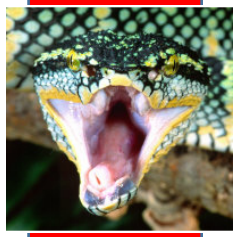
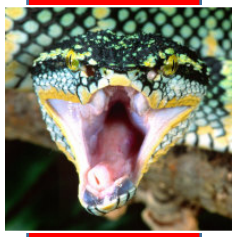


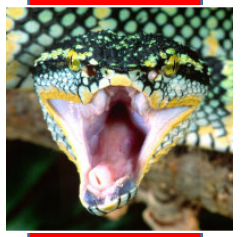
Automated Unit Test Suite (Details)

More Details



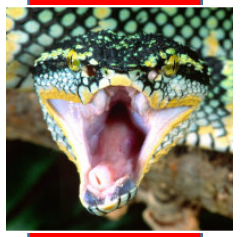
- File I/O details and options
- Manipulating buffer sizes
  - Depends on the protocols you use
- Understanding the *two* HDL assemblies used for testing
  - When to use each one (simulators vs. hardware)
  - Configuration options
- Debugging your failed tests
  - “Where did my overnight simulation go?”
- More implementation details

file_read.rcc/.hdl Basics



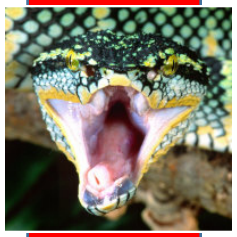
- Implements the ocpi.core/components/specs/file_read-spec.xml
- No Protocol (opcode default is zero)
- ***fileName*** path limit is 1024 char
- Two modes of operation
  - Data Streaming – Contents of file simply copied to payload of a stream of messages
    - Fixed number of bytes of data ***messageSize*** (upcoming slides) defaults to 4KB
    - All with the same opcode
  - Messaging – Contents of file are interpreted as a sequence of defined messages
    - 8-byte header precedes payload data of each message
      - Four byte length (in bytes, little-endian), one byte opcode, three padding bytes
    - ***MessagesInFile***="true" (default "false") and **overrides *messageSize***
- ***repeat*** default is false, restart reading at the beginning of file
- Produces a ZLM after last data is read

file_write.rcc/.hdl Basics



- Implements the ocpi.core/components/specs/file_write-spec.xml
- No Protocol (opcode default is zero)
- ***fileName*** path limit is 1024 char
- Two modes of operation
  - Data Streaming – Contents of file becomes the payloads of a stream of messages, no message lengths or opcodes are recorded.
  - Messaging – Contents of the output file is written as a sequence of defined messages
    - 8-byte header precedes payload data of each message
      - Four byte length (in bytes, little-endian), one byte opcode, three padding bytes
    - ***messagesInFile***=”true” (default “false”)
- “Done” upon detection of a ZLM after last data is read

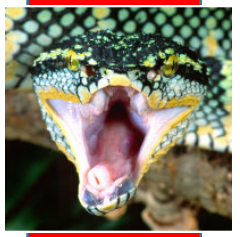
File Read/Write's *messagesInFile*



- If more than one opcode is needed, use messagesInFile
- Input data needs to have:
  - Length of message (in Bytes)
  - Opcode Number to transmit
  - Message (if non-zero length)
- Example with 4 Operations:

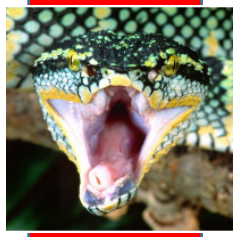
2	0	short scalar mesg.	4	1	short sequence(0) mesg.	short sequence(1) mesg.	4	2	long scalar mesg.	1	3	char scalar mesg.

File Read *messageSize*



- In general, bigger the buffer, higher the throughput
- Determine a *messageSize* that best utilizes the protocol buffer size
  - Protocol buffer size is the largest of all the operations in a protocol
  - This is needed because file_read doesn't "know" the protocol within
- How to calculate the protocol buffer size?
  - Protocol buffer size = max operation size × *length* of the max operation size
    - max operation size - the operation in the protocol with largest number of bytes
    - *length* of the max operation size - either the StringLength, SequenceLength, ArrayDimensions, or size of the type

