Open Source VHDL Verification Methodology (OSVVM)

Leading Edge Verification for the VHDL Community

by

Jim Lewis

OSVVM Chief Architect

IEEE 1076 Working Group Chair

VHDL Training Expert at SynthWorks

Jim@SynthWorks.com

OSVVM: Leading Edge Verification

This material is derived from SynthWorks' class, VHDL Testbenches and Verification

This material is updated from time to time and the latest copy of this is available at
http://www.SynthWorks.com/papers

Contact Information
Jim Lewis, President
SynthWorks Design Inc
11898 SW 128th Avenue
Tigard, Oregon 97223
503-590-4787
jim@SynthWorks.com

www.SynthWorks.com

SynthWorks

OSVVM: Leading Edge Verification

- Agenda
  - VHDL is #1 for FPGA Design and Verification
  - What is OSVVM?
  - OSVVM Verification Framework
  - Verification Components
  - Self-Checking Tests Made Easy
  - Simplifying Test Printing with OSVVM Logs
  - Constrained Random Tests
  - Scoreboards
  - Functional Coverage & Intelligent Coverage Random
  - Protocol and Parameter Checks
  - Test Watch Dog Timers
  - Test Reporting

SynthWorks

Background

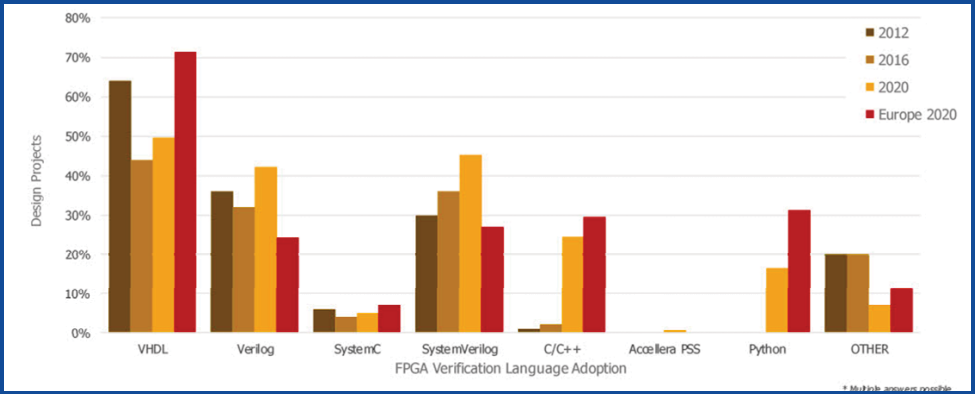
- About Jim Lewis
  - 30+ years: Design and Verification
  - 30 years: VHDL
  - 20+ years: Active in IEEE VHDL WGs
  - 14 years: IEEE VHDL WG chair
  - Chief Architect of OSVVM
  - VHDL Consultant and Trainer for SynthWorks

- Mission: Innovate simple solutions to challenging problems

SynthWorks

VHDL is #1 in FPGA Design & Verification

- From Wilson Research Group 2020 Functional Verification Survey
  - For FPGA design: 64% worldwide / Europe 84% use VHDL
  - For FPGA verification: 50% worldwide / Europe 72% use VHDL

