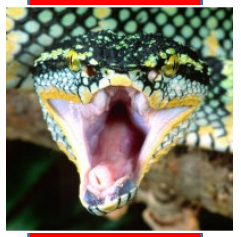
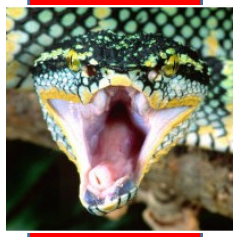


OpenCPI

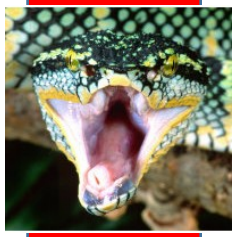
Automated Unit Test Suite

Application Execution



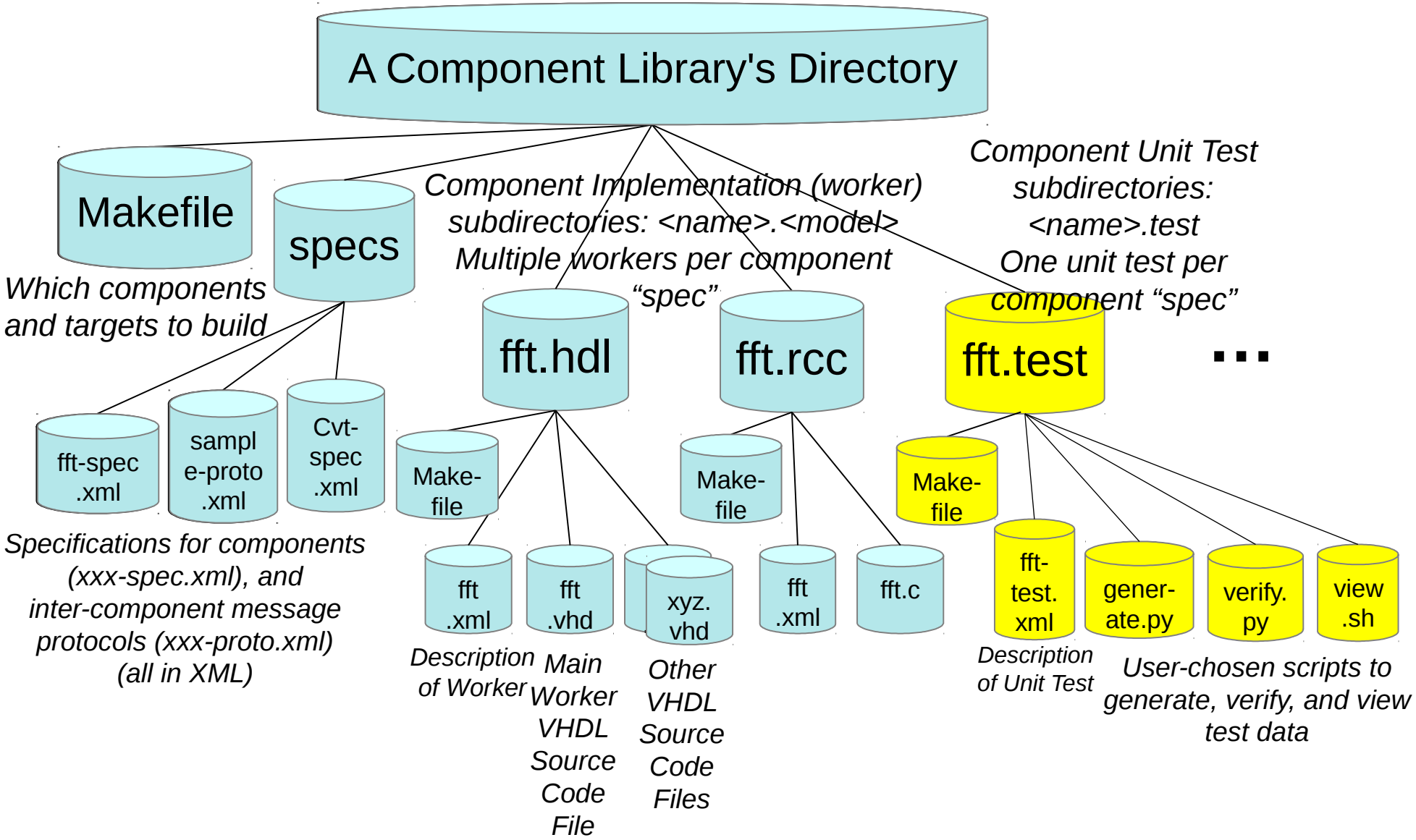
- Two ways for executing applications
  1) ocpirun
     - Simple command-line program
     - Only support static-control
     - **Used by the Automated Unit Test Suite**
  2) OpenCPI Application Control Interface (ACI)
     - Written in C++
     - Allows for dynamic-control
     - Sometimes referred to as “C API”