Protocol Buffer Size

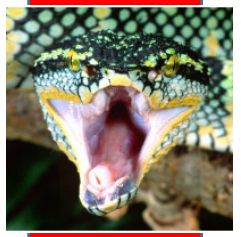


protocol buffer size = max operation size × length of the max operation size

Example 1

```
<Protocol>
  <Operation Name="demo">
    <Argument Name="data" Type="Struct" SequenceLength="2048">
      <Member Name="I" Type="Short"/>
      <Member Name="Q" Type="Short"/>
    </Argument>
  </Operation>
</Protocol>
```

Protocol Buffer Size



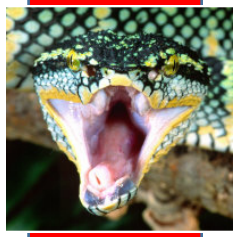
protocol buffer size = max operation size × length of the max operation size

Example 1

```
<Protocol>
   <Operation Name="demo">
      <Argument Name="data" Type="Struct" SequenceLength="2048">
        <Member Name="I" Type="Short"/>
        <Member Name="Q" Type="Short"/>
      </Argument>
    </Operation>
 </Protocol>
```

Solution: Protocol Buffer Size is 4 Bytes x 2K = **8K Bytes**

Protocol Buffer Size

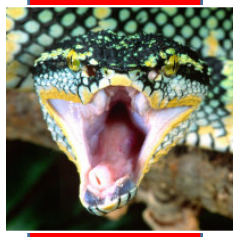


protocol buffer size = max operation size × length of the max operation size

Example 2

```
<Protocol>
   <Operation Name=“demo”>
      <Argument Name=“data” Type=“Short”></Argument>
    </Operation>
 </Protocol>
```

Protocol Buffer Size



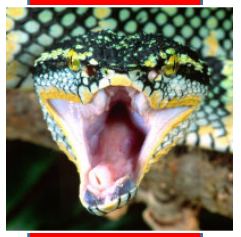
protocol buffer size = max operation size × length of the max operation size

Example 2

```
<Protocol>
   <Operation Name="demo">
      <Argument Name="data" Type="Short"></Argument>
    </Operation>
 </Protocol>
```

Solution: Protocol Buffer Size is 2 Bytes x 1 Scalar = **2 Bytes**

Protocol Buffer Size

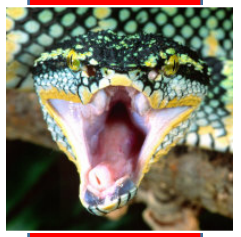


protocol buffer size = max operation size × length of the max operation size

Example 2

```
<Protocol>
  <Operation Name="demo1">
    <Argument Name="data" Type="longlong" ArrayDimensions="2048"> </Argument>
 </Operation>
 <Operation Name="demo2">
    <Argument Name="data" Type="Struct" SequenceLength="2048">
<Member Name="I" Type="Short"/>
<Member Name="Q" Type="Short"/>
    </Argument>
  </Operation>
</Protocol>
```

Protocol Buffer Size



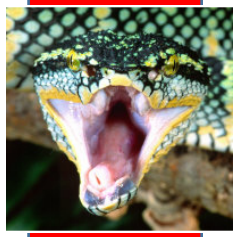
protocol buffer size = max operation size × length of the max operation size

Example 2

```
<Protocol>
  <Operation Name=“demo1”>
    <Argument Name=“data” Type=“longlong” ArrayDimensions=“2048”> </Argument>
 </Operation>
 <Operation Name=“demo2”>
    <Argument Name=“data” Type=“Struct” SequenceLength=“2048”>
<Member Name=“I” Type=“Short”/>
<Member Name=“Q” Type=“Short”/>
    </Argument>
  </Operation>
</Protocol>
```

Solution: Protocol Buffer Size is 8B x 2K = **16KB**, 4B x 2K = 8KB

Two HDL Test Assemblies

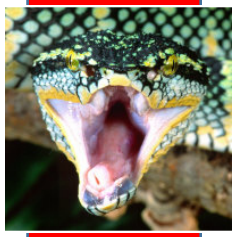


- Both built around “core” of

  (data in) ⇨ metadata_stressor ⇨ UUT ⇨ backpressure ⇨ (data out)

- {component}.test/gen/assemblies/<component>_0/
  - Uses file I/O in an RCC worker
    - (data in) and (data out) above are the assembly’s external ports
  - Can be built for *any* HDL platform
    - Including simulators (“co-simulation”)
      - Uses the simulatable portions of the data plane
        - No DMA Engine
- {component}.test/gen/assemblies/<component>_0_frw/
  - Uses HDL file I/O workers
    - Can *only* be used on simulator platforms
    - Generally much faster simulations than co-simulation
    - (data in) and (data out) above are file_read.hdl and file_write.hdl