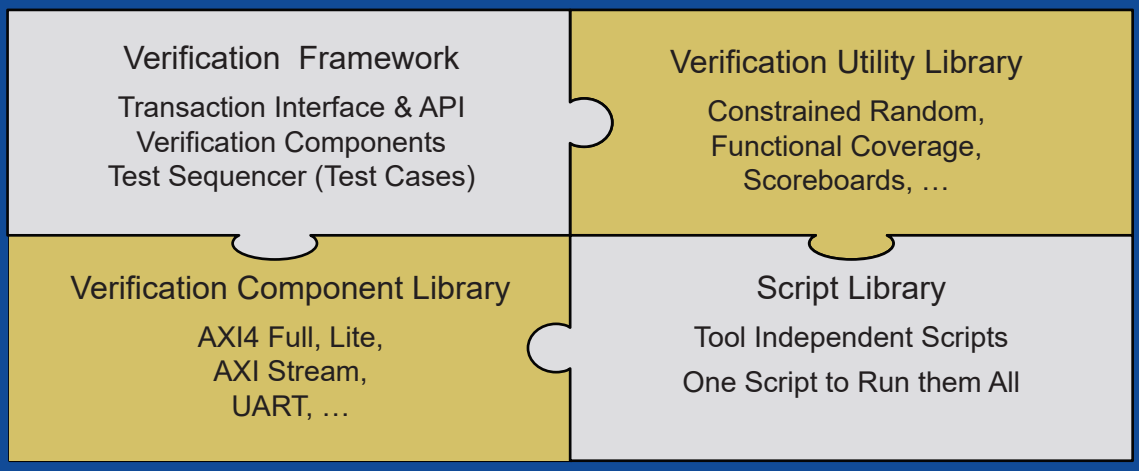


- For FPGA Verification Libraries,
  - Worldwide: 18% use OSVVM = 36% of the VHDL FPGA users
  - In Europe: 34% use OSVVM while only 26% use SV & UVM

- © Siemens 2020 https://blogs.sw.siemens.com/verificationhorizons/2020/12/16/part-6-the-2020-wilson-research-group-functional-verification-study/

SynthWorks

What is OSVVM?



- OSVVM Benefits
  - Simplifies writing Test Cases and Verification Components
  - Makes Test Cases Readable and Reviewable by All
  - Powerful reporting – HTML (for humans) and JUnit XML (for CI)
  - Works like built-in language features
  - Upgrade your current VHDL testbench and models incrementally

OSVVM History

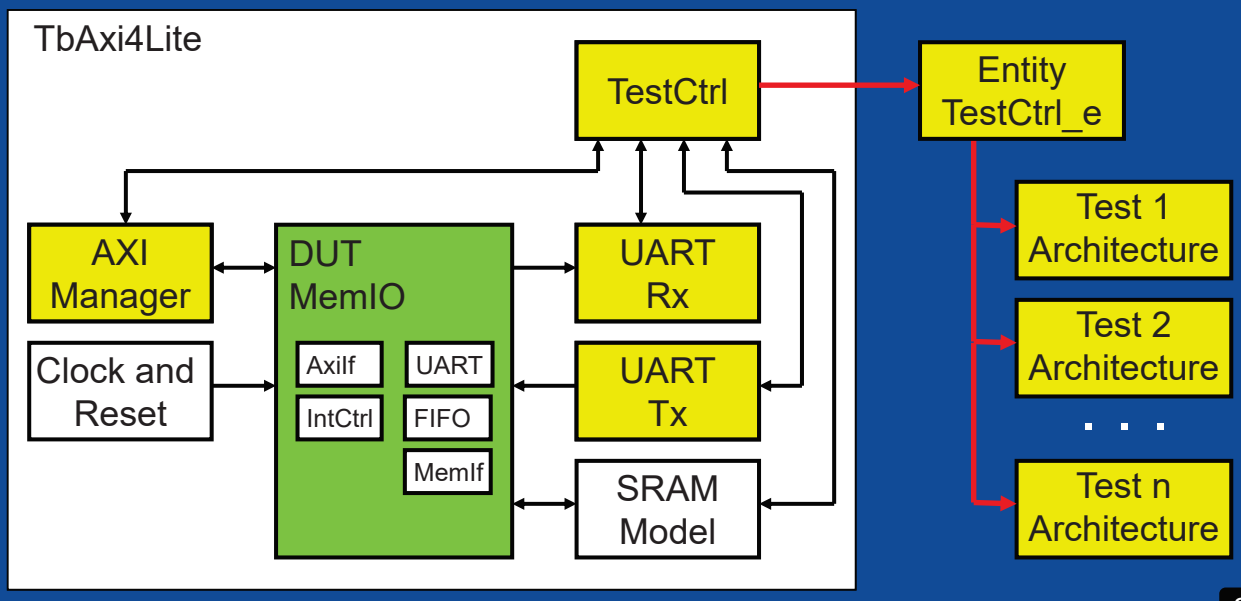
SynthWorks

	SynthWorks	OSVVM	
1997	Transaction Framework, TbUtilPkg		
2006	RandomPkg, ResolutionPkg, ScoreboardPkg, MemoryPkg		
2010	CoveragePkg		
2011		2.0	RandomPkg, CoveragePkg
2015		2.5	AlertLogPkg, TranscriptPkg, MemoryPkg
2016		3.0	ScoreboardGenericPkg, TbUtilPkg ResolutionPkg, VendorCovApi
2018		4.0	Axi4Lite, AxiStream, UART
2020		5.0	Scripting, Specification Tracking, MIT, Virtual Interfaces, Axi4 Full and AxiStream, both with Bursting
2021		6.0	Singleton Data Structures, HTML & JUnit XML reports