Automated Unit Test Suite

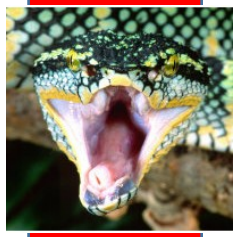


- Based on a single Component Specification
  - <component>.test/
- Creates individual parameterized test Cases based on
  - HDL hardware, simulation, and RCC target platforms
  - Multiple workers
    - one OCS implemented by many Workers (Authoring Models and/or different implementations)
  - <worker>-build.xml - Worker build configurations
    - Build-time/Compile-time (property parameters)
  - <component>-test.xml
    - Run-time settable property values from OCS and OWD
    - Run-time test properties (optional)
    - Customized Cases (ability to over-ride the default of “all combinations” of the above)
- **In v1.3, Automated Unit Test Suite only supports Application Workers**

Component Directory Layout



Create Component Unit Test directory

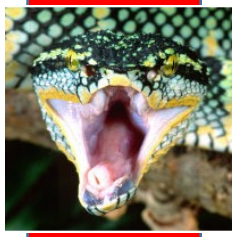


- **$ ocpidev create test <components-spec>**
  - creates <component>.test/ containing skeleton files
    - Makefile
    - <component>-test.xml
    - generate.py (optionally generate input test data; one script per input port*)
    - verify.py (optionally verify output test data; one script per output port*)
    - view.sh (optionally view data)

AV IDE v1.3 does NOT support creation of Unit Test directory

* Not supported in v1.3 – limited to a single input/output port

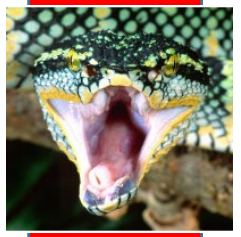
Unit Test – Five Phases



- **Generate** – creates the “gen” sub-directory
  - generates input data, OAS, OHAD, case configuration(s)
- **Build** (HDL workers only; no-op for RCC workers)
  - builds HDL subassemblies for target platforms (bitstream/executable)
- **Prepare** – creates the “run” sub-directory
  - examines available built workers and platforms; creates execution scripts
- **Run**
  - executes tests for all workers, configurations, test cases, and platforms
- **Verify**
  - verifies results from the execution of test cases on workers and platforms

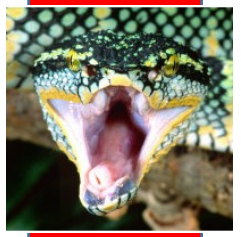
AV IDE v1.3 does NOT provide ability to execute phases

Unit Test – Generate



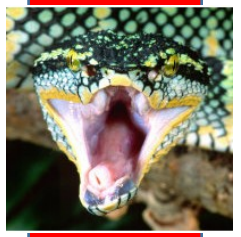
- Discovers
  - OCS associated with this test directory
  - Workers in the same component library that implement this OCS
  - Build configurations (parameter values) for each worker per each <worker>-build.xml or <worker>.build
- Derives
  - Test cases that are “baselined” for build (parameter) configurations from any worker
  - The actual tests appropriate for all build (parameter) combinations vs. the actual worker build configurations they apply to
    - **default** is all, but may be reduced by the property element in the Unit Test Suite Description XML file
- Generates
  - OAS application XML files for launching unit tests
  - OHAD HDL assembly XML files that are built for HDL platforms (hardware and simulation)
  - If necessary, generates input and/or property value files

Unit Test – Build (HDL workers only)



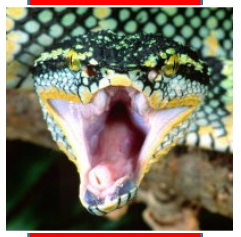
- Builds the generated HDL assemblies for specified platforms (hardware and simulators)
- gen/assemblies/<component>_<ParamConfig>/<component>_<ParamConfig>.xml
  - OHAD is the UUT only with external ports
  - may be used in both hardware and Co-Simulation unit tests – does simulate the data plane (slow)
- gen/assemblies/<component>_<ParamConfig>_frw/<component>_<ParamConfig>_frw.xml
  - OHAD includes the UUT with possible file_read/file_write HDL workers
  - used in simulation only – does NOT simulate the data plane (fast)
- <ParamConfig> is found in <worker>.hdl/gen/<component>-params.mk
  - look-up table listing all parameter values for each particular worker build configuration

Unit Test – Prepare



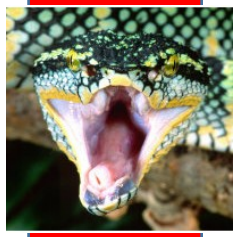
- Discovers available
  - Execution platforms, both local and remote (network)
  - Built artifacts that can be executed on available platforms
- Generates
  - Test scripts to perform all feasible tests on all available platforms

