OSVVM in a Nutshell,
VHDL's #1 Verification Methodology

by
Jim Lewis
OSVVM Chief Architect
IEEE 1076 VHDL Working Group Chair
VHDL Training Expert at SynthWorks
Jim@SynthWorks.com
Why VHDL?

- VHDL is #1 for FPGA Design and Verification
- From Wilson Research Group 2022 Functional Verification Survey
  - For FPGA design: 66% worldwide use VHDL
  - For FPGA verification: 56% worldwide use VHDL

Why OSVVM?

- OSVVM is VHDL's #1 Verification Methodology
- For FPGA Verification,
  - Worldwide: 28% use OSVVM = 50% of the VHDL FPGA users

What is OSVVM?

Verification Framework
Transaction Interface & API
Verification Components
Test Sequencer (Test Cases)

Verification Utility Library
Constrained Random, Scoreboards,
Functional Coverage, Memory Models,
Error tracking, and Message filtering, ...

Verification Component Library
AXI4 Full, Lite, AXI Stream,
UART, xMII, DPRam, ...

Script Library
Tool Independent Scripts
One Script to Run them All

Co-Simulation
Run Software in Hardware Simulator
Write tests in C++

Test Reports
HTML Test Suite, Test Case, Log Files
JUnit XML for CI/CD tools

• OSVVM is Free, Open Source
• Developed by the same VHDL experts who helped with VHDL Standards
# OSVVM History

<table>
<thead>
<tr>
<th>Year</th>
<th>OSVVM and SynthWorks Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>Transaction Framework, TbUtilPkg</td>
</tr>
<tr>
<td>2006</td>
<td>RandomPkg, ResolutionPkg, ScoreboardPkg, MemoryPkg</td>
</tr>
<tr>
<td>2010</td>
<td>CoveragePkg</td>
</tr>
<tr>
<td>2011</td>
<td>RandomPkg, CoveragePkg</td>
</tr>
<tr>
<td>2015</td>
<td>AlertLogPkg, TranscriptPkg, MemoryPkg</td>
</tr>
<tr>
<td>2016</td>
<td>ScoreboardGenericPkg, TbUtilPkg, ResolutionPkg</td>
</tr>
<tr>
<td>2018</td>
<td>Axi4Lite, AxiStream, UART</td>
</tr>
<tr>
<td>2020</td>
<td>Scripting, Specification Tracking, MIT, Virtual Interfaces, Axi4 Full and AxiStream, both with Bursting</td>
</tr>
<tr>
<td>2021</td>
<td>Singleton Data Structures, HTML &amp; JUnit XML reports</td>
</tr>
<tr>
<td>2022</td>
<td>HTML Log &amp; Scoreboard Reports, Code Coverage Reports, Ethernet VC, Arrays of Transaction Interfaces</td>
</tr>
<tr>
<td>2023</td>
<td>Co-simulation of C++ Software in a Hardware Simulator VC with delay randomization, Specification Tracking Part 2</td>
</tr>
</tbody>
</table>
OSVVM Verification Framework

- Looks identical to a SystemVerilog framework:
  - Verification components (VC) implement interface signaling
  - Test sequencer (TestCtrl) calls transactions = test case
  - Each test case is a separate architecture of TestCtrl

![Diagram of OSVVM Verification Framework]

- TbAxi4
- AXI Manager
- Clock and Reset
- DUT MemIO
  - Axilf
  - IntCtrl
  - UART
  - FIFO
  - MemIf
- UART Rx
- UART Tx
- SRAM Model
- TestCtrl
- Entity TestCtrl_e
- Test 1 Architecture
- Test 2 Architecture
- Test n Architecture
library osvvm, osvvm_Axi4;
  context osvvm.OsvvmContext;
...
entity TbAxi4 is
end entity TbAxi4;
architecture TestHarness of TbAxi4 is
  ...
  signal ManagerRec : AddressBusRecType (  
    Address (AXI_ADDR_WIDTH-1 downto 0),  
    DataToModel (AXI_DATA_WIDTH-1 downto 0),  
    DataFromModel(AXI_DATA_WIDTH-1 downto 0)  
  );
begin
  osvvm.TbUtilPkg.CreateClock(Clk, tperiod_Clk);
  osvvm.TbUtilPkg.CreateReset(nReset, . . .);

DUT_1: DUT ( . . . );

Axi4Manager_1 : Axi4Manager (MRec, . . . );
UartRx_1 : UartRx(RxRec, . . . );
UartTx_1 : UartTx(TxRec, . . . );
TestCtrl_1 : TestCtrl (TxRec, RxRec, MRec, nReset);
end TestHarness;
3 Steps to Verification Components

- Step 1: Transaction Interface
- Step 2: Transaction API
- Step 3: Verification component that implements the interface behavior
Step 1: Transaction Interface = Record

```vhdl
type AddressBusRecType is record
  Rdy : RdyType;
  Ack : AckType;
  Operation : AddressBusOperationType;
  Address : std_logic_vector_max_c;
  AddrWidth : integer_max;
  DataToModel : std_logic_vector_max_c;
  DataFromModel : std_logic_vector_max_c;
  DataWidth : integer_max;
  ...
end record AddressBusRecType;
```

- The record is an "inout" port
- The "magic" is in the types and resolution functions (from ResolutionPkg)

