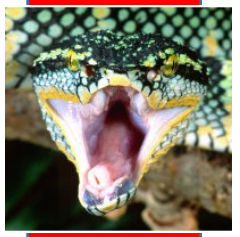
- To simplify this lab we will use a single “counter” HDL application worker
- This worker increments a counter vector repeatedly
- We will discover that something is wrong with the worker!

Steps 1 - Build the HDL Worker (Xilinx XSIM)



1) IDE: “Refresh” the Project Explorer panel
2) IDE: “Refresh” the OpenCPI Projects panel
3) Use the IDE to “Add” the App Worker to the Project Operations panel
4) Check the HDL Targets box and highlight “xsim” under “xilinx”
5) Click “Build Assets”
6) Review the Console window messages to ensure this step is error free

Steps 2 - Build the Unit Test (Xilinx XSIM)



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) **Click** "Build Tests"
4) 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) 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) Click “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:
    - Property 0: 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

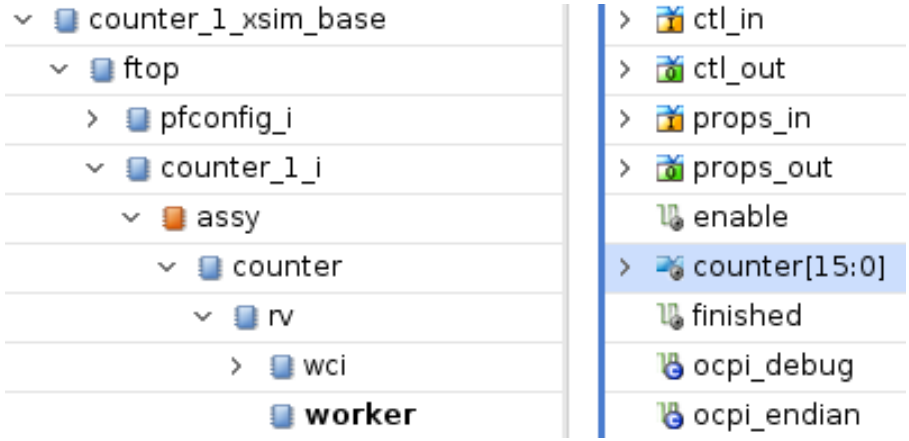
- Edit counter.test/counter-test.xml to test the value ‘9’ for the ‘max’ property
- Regenerate the application XML:
  1) Use the IDE to “Add” the Unit Test to the Project Operations panel
  2) Highlight “xsim” the HDL Platforms panel (HDL Targets box unchecked)
  3) Click “Build Tests”
  4) Review the Console window messages and address any errors
  5) Review gen/applications/*.xml(s) to ensure the max value has been updated
- Rerun the Unit Test with the new maximum value
  - In a terminal window, browse to counter.test/

  $ make run OnlyPlatforms=xsim KeepSimulations=1 TestVerbose=1

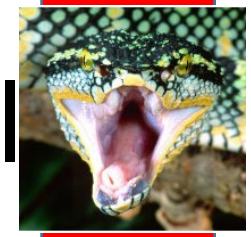
  - KeepSimulations=1 so that we can view the simulation output
- Output:
  - Property 0: counter.counter = "10"
  - The counter PASSED the maximum value of ‘9’!?

Step 3(b) - Viewing Simulation Signals

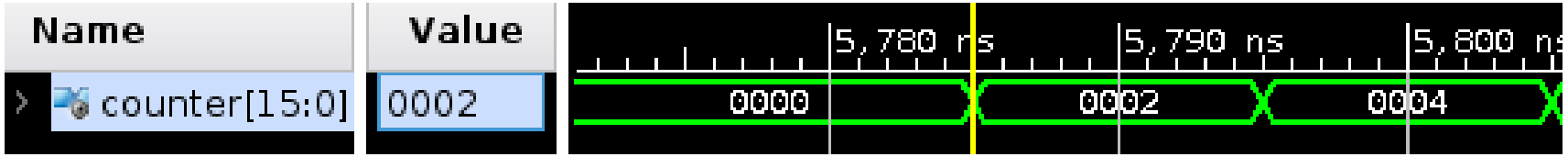
- To view the simulation results:
  - $ ocpiview run/xsim/case00.00.counter-1.hdl.simulation &
- Navigate to the counter object (add the signal)



Step 3(b) – Observe the Counter Signal



- Note the change in the counter value
- Feel free to add other signals and observe their alignment

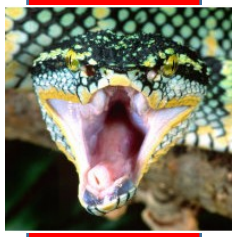


Step 4 - Adding Debugging Functionality

- 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
  - counter.vhd
    - When the “ocpi_debug” parameter is true, “enable” depends on “step”

Step 4 - Adding Debugging Functionality (cont.)