Unit Test – Run and Verify



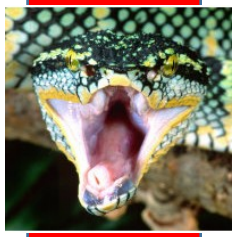
- Sequential or Interleaved (default)
- Run
  - All scripts are run per subcase
    - Access to parameter properties and
    - Run-time properties of the subcase being tested
- Verify
  - Optionally execute view.sh to view test input/output data
    - “View=1”
  - Simulation data
    - (default) deleted upon successful verification (PASSED)
    - “KeepSimulations=1” retains output data after verification, regardless of results

Unit Test Suite Description XML File



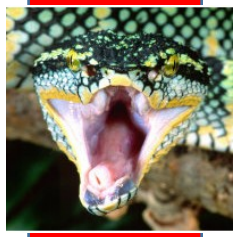
- <component>-test.xml
  - specifies test cases and defaults that apply to all test cases
- input
  - defines a prepared input file or generator script
- output
  - defines an output file or verify script
- property
  - defines property values for all test cases (OCS, OWD, or test)
- case
  - used to define a non-default test case when needed
  - necessary when the automatic parameterization of the default test case is invalid or excessive for testing the worker(s)
  - when no case element is defined, the default is used (common)

Unit Test Suite Description XML File Example 1



```
<tests useHDLFileIo='true'>
  <input port='in' script='generate.py 100 12.5 32767 16384' messagesize='8192'/>
  <output port='out' script='verify.py 100 16384' view='view.sh'/>
  <property name='phs_inc' values='-8192'/>
  <property name='enable' values='0,1'/>
  <property name='data_select' values='0,1'/>
</tests>
```

Unit Test Suite Description XML File Example 2



```
<tests useHDLFileIo='true'>
  <output port='out' script='verify.m' />
  <property name='ta'  values='0x1ACE2ACE3ACE'/>
  <property name='ra'  values='0x0FACEBA1B0A0'/>
  <property name='tmr' values='0x000FA0'/>
  <property name='pre' values='0x00'/>
  <property name='start' values='true'/>
  <case>
    <property name='messageSize' values='3872'/>
    <property name='sig' values='0x00'/>
  </case>
  <case>
    <property name='messageSize' values='2992,1936 '/>
    <property name='sig' values='0x01'/>
  </case>
  <case>
    <property name='messageSize' values='2432,1376'/>
    <property name='sig' values='0x02'/>
  </case>
  <case>
    <property name='messageSize' values='2272,1216'/>
    <property name='sig' values='0x03'/>
  </case>
</tests>
```

Unit Test Suite Description XML File (cont)

Tests **element Attribute**	**Data Type**	**Description**
Spec	string	Overrides the default behavior where the OCS is inferred from the name of the <component>.test directory, and found in the ../specs/<component>-spec.xml file.
UseHdlFileIO	boolean	Applies only when HDL workers are being tested on simulation platforms. When true, file I/O is handled in the simulator using file read/write HDL workers. When false (default), file I/O is handled using file read/write RCC workers outside the simulator.
ExcludeWorkers	string	A list of comma-separated workers that should *not* be tested, even if they implement the spec of this test suite.
OnlyWorkers	string	A list of comma-separated workers that should be the *only* ones tested, where others found to implement the same spec will be ignored.
ExcludePlatforms	string	A list of comma-separated platforms that should *not* be tested. Any other available platforms that have built artifacts will be used.
OnlyPlatforms	string	A list of comma-separated platforms that should be the *only* ones tested. Any other available platforms will be ignored.
Duration	integer	An amount of time the application should run before being considered successfully ***done***. **Cannot be used with *Timeout.***
Timeout	integer	An amount of wall clock time in seconds after which the execution of a test subcase is considered a failure. **Cannot be used with *Duration.***

Unit Test Suite Description XML File (cont)

***Tests* element child element**	Attribute	Type	Description
input	name	string	Optionally specifies the name of this input source. Not necessary if applied to all cases, but useful when shared across multiple cases. More common to use the port attribute if it applies only to a specific port. **Must specify either name or port.**
	port	string	Optionally specifies the name of the port that this input source will always apply to. If there is only one input source for a port, it will be used for all cases. **Must specify either name or port.**
	script	string	Used to indicate a command to execute to produce data. Can be a command name followed by command arguments, where the last argument is implicitly the file to be written into. **Must have execute permissions.**
	file	string	Used to specify the name of a file to be used as the source of data.
	messageSize	integer	A positive number that specifies the size of messages (in Bytes).
	messagesInFile	boolean	Indicates that the input data contains message boundaries and opcodes.

```
<input port='in' script='generate.py 32768' messagesize='8192'/>
<input port='in' file='../../../applications/FSK/idata/Os.jpeg' messagesize='8192'/>
```

Unit Test Suite Description XML File (cont)

Tests element child element	Attribute	Type	Description
output	name	string	Optionally specifies the name of this output source. Not necessary if applied to all cases, but useful when shared across multiple cases. More common to use the port attribute if it applies only to a specific port. **Must specify either name or port.**
	port	string	Optionally specifies the name of the port that this output source will always apply to. If there is only one output source for a port, it will be used for all cases. **Must specify either name or port.**
	script	string	Used to indicate a command to execute to validate data. Can be a command name followed by command arguments, where the last arguments are implicitly the output filename from the given port followed by each input port filename in the order the ports are declared in the OCS. **Must have execute permissions.**
	file	string	Used to specify the name of a golden file used to compare to output data.
	view	string	Optionally "view" the data for the port in some viewing or plotting window. Takes all of the same arguments as the verification script mentioned above.