Long term plan is to migrate to VHDL-2019 Interfaces
- Only requires a mode view declaration for the record
Step 2: Transaction API = VHDL Procedure

```
procedure Read (  
  signal TransRec : InOut AddressBusRecType ;  
  iAddr  : In  std_logic_vector  ;  
  variable oData   : Out std_logic_vector ;  
  StatusMsgOn : In    boolean := FALSE  
) is  
begin  
  -- Put Transaction into Record  
  TransRec.Operation <= READ_OP ;  
  TransRec.Address <= SafeResize(iAddr, TransRec.Address'length) ;  
  TransRec.AddrWidth <= iAddr'length ;  
  TransRec.DataWidth <= oData'length ;  
  TransRec.StatusMsgOn <= StatusMsgOn ;  
  -- Handshake with Verification Component  
  RequestTransaction(Rdy => TransRec.Rdy, Ack => TransRec.Ack) ;  
  -- Get Results  
  oData := SafeResize(TransRec.DataFromModel, oData'length) ;  
end procedure Read ;
```
### Step 3: Verification Components

```vhdl
entity Axi4Manager is

    generic ( 
    tperiod_Clk : time := 10 ns ;
    . . .
    tpd_Clk_RReady : time := 2 ns 
    );

    port ( 
    -- Globals
    Clk : in std_logic ;
    nReset : in std_logic ;

    -- AXI Master Functional Interface
    AxiBus : inout Axi4RecType ;

    -- Testbench Transaction Interface
    TransRec : inout AddressBusRecType
    );

```
Step 3: Verification Components

TransactionHandler : process
begin

   WaitForTransaction(
      Clk => Clk,
      Rdy => TransRec.Rdy,
      Ack => TransRec.Ack
   );

   -- Decode and execute the transaction
   case TransRec.Operation is
      when WRITE_OP =>
         AxiWrite(TransRec.Address, TransRec.Data, ...);
      when READ_OP =>
         AxiRead (TransRec.Address, TransRec.Data, ...);
      when . . . =>
         -- Other Transactions
   end case;

end process TransactionHandler;

Benefit: Coding OSVVM VC is well within the capabilities of any VHDL engineer
Simplifying VC Development

- Observation: Some interfaces do the same transactions
  - Address Bus Interfaces (AXI4, Avalon, ...) do read and write
  - Streaming Interfaces (AxiStream, UART, ...) do send and get

- For these interfaces, Model Independent Transactions (MIT) define
  - Transaction Interface (record) and Transaction API (procedures)

- ... Address Bus MIT (basic subset)

```pascal
type AddressBusRecType is record . . .;
Write(AddrRec, iAddr, iData);
Read (AddrRec, X"1111_1110", oData);
. . .
```

- ... Stream MIT (basic subset)

```pascal
type StreamRecType is record . . .;
Send (TxRec, iData [, iParam]);
Get (RxRec, oData [, oParam]);
. . .
```
Benefits of OSVVM MIT

• Accelerates VC Development, Test Writing, and Documentation
  • Verification Component Developers
    • Re-use the transaction interface and API defined by MIT
    • Focus on writing VC behavior
  • Test Writers
    • Similar VC use the same transaction API
    • Similar VC can share sequences of transactions
  • Co-Simulation Interface
    • Supports all MIT based VC - including ones you write
  • Documentation
    • VC only need to identify which transactions they support
• More Information is in user guides in OsvvmLibraries/Documentation/
  • Address_Bus_Model_Independent_Transactions_user_guide.pdf
  • Stream_Model_Independent_Transactions_user_guide.pdf
entity TestCtrl is

port (
    TxRec    : InOut StreamRecType;
    RxRec    : InOut StreamRecType;
    ManagerRec : InOut AddressBusRecType;
    nReset   : In std_logic
);
end TestCtrl;
architecture UartTx1 of TestCtrl is

begin

ControlProc : process begin

....

WaitForBarrier(TestDone, 5 ms); EndOfTestReports; std.env.stop;
end process;

AxiManagerProc : process begin

wait until nReset = '1'; Write(. . .); WaitForBarrier(TestInit);

....

WaitForBarrier(TestDone); end process;

TxProc : process begin

....

WaitForBarrier(TestInit); Send(. . .);

....

WaitForBarrier(TestDone); end process;

....

Aspects of a Test Sequencer

• Whole test in one file
• Control Process
  • Initialize & finalize test
• One process per interface
  • Concurrent, just like design
• Tests =
  • Calls to transactions
• Easy to add and mix in
  • Directed Tests
• Constrained Random
• Scoreboards
• Functional Coverage
• Synchronization
• Error Reporting & Messaging
Test Initialization in ControlProc

ControlProc : process
begin
    SetTestName("UartRx1");

    TranscriptOpen;
    SetTranscriptMirror(TRUE);

    TBID <= NewID("TB");
    RxID <= NewID("UartRx_1");
    SB <= NewID("SB", ModelID);

    SetLogEnable(PASSED, TRUE);
    SetLogEnable(RxID, INFO, TRUE);

    WaitForBarrier(TestDone, 5 ms);

    . . . -- Test Finalization
OSVVM Makes Writing Tests Easy

- Call transactions such as Send, Get, and Check

\[
\begin{align*}
\text{TxProc} & : \text{process} \\
& \text{begin} \\
& \quad \text{Send} (\text{TxRec}, \text{X"10"}) \; ; \\
& \quad \text{Send} (\text{TxRec}, \text{X"11"}) \; ; \\
& \quad \cdots \\
& \quad \text{WaitForBarrier(...)} \\
& \text{end process TxProc} ;
\end{align*}
\]

\[
\begin{align*}
\text{RxProc} & : \text{process} \\
& \text{variable RxD : ByteType} \\
& \text{begin} \\
& \quad \text{Get}(\text{RxRec}, \text{RxD}) \\
& \quad \text{AffirmIfEqual}(\text{TBID}, \text{RxD}, \text{X"10"}); \\
& \quad \text{Check}(\text{RxRec}, \text{X"11"}); \\
& \quad \cdots \\
& \quad \text{WaitForBarrier(\text{TestDone})} \\
& \text{end process RxProc} ;
\end{align*}
\]