Data Flow Configurations

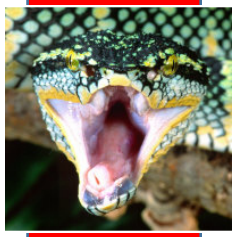


- Data flow as noted on previous slide:

  (data in) ⇨ metadata_stressor ⇨ UUT ⇨ backpressure ⇨ (data out)

- By default “backpressure” is applied to ensure an HDL worker properly respects flow control on its output
  - Can be disabled via XML with “`disableBackpressure`”
- “metadata_stressor” can ensure worker’s input properly handles intra-worker protocols and data starvation
  - Configured via XML with “`stressorMode`”
    - `bypass` – disabled (default configuration)
    - `throttle` – randomly holds back data (starvation)
    - `metadata` – manipulates metadata randomly (*e.g.* `som`) (edge conditions)
    - `full` – above two plus random idle cycles between messages (full compliance)

Unit Test Suite Debug

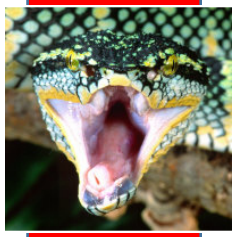


- Log files may be found at run/<platform>/case##.##.<component>.<implementation>.log
- The ocpirun command that was executed may be found in the log file

```
$ OCPI_LIBRARY_PATH=../../../lib/rcc:../../gen/assemblies:
$OCPI_CDK_DIR/lib/components/rcc ocpirun -d -v -mcomplex_mixer=hdl
-wcomplex_mixer=complex_mixer -Pcomplex_mixer=modelsim --sim-
dir=case00.00.complex_mixer.hdl.simulation --dump-
file=case00.00.complex_mixer.hdl.props
-pfile_write=fileName=case00.00.complex_mixer.hdl.out.out
../../gen/applications/case00.00.xml
```

- You can copy & paste (or modify) the above command, but it must be run from the run/<platform> directory

Unit Test Suite Simulation



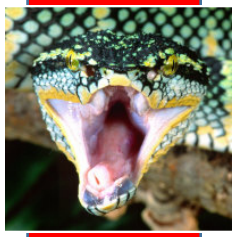
- Execute the ‘ocpiview’ command from the <component>.test/ directory

$ ocpiview run/xsim/case##.##.<component>.simulation/ &

- The ‘sim.out’ simulator log file may be found at

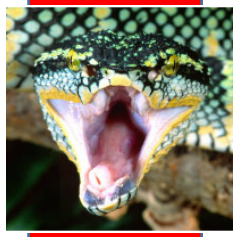
run/xsim/case##.##.<component>.simulation/<component>_0_frw.xsim.<datestamp>/sim.out

Peeking “Under the Hood”



- Advanced Usage Information
  - make targets for CLI usage
  - Directory layouts
  - Two examples of cases.xml
    - First is simple training example with Xilinx-provided cores
    - Second is ocpi.assets copy (complicated, CORDIC-based)

Make Targets (and GUI Buttons)



- From the project directory or component library directory

$ make test Tests=<component>.test HdlPlatform=xsim **(Generate/Build)**

$ make runtest Tests=<component>.test OnlyPlatforms=xsim View=1 **(Prepare/Run/Verify/View a specific test)**

$ make runtest OnlyPlatforms=xsim View=1 **(Prepare/Run/Verify/View a ALL tests)**

- From the <component>.test/ directory

$ make run OnlyPlatforms=xsim View=1 **(Prepare/Run/Verify/View)**

- Or from the <component>.test directory

$ make HdlPlatform=xsim **(Generate/Build, where default is Build that depends on Generate)**

$ make run OnlyPlatforms=xsim View=1 **(Prepare/Run/Verify/View, where Run means an interleaved Run/Verify that depends on Prepare)**

- Or from the <component>.test directory

$ make generate OnlyPlatforms=xsim **(Generate)**

$ make build OnlyPlatforms=xsim **(Build)**

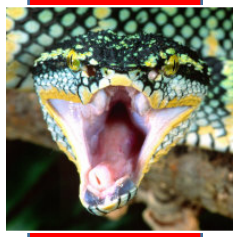
$ make prepare OnlyPlatforms=xsim **(Prepare)**