<output port='out' script='verify.py 2048' view='view.sh 2048'/>
<output port='out' file='goldfile.bin' view='view.sh'/>

Unit Test Suite Description XML File (cont)

Tests element child element	Attribute	Type	Description
property	name	string	Required. Identifies a property defined in the OCS or OWD, or declares a new test property.
	test	boolean	Optionally indicates that the property is a test property and not a property of the worker(s) being tested. Must be a unique name not already used in the OCS/OWD. Used to generate other test cases not defined simply by the values of worker properties.
	value		Specifies a single value to be tested.
	values		Specifies a comma-separated sequence of values to be tested.
	valueFile	string	Specifies the name of a file containing a single value to be tested. Multiple lines in the file are considered elements of a sequence or array value.
	valuesFile	string	Specifies the name of a file containing multiple values to be tested. Multiple lines are considered separate values to be tested.
	generate	string	Specifies a command to execute to create a file containing a value to be tested. Used when a property depends on others in a complex way. The last argument is implicitly the file to be written into. Must have execute permissions.

```
<property name='ref' value='0x1B26'/>
<property name='data_select' values='0,1'/>
```

```
<property name='taps' generate='gen_lpf_taps.py'/>
<property name='samplesPerBaud' test='true' value='25'/>
```

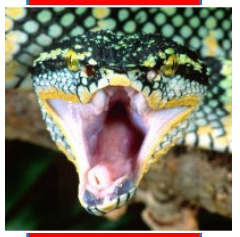
Unit Test Suite Description XML File (cont)

Tests **element child element**	**Attribute**	**Type**	**Description**
case	Name	string	Optionally specifies the name of the test case. If omitted, the name of the case is **case** followed by a case number (zero-based) with at least two digits. May be used when specifying that only certain cases (rather than all) should be executed or verified.
	ExcludeWorkers	string	A list of comma-separated workers that should *not* be tested, even if they implement the spec of this test suite.
	OnlyWorkers	string	A list of comma-separated workers that should be the *only* ones tested, where others found to implement the same spec will be ignored.
	ExcludePlatforms	string	A list of comma-separated platforms that should *not* be tested. Any other available platforms that have built artifacts will be used.
	OnlyPlatforms	string	A list of comma-separated platforms that should be the *only* ones tested. Any other available platforms will be ignored.

Unit Test Suite Description XML File (cont)

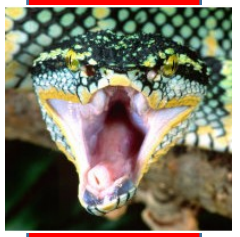
Case element child element	Description
input	Overrides the default input per port. If no **input** is defined for an input port, the default **input** is used as defined for the port under the top-level **tests** element. The **port** attribute may be used, but if the **name** attribute names an input source at the top-level that already has a **port** attribute, it may be omitted. When the **name** attribute is used both **file/script** attributes are disallowed.
output	Overrides the default output per port. If no **output** is defined for an output port, the default **output** is used as defined for the port under the top-level **tests** element. The **port** attribute may be used, but if the **name** attribute names an output source at the top-level that already has a **port** attribute, it may be omitted. When the **name** attribute is used both **file/script** attributes are disallowed.
property	Overrides default value(s). If no **property** element is present for a property under a **case** element, then the value(s) defined at the top-level are used. For parameter properties, the default value(s) tested are derived from value(s) defined in all worker(s)' build configurations, but this automatic default may be overridden (limited) by a **property** element either at the top-level or under a **case** element. A single test **case** can have multiple values for any property, where sub-cases are automatically generated for all combinations of property values for the test case whether specified at the top level as defaults sets of values or specified for the test case in the **case** element.

Unit Test Suite Makefile



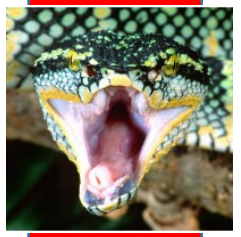
- Typically includes only one line – other mods are for the power-user
  - include $(OCPI_CDK_DIR)/include/test.mk
- Make variables are available to control various phases of unit testing (set on the make command-line or within Makefile)
  - HdlPlatform(s) – Build phase
  - OnlyPlatform(s), ExcludePlatforms(s) – Build/Prepare/Run/Verify phases
    - Recommend instead setting these in the Unit Test Suite Description XML file
  - View = 1 causes the “view” script to be run whenever verification is requested
  - TestVerbose = 1 causes execution and verification output to be output to the console/shell, in addition to per platform/subcase log files
  - KeepSimulations = 1 causes the contents of the simulations directory to be retained after successful execution on simulation platforms
  - Cases = is a wildcard pattern indicating which cases/subcases should be executed or verified