- Test Output for AffirmIfEqual

\[
\begin{align*}
\%\% & \text{ Log } \text{ PASSED In TB, Received: 10 at 2150 ns} \\
\%\% & \text{ Alert ERROR In TB, Received: 08 }=\text{ Expected: 10 at 2150 ns}
\end{align*}
\]

Benefit: Improves readability. Simplifies writing self-checking tests.
OSVVM Makes Randomization Easy

• Why Randomize?
  • Directed test of a FIFO (tracking words in FIFO):

<table>
<thead>
<tr>
<th>FifoWordCount</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadEmpty</td>
<td>0</td>
</tr>
<tr>
<td>ReadEmptyCovCnt</td>
<td>10</td>
</tr>
<tr>
<td>WriteFull</td>
<td>0</td>
</tr>
<tr>
<td>WriteFullCovCnt</td>
<td>10</td>
</tr>
</tbody>
</table>

• Constrained Random test of a FIFO:

<table>
<thead>
<tr>
<th>FifoWordCount</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadEmpty</td>
<td>0</td>
</tr>
<tr>
<td>ReadEmptyCovCnt</td>
<td>10</td>
</tr>
<tr>
<td>WriteFull</td>
<td>0</td>
</tr>
<tr>
<td>WriteFullCovCnt</td>
<td>10</td>
</tr>
</tbody>
</table>

• Key Benefits:
  • Generates realistic stimulus in a timely fashion (to write)
  • Ideal for large variety of similar items
    • Modes, sequences, network packets, processor instructions, ...
OSVVM Makes Randomization Easy

- Randomize a value in an inclusive range, 0 to 15, except 5 & 11

  Data1 := RV.RandInt(Min => 0, Max => 15) ;
  Data2 := RV.RandInt(0, 15, Exclude => (5,11) ) ;

- Randomize a value within the set (1, 2, 3, 5, 7, 11), except 5 & 11

  Data3 := RV.RandInt( (1,2,3,5,7,11) ) ;
  Data4 := RV.RandInt( (1,2,3,5,7,11), Exclude => (5,11) ) ;

- Weighted Randomization: Weight, Value = 0 .. N-1

  Data5 := RV.DistInt ( (7, 2, 1) ) ;

- Weighted Randomization: Value + Weight

  . . . -- (((val1, wt1), (val2, wt2), ...)
  Data6 := RV.DistValInt( ((1,7), (3,2), (5, 1)) ) ;

By itself, this is not constrained random
OSVVM Constrained Random

= Code Pattern + Randomization + Transaction Calls

TxProc : process
  variable RV : RandomPType;
  ...
for I in 1 to 10000 loop
  case RV.DistInt( (70, 10, 10, 5, 5) ) is
    when 0  =>  -- Nominal case  70%
      Operation := UARTTB_NO_ERROR;
      TxD := RV.RandS1v(0, 255, Data'length);
    when 1  =>  -- Parity Error  10%
      Operation := UARTTB_PARITY_ERROR;
      TxD := RV.RandS1v(1, 255, Data'length);
    when 2  =>  -- Stop Error   10%
      Operation := UARTTB_STOP_ERROR;
      TxD := RV.RandS1v(1, 255, Data'length);
    when . . .  -- (3 and 4)
      Operation := UARTTB_NO_ERROR;
      TxD := RV.RandS1v(0, 255, Data'length);
  end case;
  Send(TxRec, TxD, Operation);
end loop;
...
Constrained Random and Checking?

For checking, RxProc could repeat the randomization from TxProc, however, this is tedious and potentially error prone.

TxProc : process
  variable TxD : ByteType;
  variable RV : RandomPType;
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      ... 
    end case ;
    Send(TxRec, TxD, Op);
  end loop ;
  ... 
  WaitForBarrier(TestDone);
end process TxProc ;

RxProc : process
  variable ExpD : ByteType;
  variable RV : RandomPType;
begin
  for I in 1 to 10000 loop
    case RV.DistInt((. . .)) is
      ... 
    end case ;
    Check(TxRec, ExpD, Op);
  end loop ;
  ... 
  WaitForBarrier(TestDone);
end process RxProc ;
Scoreboards

- Simplify self-checking when data is minimally transformed

- Internally it is a FIFO + Checker
- Uses package generics to support different types
- Handles small data transformations
- Handles out of order execution
- Handles dropped values
Using OSVVM's Scoreboard is Easy

```vhdl
use osvvm.ScoreboardPkg_slv.all;
signal SB : ScoreboardIDType;

SB <= NewID("SB", ModelID);  -- Constructor in ControlProc
```

TxProc : process
begin
    for I in 1 to 10000 loop
        case RV.DistInt(...) is
            ...
        end case;
        Push(SB,(TxD, Op));
        Send(TxRec, TxD, Op);
    end loop;
end loop;

RxProc : process
begin
    for I in 1 to 10000 loop
        Get(RxRec, RxD, RxOp);
        Check(SB,(RxD, RxOp));
    end loop;
end loop;

WaitForBarrier(TestDone);

OSVVM's Data Structures (SB, FIFO, FC, and Alerts) use singletons
- Singletons use ordinary types and constructors (NewID)
- Easier than our older methodology which uses protected types.
Functional Coverage

- **What:** Code that tracks that items in the test plan occur
  - Tracks requirements, features, and boundary conditions

- **Why?**
  - With Randomization, how do you know what the test did?
  - Test Done = Functional Coverage and Code Coverage @ 100 %

- **Item Coverage (aka Point Coverage)**
  - Track relationships within a single object
  - Bin transfer sizes into: 1, 2, 3, 4-127, 128-252, 253, 254, 255

- **Cross Coverage**
  - Track relationships between independent objects
  - Has each set of registers been used with each input of an ALU?