Verification Framework

SynthWorks

- OSVVM framework looks identical to a SystemVerilog framework:
  - Verification components implement interface signaling
  - Test case = sequences of transactions in test sequencer (TestCtrl)
  - Each test case is a separate architecture of TestCtrl



Verification Framework - Netlist

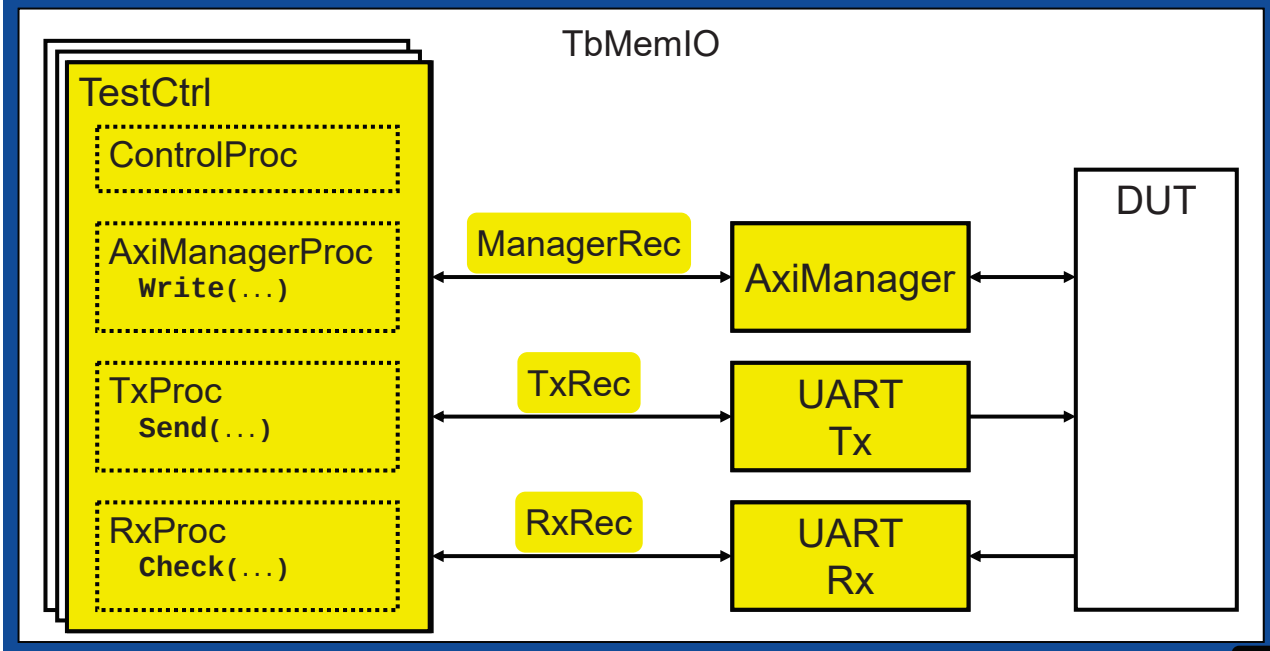
SynthWorks

```
library osvvm, osvvm_Axi4 ;
  context osvvm.OsvvmContext ;
. . .
entity  TbAxi4  is
end entity TbAxi4 ;
architecture Test1 of TbAxi4 is
  . . .
begin
  osvvm.TbUtilPkg.CreateClock(Clk, tperiod_Clk) ;
  osvvm.TbUtilPkg.CreateReset(Reset, . . .) ;

  DUT_1: DUT ( . . . ) ;

  Axi4Manager_1 : Axi4Manager (ManagerRec, . . . ) ;
  UartRx_1      : UartRx(RxRec, . . . ) ;
  UartTx_1      : UartTx(TxRec, . . . ) ;
  TestCtrl_1    : TestCtrl (ManagerRec, RxRec, TxRec, . . . ) ;
end Test1 ;
```