Unit Test Suite Scripts



- All Parameter and Initial/Writable run-time property values are supplied to each script as environment variables
  - OCPI_TEST_<myprop>
  - Output scripts also have access to final values of Writable and Volatile properties
- Scripts may be parameterized by these values for the subcase being generated
- Exit status of zero indicates success, while a non-zero exit status indicates failure
  - Framework prints green/red colors for PASSED/FAILED based on the exit status (terminal settings permitting)
  - May write other informational messages to STDERR to be logged
- Scripts must be executable and found in the <component>.test directory
- C/C++
  - getenv(“OCPI_TEST_myprop”)
- python
  - os.environ.get(“OCPI_TEST_myprop”)
  - #!/usr/bin/env python
- bash/shell
  - $OCPI_TEST_myprop
  - #!/bin/bash --noprofile

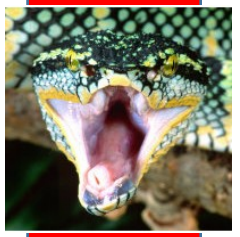
Unit Test Suite view.sh Examples



```
#!/bin/bash –noprofile

# Plot the time/fft of input file ($2)

$OCPI_PROJECT_DIR/scripts/plotAndFft.py $2 complex 32768 100 &

# Plot the time/fft of output file ($1)

$OCPI_PROJECT_DIR/scripts/plotAndFft.py $1 complex 32768 100 &
```

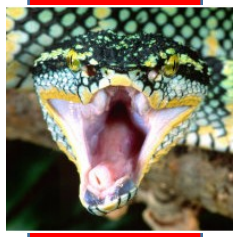
```
#!/bin/bash –noprofile

# Plot the time/fft of output file ($1), and use property value ($OCPI_TEST_num_zeros) to
calculate the number of real samples

$OCPI_PROJECT_DIR/scripts/plotAndFft.py $1 real `echo "$OCPI_TEST_num_zeros+1*100" |
bc` 1 &
```

Unit Test Generate/Build Summary



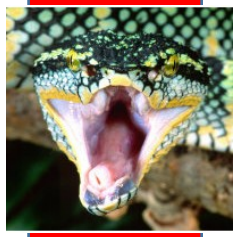
- Create the unit test suite via ocpidev create test <OCS>
- Modify the Makefile (rarely needed)
- Modify the <component>-test.xml file with input/output/property/case
- Prepare input data files and/or input data generator scripts
- Prepare output data gold files and/or output verification scripts
- Execute the Generate phase
- Examine the gen/cases.txt report to verify the generated subcases
- For HDL workers, execute the Build phase to build HDL assemblies
- All Generate/Build results are found in the gen/ directory

Peeking “Under the Hood”



- Two examples of cases.xml
  - First is simple training example with Xilinx-provided cores
  - Second is ocpi.assets copy (complicated, CORDIC-based)
- Assemblies with and without HDL-based file I/O

Tree of complex_mixer.test/gen/ directory



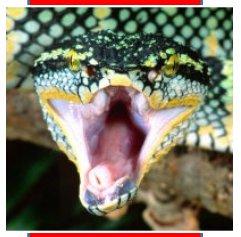
$ tree -L 3 gen

```
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
```

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

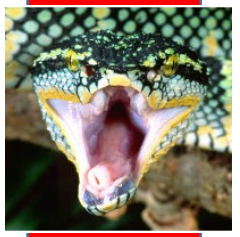
Subcase #	enable	data_select
0:	false	false
1:		true
2:	true	false
3:		true