- **Why not just use code coverage?**
  - Code coverage tracks code execution
  - Misses anything not in code (bins, uncorrelated items)
OSVVM Functional Coverage is Easy

architecture CR_1 of TestCtrl is

signal RxCov : CoverageIdType ;

RxProc : process

begin

RxCov <= NewID("RxCov", TB_ID) ;
wait for 0 ns ;

AddBins(RxCov, GenBin(1) ) ; -- Normal
AddBins(RxCov, GenBin(3) ) ; -- Parity Error
AddBins(RxCov, GenBin(5) ) ; -- Stop Error
AddBins(RxCov, GenBin(7) ) ; -- Parity + Stop
AddBins(RxCov, GenBin(9, 15, 1) ) ; -- Break

for I in 1 to 10000 loop

Get(RxRec, RxD, RxOp);
Check(SB,(RxD, RxOp));

ICover(RxCov, to_integer(RxOp));
end loop ;
...

Collect Coverage

Functional Coverage with OSVVM is as simple and concise as language syntax.
OSVVM Intelligent Coverage Randomization

= Randomize Using Functional Coverage Holes

TxProc : process
    variable StimCov : CoverageIdType ;
begin
    StimCov := NewID("StimCov", TB_ID) ;
    wait for 0 ns ;
    AddBins(StimCov, "NORMAL", 7000, GenBin(1) ) ;
    AddBins(StimCov, "PARITY", 1000, GenBin(3) ) ;
    AddBins(StimCov, "STOP", 1000, GenBin(5) ) ;
    . . .
for I in 1 to 10000 loop
    iOperation := GetRandPoint(StimCov) ;
    case iOperation is
        when 1  =>    . . . -- Nominal 70%
        when 3  =>    . . . -- Parity  10%
        . . .
    end case ;
    Push(SB, (Data, Operation) ) ;
    Send(TxRec, Data, Operation) ;
    ICover(StimCov, iOperation ) ;
    wait for Idle * UART_BAUD_PERIOD_115200 ;
end loop ;

Intelligent Coverage goes beyond what SV does
OSVVM Protocol and Parameter Checkers

• Protocol Check: OE and WE of a RAM never asserted simultaneously

SimultaneousAccessCheck: process
begin
  wait on iCE, iWE, iOE ;
  AlertIf(SramAlertID, (iCE and iWE and iOE) = '1',
         "nCE, nWE, and nOE are all active") ;
end process SimultaneousAccessCheck ;

• Alerts signal errors and keep counts in the AlertLog data structure
• Alert Levels: FAILURE, ERROR (default), WARNING

• Controls: StopCount, PrintCount, Enable/Disable

  SetAlertStopCount(ERROR, 20) ;          -- Stop when 20
  SetAlertPrintCount(CpuID, ERROR, 10) ;   -- Limit printing
  SetAlertEnable(WARNING, FALSE) ;        -- Disable Alerts

• Alerts are enabled by default and rarely disabled
OSVVM Logs Simplify Debug

• Logs are conditional printing

```vhdl
TxProc : process
begin
  ...
  Log(TbID, "Sequence 1 Starting", ALWAYS) ;
  ...
  Log(TbID, "Test Last Failed Here", DEBUG) ; -- Disabled
```

• Log Levels: ALWAYS (default), DEBUG, INFO, FINAL, PASSED
• Logs only print when enabled.

• Controls: Enable/Disable

```vhdl
SetLogEnable(DEBU DEBUG, FALSE) ; -- Disable Alerts
```

• Log output for above

```text
%% Log  ALWAYS In TB, Sequence 1 Starting at 2200 ns
```

• Message with level DEBUG does not print since it is disabled
Test Finalization

ControlProc : process
begin
  SetTestName("UartRx1") ;

  . . . - Test Initialization

  WaitForBarrier(TestDone, 5 ms) ; Stop until Test Done

  AlertIf(TBID, NOW >= 5 ms, "Test timed out");

  AlertIf(TBID, not Empty(SB), "Scoreboard not empty");

  AlertIf(TBID, GetAffirmCount < 1, "Checked < 1 items");

  AffirmIfNotDiff("UartRx1.log", "Checked/UartRx1.log"); Create Reports

  EndOfTestReports ;
  std.env.stop ;
  wait ;
end process ControlProc
OSVVM Test Watch Dog

• **Purpose:** Stop a process until all processes have reached the barrier

```vhdl
signal TestDone : integer_barrier := 1;
```

ControlProc : process begin begin
  SetTestName("UartRx1") ;
  . .
  WaitForBarrier(TestDone, 5 ms);
  . .
  ReportAlerts ;
  std.env.stop(GetAlertCount) ;
end process ControlProc ;

TestProc1 : process begin
  . .
  WaitForBarrier(TestDone) ;
  wait ;

TestProc2 : process begin
  . .
  WaitForBarrier(TestDone) ;
  wait ;

• **Benefit**
  • With "TestDone", simulator scripts do not need to know run length
  • The 5 ms is a time out – aka watch dog timer on the test
OSVVM Scripting