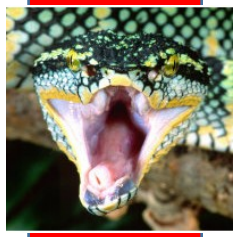
- Enables the counting operation based on **ocpi_debug** flag

- 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 the following in counter.hdl/Makefile:

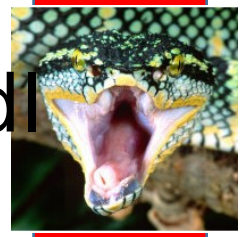
```
ParamValues_ocpi_debug=false/true
```

  - This makes sure the worker (counter.hdl) builds for both
- Navigate to counter.test
  - In counter-test.xml, set ocpi_debug to true
- Use the IDE to “clean”
  - counter.hdl
  - counter.test

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

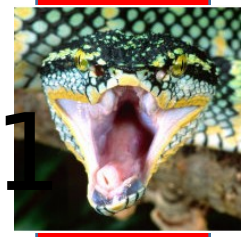
- Build HDL App Worker for Zynq Targets
  1) Use the IDE to “Add” the App Worker to the Project Operations panel
  2) Check the HDL Targets box and highlight “xsim” and “zynq”
  3) Click “Build Assets”
  4) Review the Console window messages and address any errors
- Build Unit Test Suite for target hardware 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) **Click** “Build Tests”
  4) 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 Matchstiq-Z1
  b) Confirm Debug Mode
  c) Collect HDL Worker/Assembly Information
  d) Debug the HDL App Worker using ocpihdl

Step 7(a) - Execute on the Matchstiq-Z1



- Setup deployment platform
  1. Connect to serial port via USB on rear of Matchstiq-Z1 using host
     - ‘screen /dev/ttyUSB0 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

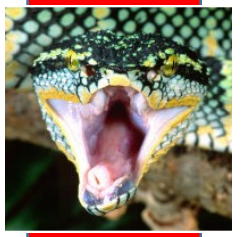
Step 7(b) - Run Unit Test (Matchstiq-Z1)

- Prior to launching the IDE, OCPI_REMOTE_TEST_SYSTEMS must be set

  $ **export OCPI_REMOTE_TEST_SYSTEMS=<IP of Matchsitq-Z1>=root=root=/mnt/training_project**

- Run Unit Test Suite for target hardware platform
  - Use the IDE to "**Add**" the Unit Test to the Project Operations panel
  - **Highlight** "matchstiq_z1" the HDL Platforms panel (HDL Targets box unchecked)
  - **Click** "Run Tests"
  - Review the Console window messages and address any errors

Step 7(b) - Confirm Debug Mode



- On the Matchstiq-Z1, the application is now running
- We have the application “hanging”
  - It is in debug mode and waiting for our input
  - You should see the following output on the Development Host:

```
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.step = "false" (cached)
Application started/running
Waiting for application to be finished (no timeout)
```

- On the Matchstiq-Z1 we can now use ocpihdl commands 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 syste 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
```

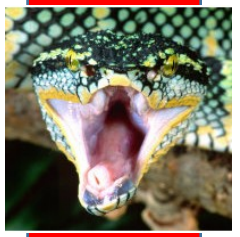
For reference (Output of: ocpihdl)

```
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 (default 4) at offset
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 (default 4) 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 'matchstiq_z1' part 'xc7z020' and UUID '35b3d880-2c30-1
1e7-9c2c-e36cff07cd89'
Platform configuration workers are:
  Instance p/matchstiq_z1 of io worker matchstiq_z1 (spec ocpi.platform) with index 0
  Instance p/time_server of io worker time_server (spec ocpi.devices.time_server) with
 index 1
Container workers are:
  Instance c/ocscp of normal worker ocscp (spec ocpi.ocscp)
  Instance c/unoc_term0_0 of io worker sdp_term (spec ocpi.devices.sdp_term)
  Instance c/unoc_term1_0 of io worker sdp_term (spec ocpi.devices.sdp_term)
  Instance c/unoc_term2_0 of io worker sdp_term (spec ocpi.devices.sdp_term)
  Instance c/unoc_term3_0 of io worker sdp_term (spec ocpi.devices.sdp_term)
  Instance c/metadata of normal worker metadata (spec ocpi.metadata)
Application workers are:
  Instance a/counter of normal worker counter-1 (spec local.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, ocpi_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

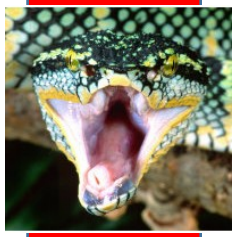
```
counter: 2
    max: 9
ocpi_debug: true
ocpi_endian: little
    step: true
```

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

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

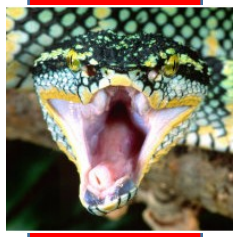
Found the Bug



- 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 counter signal 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

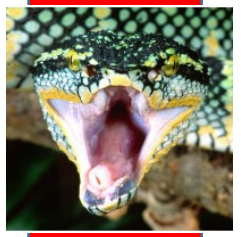
- This was a very simple example, but **ocpihdl** can be very helpful for debugging and controlling OpenCPI applications

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