$ make runonly OnlyPlatforms=xsim **(Run)**

$ make verify OnlyPlatforms=xsim **(Verify)**

$ make view OnlyPlatforms=xsim **(View)**

Make Targets (cont)



- From the project directory or component library directory

$ make cleantest – clean all test results (removes the gen and run directories)

$ make cleanrun – clean all run results (removes the run directory)

$ make cleansim – clean all simulation output (removes the run/<simulator>/simulations directory)

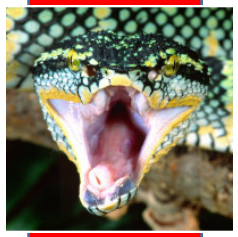
- Or from the <component>.test directory

$ make clean – clean all test results (removes the gen and run directories)

$ make cleanrun – clean all run results (removes the run directory)

$ make cleansim – clean all simulation output (removes the run/<simulator>/simulations directory)

Tree of complex_mixer.test/gen/ directory



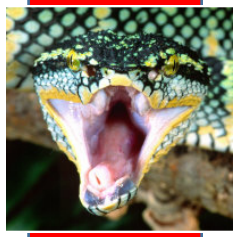
$ tree -L 3 gen

```
gen
├── applications
│   ├── case00.00.xml
│   ├── case00.01.xml
│   ├── case00.02.xml
│   ├── case00.03.xml
│   └── verify_case00.sh
├── assemblies
│   ├── complex_mixer_0
│   │   ├── complex_mixer_0.xml
│   │   └── Makefile
│   ├── complex_mixer_0_frw
│   │   ├── complex_mixer_0_frw.xml
│   │   ├── container-complex_mixer_0_frw_xsim_base
│   │   ├── container-complex_mixer_0_frw_modelsim_base
│   │   ├── gen
│   │   ├── lib
│   │   ├── Makefile
│   │   ├── target-xsim
│   │   └── target-modelsim
│   └── Makefile
```

$ tree -L 3 gen (continued)

```
gen
├── cases.txt
├── cases.xml
├── cases.xml.deps
└── inputs
    ├── case00.00.in
    ├── case00.01.in
    ├── case00.02.in
    └── case00.03.in
```

Tree of complex_mixer.test/run/ directory



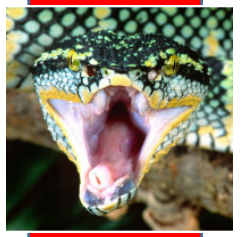
$ tree -L 3 run

```
run
├── centos7
│   ├── case00.00.complex_mixer.rcc.log
│   ├── case00.00.complex_mixer.rcc.out.out
│   ├── case00.00.complex_mixer.rcc.props
│   ├── case00.02.complex_mixer.rcc.log
│   ├── case00.02.complex_mixer.rcc.out.out
│   ├── case00.02.complex_mixer.rcc.props
│   └── run.sh
├── xsim
│   ├── case00.00.complex_mixer.hdl.log
│   ├── case00.00.complex_mixer.hdl.out.out
│   ├── case00.00.complex_mixer.hdl.props
│   ├── case00.00.complex_mixer.hdl.simulation
│   │   └── complex_mixer_0_frw.xsim.20170512172244
│   ├── case00.01.complex_mixer.hdl.log
│   ├── case00.01.complex_mixer.hdl.out.out
│   ├── case00.01.complex_mixer.hdl.props
│   ├── case00.01.complex_mixer.hdl.simulation
```

$ tree -L 3 run (continued)

```
run
│   │   └── complex_mixer_0_frw.xsim.20170512172303
│   ├── case00.02.complex_mixer.hdl.log
│   ├── case00.02.complex_mixer.hdl.out.out
│   ├── case00.02.complex_mixer.hdl.props
│   ├── case00.02.complex_mixer.hdl.simulation
│   │   └── complex_mixer_0_frw.xsim.20170512172323
│   ├── case00.03.complex_mixer.hdl.log
│   ├── case00.03.complex_mixer.hdl.out.out
│   ├── case00.03.complex_mixer.hdl.props
│   ├── case00.03.complex_mixer.hdl.simulation
│   │   └── complex_mixer_0_frw.xsim.20170512172342
│   └── run.sh
│
│   └── run.sh
└── runtests.sh
```

complex_mixer cases.txt Example 1



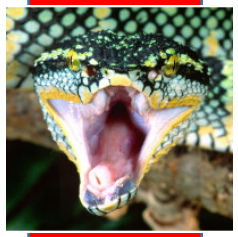
Values common to all property combinations:
==========
ocpi_debug = false
ocpi_endian = little
phs_inc = -8192