- Procedure based API that runs on top of TCL

library osvvm_TbUart

analyze ./testbench/TestCtrl_e.vhd
analyze ./testbench/TbUart.vhd

analyze ./testbench/TbUart_SendGet1.vhd
simulate TbUart_SendGet1

• Benefits
  - Simple, just like a list of source files ...
  - ... Except we also get the power of TCL
  - Settings (like current working library) are remembered
  - Paths are relative to script location to facilitate moving pieces of projects

• One Simulator Script API for
  - GHDL, NVC, Aldec, Siemens, Synopsys VCS, Cadence Xcelium
OSVVM Reports

- OSVVM Test Completion Message
- Build Summary Mini-Report – Text
- Build Summary Report – HTML
  - Summary of entire build
  - Summary of each test suite in the build
  - Summary of each test case within a test suite
- Test Case Reports – HTML
  - Reports on Alert, Functional Coverage, and Scoreboards
  - Links to HTML simulation transcript and test file output
- HTML'ized simulation transcript / log file (simulator output)
  - Errors shown in red
- JUnit Report – XML for Continuous Integration Tools
  - Similar to the build summary report with less information
OSVVM Test Completion Message

- EndOfTestReports produces a summary and if errors a detailed report

EndOfTestReports ;

%% DONE  PASSED  Test_UartRx_1  Passed: 48  Affirmations Checked: 48  at 100100100 ns

%% DONE  FAILED  Test_UartRx_1  Total Error(s) = 7  Failures: 0  Errors: 7  Warnings: 0  Passed: 41  Affirmations Checked: 48  at 100100100 ns

%% Default  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% OSVVM  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% TB  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% UART_SB  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% AxiMaster_1  Failures: 0  Errors: 7  Warnings: 0  Passed: 0
%%   AxiMaster_1 Data Err  Failures: 0  Errors: 7  Warnings: 0  Passed: 41
%%   AxiMaster_1 Protocol  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% UartTx_1  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%% UartRx_1  Failures: 0  Errors: 0  Warnings: 0  Passed: 0
Build Summary Mini-Report

- When a build finishes, a single line, mini report is produced

BuildError: Sim_Demo FAILED, Passed: 149, Failed: 3, Skipped: 0, Analyze Errors: 0, Simulate Errors: 0, Build Error Code: 0

- If there are errors, this is the first place we see an indication
## Build Summary Report

**Created by Scripts + EndOfTestReports**

### Build Status
Summarizes all tests run
- Link to Simulation Transcript
  - Both text and html
- Link to code coverage

### Test Suite Summaries
- Test Suite = multiple test cases
- Summarizes Pass/Fail+

### Test Case Summaries
- Test Case = Testbench
- Identify Failing Test(s)
- Links to Test Case Reports

---

### OsvvmLibraries_RunAllTestsWithCoverage Build Summary Report

<table>
<thead>
<tr>
<th>Build</th>
<th>OsvvmLibraries_RunAllTestsWithCoverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>PASSED</td>
</tr>
<tr>
<td>PASSED</td>
<td>389</td>
</tr>
<tr>
<td>FAILED</td>
<td>0</td>
</tr>
<tr>
<td>SKIPPED</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Elapsed Time (h:m:s):** 01:16:15
- **Elapsed Time (seconds):** 3233.578
- **Start Time:** 2023-02-01T16:15:000
- **Simulator:** RivieraPRO
- **Simulator Version:** RivieraPRO-2022.04.117.8517
- **OSVVM Version:** 2023.01
- **Simulation Transaction:** OsvvmLibraries_RunAllTestsWithCoverage
- **HTML Simulation Transcript:** OsvvmLibraries_RunAllTestsWithCoverage.log
- **Code Coverage:** CodeCoverageResults

---

### OsvvmLibraries_RunAllTestsWithCoverage Test Suite Summary

<table>
<thead>
<tr>
<th>TestSuite</th>
<th>Status</th>
<th>PASSED</th>
<th>FAILED</th>
<th>SKIPPED</th>
<th>Requirements passed / goal</th>
<th>Disabled Alerts</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamTransactionPkg</td>
<td>PASSED</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>104,213</td>
</tr>
<tr>
<td>StreamTransactionArrayPkg</td>
<td>PASSED</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>85,822</td>
</tr>
<tr>
<td>AddressbusTransactionPkg</td>
<td>PASSED</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>308,199</td>
</tr>
<tr>
<td>AddressbusTransactionArrayPkg</td>
<td>PASSED</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>369,189</td>
</tr>
<tr>
<td>InterruptHandler_Generic</td>
<td>PASSED</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>65,216</td>
</tr>
<tr>
<td>AddrBus</td>
<td>PASSED</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>113,467</td>
</tr>
<tr>
<td>AddrFull</td>
<td>PASSED</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>651,499</td>
</tr>
<tr>
<td>AddrQueue</td>
<td>PASSED</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>439,512</td>
</tr>
<tr>
<td>Unit</td>
<td>PASSED</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>75,655</td>
</tr>
<tr>
<td>Debar</td>
<td>PASSED</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>8,386</td>
</tr>
<tr>
<td>Ethernet</td>
<td>PASSED</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>38,852</td>
</tr>
<tr>
<td>AddrBus_VI</td>
<td>PASSED</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>409,860</td>
</tr>
<tr>
<td>AddrQueue_VI</td>
<td>PASSED</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>395,580</td>
</tr>
</tbody>
</table>