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>
```

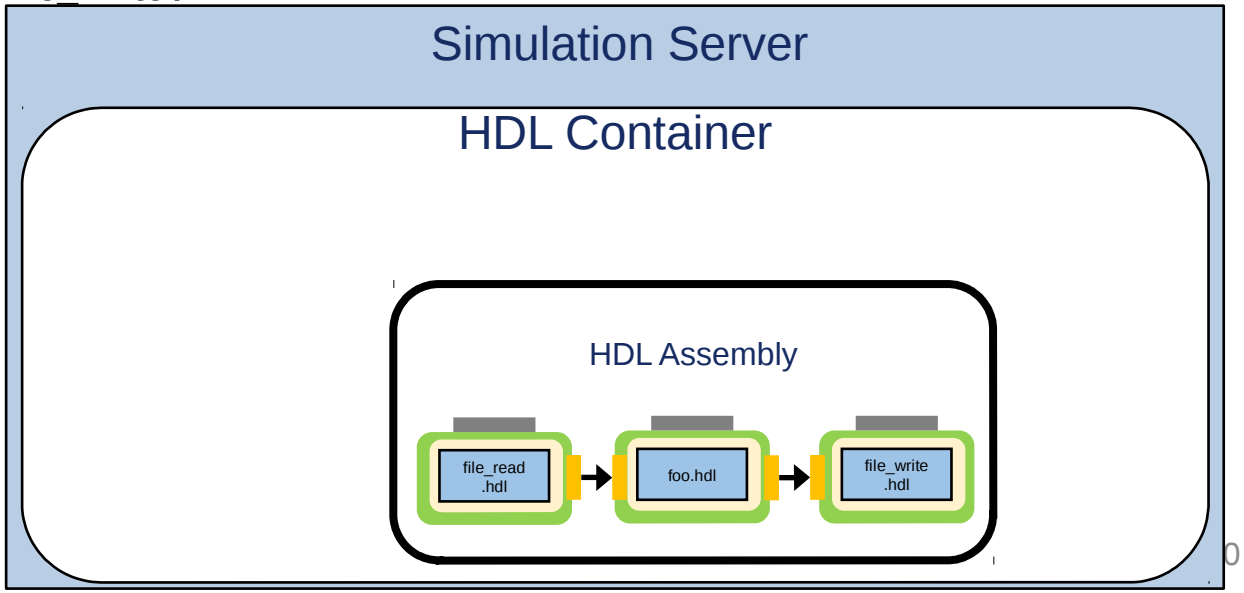
{component}.test/gen/assemblies/<component>_0_frw/

../<component>_0_frw.xml:

```
<HdlAssembly>
  <Instance Worker='complex_mixer' ParamConfig='0'/>
  <Instance name='complex_mixer_in' Worker='file_read'/>
  <Connection>
    <port instance='complex_mixer_in' from='out'/>
    <port instance='complex_mixer' to='in'/>
  </Connection>
  <Instance name='complex_mixer_out' Worker='file_write'/>
  <Connection>
    <port instance='complex_mixer_out' to='in'/>
    <port instance='complex_mixer' from='out'/>
  </Connection>
</HdlAssembly>
```

Makefile

```
override HdlPlatform:=$(filter %sim,$(HdlPlatform))
override HdlPlatforms:=$(filter %sim,$(HdlPlatforms))
include $(OCPI_CDK_DIR)/include/hdl/hdl-assembly.mk
```



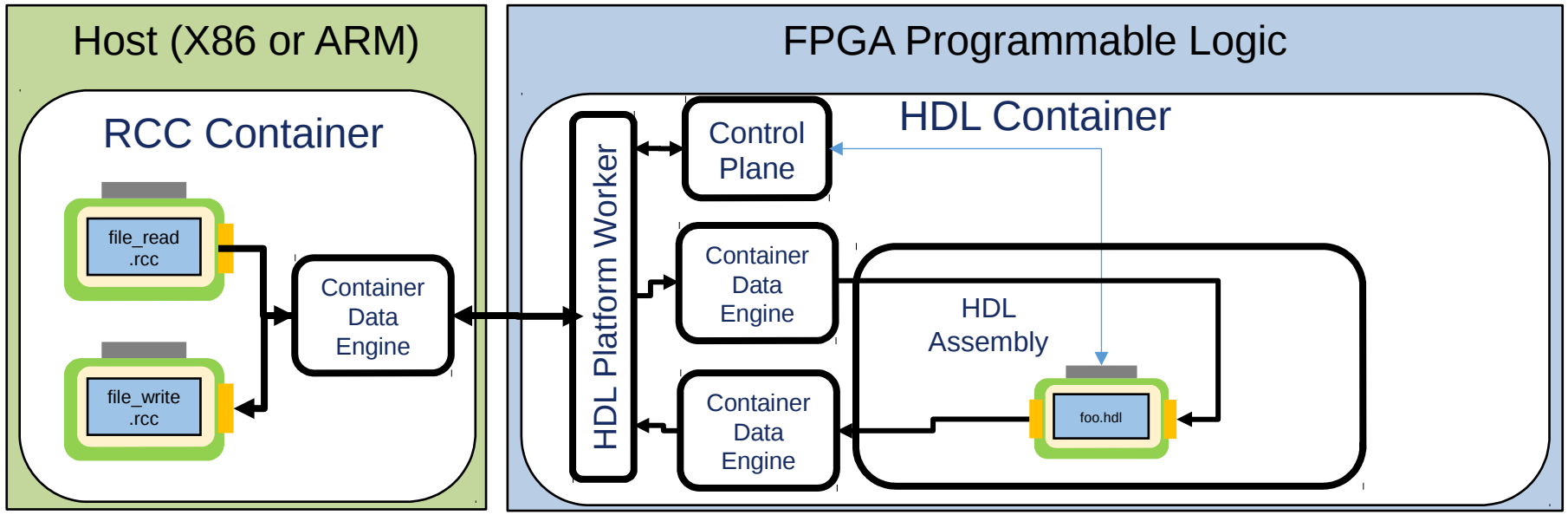
{component}.test/gen/assemblies/<component>_0/

<component>_0.xml

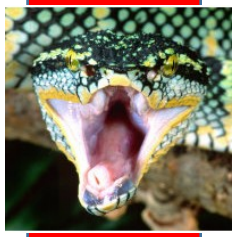
```
<HdlAssembly>
  <Instance Worker='complex_mixer' ParamConfig='0' externals='true'/>
</HdlAssembly>
```

Makefile

