



OpenCPI - Open Component Portability Infrastructure

**An Open Source Infrastructure for
Heterogeneous/Embedded Component-
Based Systems and Applications**

OpenCPI Unit Test Framework: Details

- Configuring file I/O handling
 - Using embedded opcodes
 - Calculating the best buffer size for the protocols in use
- Understanding the *two* HDL assemblies used for testing
 - When to use each one (simulators vs. hardware)
 - Test XML attributes for data flow configuration
- Debugging your failed tests
 - "Where did my overnight simulation go?"
- Understanding test cases: two examples of cases.xml

- OpenCPI unit test framework uses the OpenCPI utilities **ocpi.core.file_read** and **ocpi.core.file_write** for file I/O.
- These utilities have two modes of operation:
 - Data Streaming
 - **file_read**: input file contents are copied to payload of a stream of messages. Fixed number of bytes of data (defaults to 4KB), all with the same opcode.
 - **file_write**: output file contents become the payloads of a stream of messages. No message lengths or opcodes are recorded.
 - Messaging
 - **file_read**: input file contents are interpreted as a sequence of defined messages. An 8-byte header precedes payload data of each messages. Four-byte length (in bytes, little-endian): one byte opcode, three padding bytes.
 - **file_write**: output file contents are written as a sequence of defined messages. Uses same 8-byte header and 4-byte format as **file_read**.
- For messaging mode, use the **messageInFile** test XML attribute to select it and follow the input data format
- For data streaming mode, use the **messageSize** test XML attribute to define buffer size
- For details as to how file I/O is typically performed in OpenCPI applications, see the [ocpi.core.file_read](#) and [ocpi.core.file_write](#) data sheets

Using *messagesInFile*

- The test XML attribute **messagesInFile** enables messaging mode for **file_read** workers
- If more than one opcode is needed, set **messagesInFile="true"**
- Input data needs to have:
 - Length of message (in bytes)
 - Opcode number to transmit
 - Message (if non-zero length)
- Example with four 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.
---	---	--------------------------	---	---	-------------------------------	-------------------------------	---	---	-------------------------	---	---	-------------------------

- The test XML attribute **messageSize** specifies the buffer size, which also determines the size of messages read from test input
- In general, bigger the buffer, higher the throughput
- Determine a **messageSize** that best uses 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 = 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**

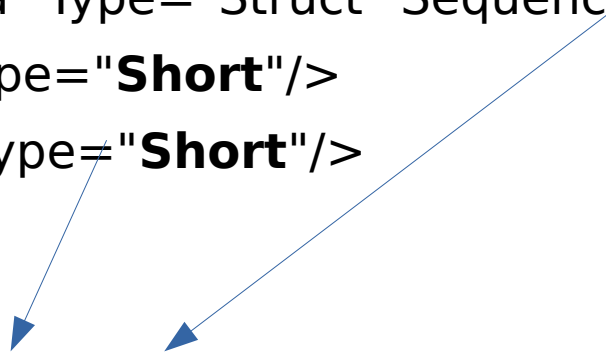
<Member Name="Q" Type="**Short**

</Argument>

</Operation>

</Protocol>

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



Protocol Buffer Size (Continued)

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

Example 2

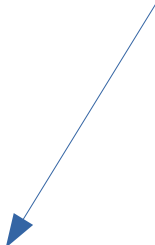
<Protocol>

<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 (Continued)

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) $\Rightarrow$ metadata_stressor $\Rightarrow$ UUT $\Rightarrow$ backpressure $\Rightarrow$ (data out)
- One test assembly uses file I/O in an RCC worker
 - Can be built for *any* HDL platform, including simulators ("co-simulation")
 - Uses the simulatable portions of the data plane
 - No DMA engine
 - (data in) and (data out) above are the assembly's external ports
 - This test assembly is used by default
- The other test assembly 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** and **file_write** HDL workers
 - To use, set test XML attribute **UseHdlFileIO="true"**

Test XML Attributes for Data Flow Configuration

- Data flow as noted on previous slide:
(data in) $\Rightarrow$ metadata_stressor $\Rightarrow$ UUT $\Rightarrow$ backpressure $\Rightarrow$ (data out)
- By default, "backpressure" is applied to ensure an HDL worker properly respects flow control on its output
 - Can be disabled with test XML attribute **disableBackpressure**
- "metadata_stressor" can ensure worker's input properly handles intra-worker protocols and data starvation
 - Configured with test XML attribute **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)

- Log files are located in:
`<component>.test/run/<platform>/case##.##.<component>.<implementation>.log`
- The **ocpirun** command that was executed can be found in the log file.
For example:

```
$ 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

- Execute the **ocpiview** command from the `<component>.test/` directory

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

- The **sim.out** simulator log file can be found at

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

Two Examples of cases.xml

- First is simple **ocpi.tutorial** example with Xilinx-provided cores
- Second is **ocpi.assets** copy (complicated, CORDIC-based)

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.tutorial.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        true
2:                                     true        false
3:                                     true        true
4:          16              16              false      false
5:                                     true        true
6:                                     true        false
7:                                     true        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>
  </case>
</cases>
```

```
<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>
```