---

### StreamTransactionPkg Test Case Summary

### StreamTransactionArrayPkg Test Case Summary

### AddressbusTransactionPkg Test Case Summary

### AddressbusTransactionArrayPkg Test Case Summary

### Test Case Details

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Checks checked</th>
<th>Errors</th>
<th>Requirements passed / goal</th>
<th>Functional Coverage</th>
<th>Disabled Alerts</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUse12_TransactionUtilManager</td>
<td>PASSED</td>
<td>41 / 41</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.365</td>
</tr>
<tr>
<td>TUse12_MemLogObjectManager</td>
<td>PASSED</td>
<td>8 / 8</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.096</td>
</tr>
<tr>
<td>TUse12_ReleaseObjectManager1</td>
<td>PASSED</td>
<td>38 / 38</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>3.879</td>
</tr>
<tr>
<td>TUse12_MultiObjectManager</td>
<td>PASSED</td>
<td>0 / 0</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>3.868</td>
</tr>
<tr>
<td>TUse12_ReadWrite</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>3.854</td>
</tr>
<tr>
<td>TUse12_MemReadWrite1</td>
<td>PASSED</td>
<td>40 / 40</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.014</td>
</tr>
<tr>
<td>TUse12_MemReadWrite2</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.018</td>
</tr>
<tr>
<td>TUse12_ReadPoll</td>
<td>PASSED</td>
<td>28 / 28</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.163</td>
</tr>
<tr>
<td>TUse12_WritePoll</td>
<td>PASSED</td>
<td>28 / 28</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>4.163</td>
</tr>
</tbody>
</table>
# Build Summary Report with Errors

**sim_OsvvmDemo Build Summary Report**

<table>
<thead>
<tr>
<th>Build</th>
<th>sim_OsvvmDemo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FAILED</td>
</tr>
<tr>
<td>PASSED</td>
<td>149</td>
</tr>
<tr>
<td>FAILED</td>
<td>3</td>
</tr>
<tr>
<td>SKIPPED</td>
<td>0</td>
</tr>
<tr>
<td>Analyze Failures</td>
<td>0</td>
</tr>
<tr>
<td>Simulate Failures</td>
<td>0</td>
</tr>
<tr>
<td>Elapsed Time (hours)</td>
<td>0:42:39</td>
</tr>
<tr>
<td>Elapsed Time (seconds)</td>
<td>2556.991</td>
</tr>
<tr>
<td>Start Time</td>
<td>2023-05-07T16:20:0700</td>
</tr>
<tr>
<td>Simulator</td>
<td>QuestaSim -gui</td>
</tr>
<tr>
<td>Simulator Version</td>
<td>QuestaSim-2022.01</td>
</tr>
<tr>
<td>OSVVM Version</td>
<td>2023.04</td>
</tr>
<tr>
<td>Simulation Transcript</td>
<td>sim_OsvvmDemo.log</td>
</tr>
<tr>
<td>HTML Simulation Transcript</td>
<td>sim_OsvvmDemo_log.html</td>
</tr>
<tr>
<td>Code Coverage</td>
<td>Code Coverage Results</td>
</tr>
<tr>
<td>Finish Time</td>
<td>2023-05-07T17:02:0700</td>
</tr>
</tbody>
</table>

**Test Suites Summary**

<table>
<thead>
<tr>
<th>TestSuites</th>
<th>Status</th>
<th>PASSED</th>
<th>FAILED</th>
<th>SKIPPED</th>
<th>Requirements passed / goal</th>
<th>Disabled Alerts</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AxiLite</td>
<td>PASSED</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>197.229</td>
</tr>
<tr>
<td>Ax4Full</td>
<td>FAILED</td>
<td>59</td>
<td>3</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>1042.115</td>
</tr>
<tr>
<td>Ax4Stream</td>
<td>PASSED</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>997.658</td>
</tr>
<tr>
<td>Uart</td>
<td>PASSED</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>153.148</td>
</tr>
<tr>
<td>DpBram</td>
<td>PASSED</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>8.976</td>
</tr>
<tr>
<td>Ethernet</td>
<td>PASSED</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
<td>0</td>
<td>113.837</td>
</tr>
</tbody>
</table>

**AxiLite Test Case Summary**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
<th>Checks passed / checked</th>
<th>Errors</th>
<th>Requirements passed / goal</th>
<th>Functional Coverage</th>
<th>Disabled Alerts</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TlAx4_DemoMemoryReadWrite1</td>
<td>PASSED</td>
<td>334 / 334</td>
<td>0</td>
<td>0 / 0</td>
<td>43.75</td>
<td>0</td>
<td>12.847</td>
</tr>
<tr>
<td>TlAx4_DemoMemoryReadWrite2</td>
<td>FAILED</td>
<td>300 / 302</td>
<td>2</td>
<td>0 / 0</td>
<td>100.00</td>
<td>0</td>
<td>3.378</td>
</tr>
<tr>
<td>TlAx4_BasicReadWrite</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>15.823</td>
</tr>
<tr>
<td>TlAx4_RandomReadWrite</td>
<td>PASSED</td>
<td>3000 / 3000</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>16.387</td>
</tr>
<tr>
<td>TlAx4_RandomReadWrite64x1</td>
<td>PASSED</td>
<td>30000 / 3000</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>13.892</td>
</tr>
<tr>
<td>TlAx4_SubordinateReadWrite1</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>12.095</td>
</tr>
<tr>
<td>TlAx4_SubordinateReadWrite2</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>11.955</td>
</tr>
<tr>
<td>TlAx4_SubordinateReadWrite3</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>11.529</td>
</tr>
<tr>
<td>TlAx4_ReadWriteAsync1</td>
<td>PASSED</td>
<td>60 / 60</td>
<td>0</td>
<td>0 / 0</td>
<td>-</td>
<td>0</td>
<td>11.919</td>
</tr>
</tbody>
</table>
Test Case Report