```
override HdlPlatform:=$(filter-out %sim,$(HdlPlatform))
override HdlPlatforms:=$(filter-out %sim,$(HdlPlatforms))
include $(OCPI_CDK_DIR)/include/hdl/hdl-assembly.mk
```



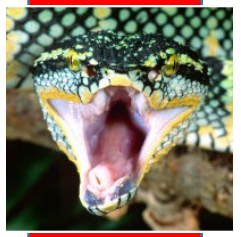
Executing Tests on Remote Test Systems



- Extends the unit test system beyond the current development system
- Remote platforms specified by **OCPI_REMOTE_TEST_SYSTEMS**
  - Colon separated list of Remote Test Systems with four fields, each separated by '='
  - **Host name/IP address = SSH user name = SSH password = Project mount path on remote system**
  - Example: 10.11.12.13=root=root=/mnt/myproj:10.11.12.14=root=root=/mnt/myproj
- Remote Test Systems are accessed via **SSH**
- Project's directory on the development system **MUST** be NFS mounted by the Remote Test System and its mounted directory name must match the project's directory, as indicated by the fourth field above
- Remote and development systems should use the same release version
- Set the OCPI_REMOTE_TEST_SYSTEMS environment variable

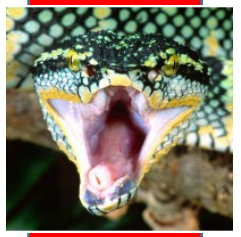
  export OCPI_REMOTE_TEST_SYSTEMS:=10.11.12.13=root=root=/mnt/myproj

Tests Execution and Verification



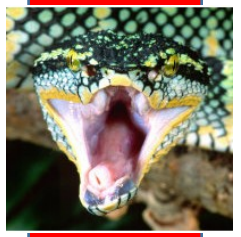
- Execute the Prepare phase
  - By **Default**, all local and remote platforms are considered
    - RCC, HDL hardware, and HDL simulators
  - Ability to filter which Platforms are used for execution
    - **OnlyPlatform(s)="xsim, cento7"** and **ExcludePlatform(s)="xsim, matchstiq_z1"**
  - Discovers available RCC built artifacts in the component library, and **local** HDL artifacts built during the Build phase from generated **HDL test assemblies**
  - From the combination of available platforms and available artifacts it determines which sub-cases can be run on which platforms, and generates test execution scripts accordingly in each run/<platform> sub-directory
  - Execution of unit tests overrides any user-specified OCPI_LIBRARY_PATH
    - Limits the artifact search to local RCC artifacts in the component library, the gen/assemblies directory for HDL artifacts, and the installed CDK on both local and remote systems for both file_read/file_write when RCC implementations are used

Tests Execution and Verification (cont)



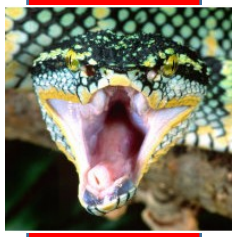
- Execute the Run phase
  - Test applications are executed for all possible cases/sub-cases per platform
  - Ability to filter which Platforms are used for execution
    - **OnlyPlatform(s)=”xsim, cento7”** and **ExcludePlatform(s)=”xsim, matchstiq_z1”**
  - The recorded results, per sub-case and platform, are
    - output data from each output port
    - final values of all properties, including volatile properties
    - a log file of the actual test execution

Tests Execution and Verification (cont)



- Execute the Verify phase
  - By **Default**, all output results are verified
    - RCC, HDL hardware, and HDL simulators
  - Ability to filter which Platforms are used for execution
    - **OnlyPlatform(s)="xsim, centos7"** and **ExcludePlatform(s)="xsim, matchstiq_z1"**
  - Relies on previous execution of appropriate sub-cases for all platforms
    - Performs verification using results previously recorded in the Run phase
  - The Verify phase by itself does not involve execution or access to local or remote platforms and may be performed off-line
  - The View = 1 option enables running the view scripts during verification
    - this happens per sub-case before each verification
  - The Verify phase will fail if any sub-cases fail by returning non-zero exit status
    - each sub-case individually reports PASSED/FAILED status

Make Targets



- 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)**

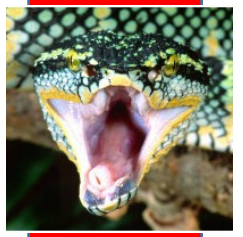
$ make prepare OnlyPlatforms=xsim **(Prepare)**

$ 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)

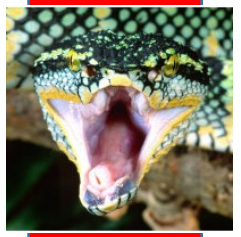
- Or from the <component>.test directory

$ 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/run/ directory



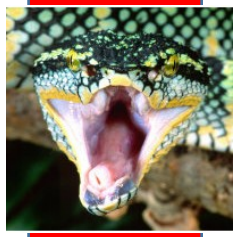
$ tree -L 3 run