Descriptions of the 1 case and its subcases:
==========
Case case00:
Summary of subcases

Subcase #	enable	data_select
0:	false	false
1:		true
2:	true	false
3:		true

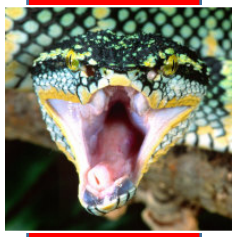
Subcase 00:
ocpi_debug = false
ocpi_endian = little
enable = false
phs_inc = -8192
data_select = false
Subcase 01:
ocpi_debug = false
ocpi_endian = little
enable = false
phs_inc = -8192
data_select = true
Subcase 02:
ocpi_debug = false
ocpi_endian = little
enable = true
phs_inc = -8192
data_select = false
Subcase 03:
ocpi_debug = false
ocpi_endian = little
enable = true
phs_inc = -8192
data_select = true

Unit Test Suite cases.xml Example 1



```
<cases spec='ocpi.training.complex_mixer'>
  <case name='case00'>
    <subcase id='0'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
      <worker name='complex_mixer' model='rcc' outputs='out'/>
    </subcase>
    <subcase id='1'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='2'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
      <worker name='complex_mixer' model='rcc' outputs='out'/>
    </subcase>
    <subcase id='3'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
    </subcase>
  </case>
</cases>
```

complex_mixer cases.txt Example 2



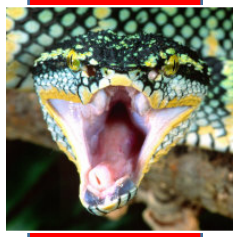
Values common to all property combinations:
==========
ocpi_debug = false
ocpi_endian = little
CHIPSCOPE_p = false (specific to worker complex_mixer.hdl)
CORDIC_STAGES_p = 16 (specific to worker complex_mixer.hdl)
PEAK_MONITOR_p = true (specific to worker complex_mixer.hdl)
phs_inc = -8192
phs_init = 0 (specific to worker complex_mixer.hdl)
mag = 1024 (specific to worker complex_mixer.hdl)
messageSize = 8192 (specific to worker complex_mixer.hdl)

Descriptions of the 1 case and its subcases:
==========
Case case00:
Summary of subcases

Subcase #	NCO_DATA_WIDTH_p	INPUT_DATA_WIDTH_p	enable	data_select
0:	12	12	false	false
1:				true
2:			true	false
3:				true
4:	16	16	false	false
5:				true
6:			true	false
7:				true

Subcase 00:
ocpi_debug = false
ocpi_endian = little
CHIPSCOPE_p = false
NCO_DATA_WIDTH_p = 12
INPUT_DATA_WIDTH_p = 12
CORDIC_STAGES_p = 16
PEAK_MONITOR_p = true
enable = false
phs_inc = -8192
phs_init = 0
mag = 1024
messageSize = 8192
data_select = false
Subcase 01:
ocpi_debug = false
ocpi_endian = little
CHIPSCOPE_p = false
NCO_DATA_WIDTH_p = 12
INPUT_DATA_WIDTH_p = 12
CORDIC_STAGES_p = 16
PEAK_MONITOR_p = true
enable = false
phs_inc = -8192
phs_init = 0
mag = 1024
messageSize = 8192
data_select = true

Unit Test Suite cases.xml Example 2



```
<cases spec='ocpi.assets.dsp_comps.complex_mixer'>
  <case name='case00'>
    <subcase id='0'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
      <worker name='complex_mixer' model='rcc' outputs='out'/>
    </subcase>
    <subcase id='1'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='2'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
      <worker name='complex_mixer' model='rcc' outputs='out'/>
    </subcase>
    <subcase id='3'>
      <worker name='complex_mixer' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='4'>
      <worker name='complex_mixer-1' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='5'>
      <worker name='complex_mixer-1' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='6'>
      <worker name='complex_mixer-1' model='hdl' outputs='out'/>
    </subcase>
    <subcase id='7'>
      <worker name='complex_mixer-1' model='hdl' outputs='out'/>
    </subcase>
  </case>
</cases>
```