TbAxi4_MemoryReadWriteDemo1 Test Case Detailed Report

Available Reports

- Alert Report
- Functional Coverage Report(s)
- ScoreboardReport(s)
- Link to Simulation Results
- TbAxi4_MemoryReadWriteDemo1.txt
- Osymlibraries_RunAllTestsWithCoverage.txt

Links

- Alert Report
- Functional Coverage Report
- Scoreboard Reports
- Simulation Results
- Test Case Transcript
- Link to Build Summary

TbAxi4_MemoryReadWriteDemo1 Alert Report

- TbAxi4_MemoryReadWriteDemo1 Alert Settings
- TbAxi4_MemoryReadWriteDemo1 Alert Results

TbAxi4_MemoryReadWriteDemo1 Coverage Report

Total Coverage: 43.75

- Cov1 Coverage Model: Coverage: 37.5
- Cov2 Coverage Model: Coverage: 37.5
- Cov1b Coverage Model: Coverage: 50.0
- Cov2b Coverage Model: Coverage: 50.0

TbAxi4_MemoryReadWriteDemo1 Scoreboard Report for Scoreboard_slv

<table>
<thead>
<tr>
<th>Name</th>
<th>ParentName</th>
<th>ItemCount</th>
<th>ErrorCount</th>
<th>ItemChecked</th>
<th>ItemsPopped</th>
<th>ItemsDropped</th>
<th>FifoCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteAddressFIFO</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteDataFIFO</td>
<td>memory_1</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteResponseFIFO</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadAddressFIFO</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadDataFIFO</td>
<td>memory_1</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteResponseScoreboard</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadResponseScoreboard</td>
<td>manager_1</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteAddressFIFO</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteDataFIFO</td>
<td>manager_1</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadAddressFIFO</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadDataFIFO</td>
<td>manager_1</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Scoresboard Report

- One Table for each Scoreboard type.
- One row in table for each scoreboard.
Test Case Report: Alert Report with Errors

TbAxi4_DemoErrorMemoryReadWrite1 Alert Report

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FailOnWarning</td>
<td>true</td>
<td>If true, warnings are a test error</td>
</tr>
<tr>
<td>FailOnDisabledErrors</td>
<td>true</td>
<td>If true, Disabled Alert Counts are a test error</td>
</tr>
<tr>
<td>FailOnRequirementErrors</td>
<td>true</td>
<td>If true, Requirements Errors are a test error</td>
</tr>
<tr>
<td>External Errors</td>
<td>0</td>
<td>Added to Alert Counts in determine total errors</td>
</tr>
<tr>
<td>External Warnings</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Expected Errors</td>
<td>0</td>
<td>Subtracted from Alert Counts in determine total errors</td>
</tr>
<tr>
<td>Expected Warnings</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Status FAILED in table indicates TbAxi4_DemoErrorMemoryReadWrite1 has 2 errors. The 2 errors were detected in the manager_1 VC. One error is in the manager_1::Data Check. One error is in manager1::ReadBurstFifo.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Checks Passed</th>
<th>Checks Total</th>
<th>Total Errors</th>
<th>Alert Counts Failures</th>
<th>Alert Counts Errors</th>
<th>Alert Counts Warnings</th>
<th>Requirements Passed</th>
<th>Requirements Checked</th>
<th>Requirements Failures</th>
<th>Requirements Errors</th>
<th>Disabled Alert Counts Failures</th>
<th>Disabled Alert Counts Errors</th>
<th>Disabled Alert Counts Warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>TbAxi4_DemoErrorMemoryReadWrite1</td>
<td>FAILED</td>
<td>300</td>
<td>302</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Default</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OSVVM</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Testbench</td>
<td>PASSED</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>manager_1</td>
<td>FAILED</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Protocol Error</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Data Check</td>
<td>FAILED</td>
<td>15</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No response</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteResponse Scoreboard</td>
<td>PASSED</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadResponse Scoreboard</td>
<td>PASSED</td>
<td>134</td>
<td>134</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteBurstFifo</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadBurstFifo</td>
<td>FAILED</td>
<td>91</td>
<td>92</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>memory_1</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No response</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Data Check</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>memory_1:memory</td>
<td>PASSED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Test Case Report: Functional Coverage

Uart7_Random_part3 Coverage Report

Total Coverage: 100.00

- UART_RX_STIM_COV Coverage Model
  Coverage: 100.0
  - UART_RX_STIM_COV Coverage Settings

<table>
<thead>
<tr>
<th>CovWeight</th>
<th>Goal</th>
<th>WeightMode</th>
<th>Seeds</th>
<th>CountMode</th>
<th>IllegalMode</th>
<th>Threshold</th>
<th>ThresholdEnable</th>
<th>TotalCovCount</th>
<th>TotalCovGoal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>at_least</td>
<td>824213985, 792842968</td>
<td>count_first</td>
<td>illegal_on</td>
<td>45.0</td>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

- UART_RX_STIM_COV Coverage Bins

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Mode</th>
<th>Data</th>
<th>Idle</th>
<th>Count</th>
<th>AtLeast</th>
<th>Percent Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>Count</td>
<td>1 to 1</td>
<td>0 to 255</td>
<td>0 to 0</td>
<td>63</td>
<td>63</td>
<td>100.0</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Count</td>
<td>1 to 1</td>
<td>0 to 255</td>
<td>1 to 15</td>
<td>7</td>
<td>7</td>
<td>100.0</td>
</tr>
<tr>
<td>PARITY</td>
<td>Count</td>
<td>3 to 3</td>
<td>0 to 255</td>
<td>2 to 15</td>
<td>11</td>
<td>11</td>
<td>100.0</td>
</tr>
<tr>
<td>STOP</td>
<td>Count</td>
<td>5 to 5</td>
<td>1 to 255</td>
<td>2 to 15</td>
<td>11</td>
<td>11</td>
<td>100.0</td>
</tr>
<tr>
<td>PARITY_STOP</td>
<td>Count</td>
<td>7 to 7</td>
<td>1 to 255</td>
<td>2 to 15</td>
<td>11</td>
<td>11</td>
<td>100.0</td>
</tr>
<tr>
<td>BREAK</td>
<td>Count</td>
<td>9 to 15</td>
<td>11 to 30</td>
<td>2 to 15</td>
<td>2</td>
<td>2</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Total Percent Coverage: 100.0