```
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
```

Unit Test Suite Debug

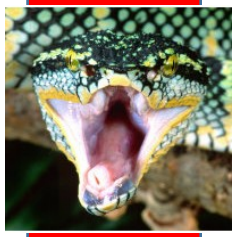


- Log files may be found at run/<platform>/case##.##.<component>.<implementation>.log
- The ocpirun command that was executed may be found in the log file

```
$ 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



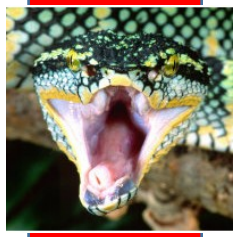
- Execute the ‘ocpiview’ command from the <component>.test/ directory

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

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

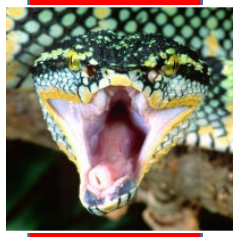
$ run/xsim/case##.##.<component>.simulation/<component>_0_frw.xsim.<datestamp>/sim.out

Unit Test Suite Limitations



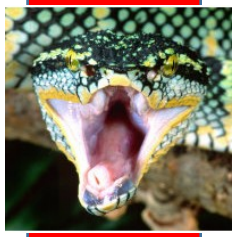
- Currently, unit testing support Application Workers **ONLY**
  - **Otherwise, unit tests are "handcrafted"**
- Coming soon – unit test support for Device Workers!
  - To include use of Device Emulators
- Unit Tests solely use ***ocpirun*** – no mechanism for an ACI
  - Coming soon – change a property while running (i.e. ACI)
- Fixed Application XML (OAS): **file_read -> UUT -> file_write**
  - Application Worker **MUST** propagate Zero-Length Messages
  - **OR** set the top-level *duration* or *timeout* attributes of the <component>-test.xml

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 becomes the payloads of a stream of messages, each carrying a fixed number of bytes of file data and all with the same opcode
    - ***messageSize*** defaults to 4KB
  - 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



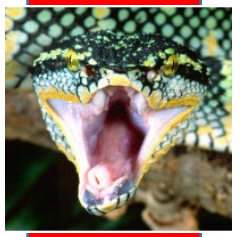
- Implements the ocpi.core/components/specs/file_write-spec.xml
- No Protocol (opcode default is zero)
- ***fileName*** path limit is 1024 char
- Two modes of operation
  - Data Streaming – Contents of file becomes the payloads of a stream of messages, no message lengths or opcodes are recorded.
  - 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:

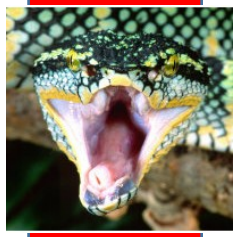
2	0	short scalar mesg.	4	1	short sequence(0) mesg.	short sequence(1) mesg.	4	2	long scalar mesg.	1	3	char scalar mesg.

File Read *messageSize*



- In general, bigger the buffer, higher the latency
- Determine a *messageSize* that best utilizes the protocol buffer size
  - protocol buffer size is the largest of all the operations in a protocol
- 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, ArrayLength, ArrayDimentions or size of the type

Protocol Buffer Size

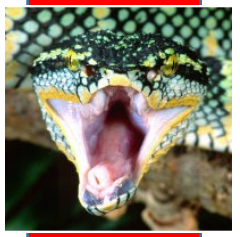


protocol buffer size = max operation size × length of the max operation size

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



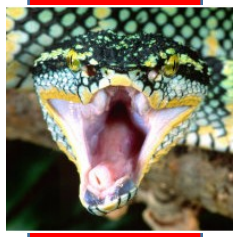
protocol buffer size = max operation size × length of the max operation size

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

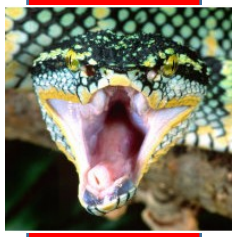


protocol buffer size = max operation size × length of the max operation size

Example 2

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

Protocol Buffer Size



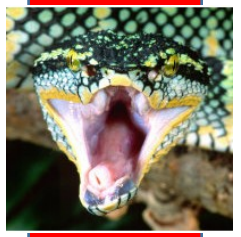
protocol buffer size = max operation size × length of the max operation size

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

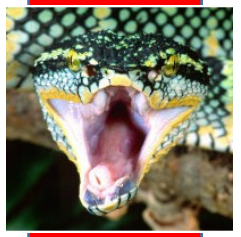


protocol buffer size = max operation size × length of the max operation size

Example 2

```
<Protocol>
  <Operation Name="demo1">
    <Argument Name="data" Type="longlong" ArrayLength="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" ArrayLength="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