- UART_RX_COV Coverage Model
  Coverage: 100.0
  - UART_RX_COV Coverage Settings
  - UART_RX_COV Coverage Bins
**Test Case Report: Scoreboards**

**TbAxi4_DemoErrorMemoryReadWrite1 Scoreboard Report for Scoreboard_slv**

<table>
<thead>
<tr>
<th>Name</th>
<th>ParentName</th>
<th>ItemCount</th>
<th>ErrorCount</th>
<th>ItemsChecked</th>
<th>ItemsPopped</th>
<th>ItemsDropped</th>
<th>FifoCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteResponse Scoreboard</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadResponse Scoreboard</td>
<td>manager_1</td>
<td>134</td>
<td>0</td>
<td>134</td>
<td>134</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteAddressFifo</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteDataFifo</td>
<td>manager_1</td>
<td>134</td>
<td>0</td>
<td>0</td>
<td>134</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadAddressFifo</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadAddressTransactionFifo</td>
<td>manager_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadDataFifo</td>
<td>manager_1</td>
<td>134</td>
<td>0</td>
<td>0</td>
<td>134</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteAddressFIFO</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteDataFifo</td>
<td>memory_1</td>
<td>134</td>
<td>0</td>
<td>0</td>
<td>134</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteResponseFifo</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadAddressFifo</td>
<td>memory_1</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadDataFifo</td>
<td>memory_1</td>
<td>134</td>
<td>0</td>
<td>0</td>
<td>134</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WriteBurstFifo</td>
<td>manager_1</td>
<td>102</td>
<td>0</td>
<td>0</td>
<td>102</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadBurstFifo</td>
<td>manager_1</td>
<td>102</td>
<td>1</td>
<td>92</td>
<td>102</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- A separate table is created for each scoreboard instance
- Each row in the table has statistics for a single scoreboard
- Tables for Scoreboard_slv and Scoreboard_int are automatically generated
- Use WriteScoreboardYaml to generate reports for user created scoreboards
Errors are shown in red in the HTML'ized report

As a result, this report can be scanned for errors
Simulation Transcript

When viewed from Test Case Report, it jumps to the simulation's results.
Getting OSVVM & Running Scripts

- Get the sources:

  
  ```
  git clone --recursive https://github.com/osvvm/OsvvmLibraries
  ```

- Alternately, a zip file is at: osvvm.org/downloads

- Initialize the simulator – see Documentation/Scripts_user_guide.pdf

  ```
  file mkdir sim ;  # In directory containing OsvvmLibraries
cd sim
source ../OsvvmLibraries/Scripts/StartUp.tcl
  ```

- Build all OSVVM and Run All VC Tests

  ```
  build $OsvvmLibraries/OsvvmLibraries.pro
  build $OsvvmLibraries/RunAllTests.pro
  ```

- Each VC has a RunAllTests and RunDemoTests
All you need is ... OSVVM

- **Benefits**
  - Powerful and Concise – rivals other verification languages
  - Unmatched reuse through the entire verification process
  - Unmatched report capability with HTML for humans and JUnit XML for CI
  - Tests are Readable and Reviewable by All
  - Adopt incrementally as needed
  - Tests and VC can be written by any VHDL Engineer
SynthWorks VHDL Classes

Comprehensive VHDL Introduction  4 Days  - beginners class
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineer's introduction to VHDL syntax, RTL coding,
and testbenches. Students get VHDL hardware experience with our FPGA based
lab board.

Advanced VHDL Testbenches and Verification - OSVVM Boot Camp - 5 days
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction based modeling,
self-checking, scoreboards, memory modeling, functional coverage, directed,
algorithmic, constrained random, and intelligent testbench test generation. Create
a VHDL testbench environment that is competitive with other verification
languages, such as SystemVerilog or 'e'. Our techniques work on VHDL simulators
without additional licenses and are accessible to RTL engineers.

VHDL Coding for Synthesis  4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Learn VHDL RTL (FPGA and ASIC) coding styles, methodologies, design techniques,
problem solving techniques, and advanced language constructs to produce better,
faster, and smaller logic.
OSVVM Resources

- Documentation
  - HTML: https://osvvm.github.io/Overview/Osvvm1About.html
  - PDF: OsvvmLibraries/Documentation - in OSVVM release
- Forum: https://osvvm.org

- Recorded Webinars
  - OSVVM: Leading Edge Verification for the VHDL Community
    - https://www.youtube.com/watch?v=KVmGDy_PHNI
  - Faster than Lite Verification Component Development with OSVVM
  - OSVVM’s Test Reports and Simulator Independent Scripting
  - Advances in OSVVM’s Verification Data Structures

- Jump start your VHDL verification effort with training
  - Advanced VHDL Testbenches and Verification – OSVVM Boot Camp
    - https://synthworks.com/vhdl_testbench_verification.htm
Why OSVVM?

- OSVVM is VHDL's #1 Verification Methodology
- For FPGA Verification,
  - Worldwide: 28% use OSVVM = 50% of the VHDL FPGA users