Verification Framework – Elements

SynthWorks

- Transaction Interface + Transaction API
- Verification components
- Test Sequencer (TestCtrl)



Transaction Interface = Record

SynthWorks

```
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 ;
```

Control / Handshaking

Transaction Information

- The record is an "inout" port, so each element is a resolved type
  - OSVVM package ResolutionPkg supports VHDL's common types

Long term plan is to migrate to VHDL-2019 Interfaces
- Only requires a view declaration for the record

Transaction Initiation API

SynthWorks

```
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 Model
  RequestTransaction(Rdy => TransRec.Rdy, Ack => TransRec.Ack) ;

  -- Get Results
  oData := SafeResize(TransRec.DataFromModel, oData'length) ;
end procedure Read ;
```

Independent of VC
- Put information for VC into record
- Handshake with VC
- Get results from VC from record

SynthWorks

Transaction Interface + Transaction API

- Observation: Some interfaces do the same transactions
  - Address Bus Interfaces (AXI4, Avalon, …) do read and write
  - Streaming Interfaces (AxiStream, UART, …) do send and get
- Address Bus Model Independent Transactions (MIT) – basics:

```
type AddressBusRecType is record  . . . ;

Write(     AddrRec, iAddr, iData [, StatusMsgOn]) ;
Read (     AddrRec, X"1111_1110", oData) ;
ReadCheck( AddrRec, X"AAAA_AAA0", X"55") ;
```

- Stream Model Independent Transactions (MIT) – basics:

```
type StreamRecType is record  . . . ;

Send (TxRec, iData [, iParam]) ;
Get  (RxRec, oData [, oParam]) ;
Check(RxRec, iData [, iParam]) ;
```

- Both support numerous additional transactions including bursting.

SynthWorks

Verification Components

```
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
) ;
```

DUT Interface

Transaction Interface

Verification Components

SynthWorks

```
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 ;
```

Find Transaction in Record

Do the Transaction

Simplified by OSVVM MIT – Interface and API already defined

TestCtrl = Test Sequencer

SynthWorks

```
entity TestCtrl is
  generic (
    tperiod_Clk         : time := 10 ns
  ) ;
  port (
    TxRec         : InOut StreamRecType ;
    RxRec         : InOut StreamRecType ;

    ManagerRec    : InOut AddressBusRecType ;

    nReset        : In    std_logic
) ;
end TestCtrl ;
```

Recommendation:
Keep TestCtrl entity in a separate file from the architecture(s).
Facilitates using multiple architectures.

```
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(CpuRdy);
    . . .
    WaitForBarrier(TestDone) ;
  end process ;

  TxProc : process
  begin
    WaitForBarrier(CpuRdy);
    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 write and read
- Easy to add
  - Directed Tests
  - Functional Coverage
  - Constrained Random
  - Mix of test approaches
- Synchronization
- Error Reporting & Messaging

Simplified by OSVVM MIT – API is the same for similar VC

ControlProc: Test Initialization

```
ControlProc : process
begin
  SetAlertLogName("UartRx1") ;

  TranscriptOpen("./results/UartRx1.txt") ;
  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
```

Set Test Name

Open Results File + Write to Console

IDs to control Messages & Errors independently

Enable Logs Message Filtering

Stop until Test Done or 5 ms has passed

NewID is overloaded constructor for OSVVM's singleton data structures.

SynthWorks

OSVVM Makes Self-Checking Tests Easy

- Send transaction on transmitter interface
- Get transaction on receiver interface and then check with AffirmIf

```
TxProc : process

begin
  Send (TxRec, X"10") ;

  Send (TxRec, X"11") ;
  . . .

  WaitForBarrier(...) ;
end process TxProc ;
```

```
RxProc : process
  variable RxD : ByteType;
begin
  Get(RxRec, RxD) ;
  AffirmIfEqual(TBID, RxD, X"10");

  Get(RxRec, RxD) ;
  AffirmIfEqual(TBID, RxD, X"11");

  . . .
  WaitForBarrier(TestDone);
end process RxProc ;
```

- AffirmIfEqual output in TB uses TB ID ("In TB,")

```
%% Alert ERROR  In TB, Received: 08 /= Expected: 10 at 2150 ns
```

```
%% Log   PASSED In TB, Received: 10 at 2150 ns
```


SynthWorks

OSVVM Makes Self-Checking Tests Easy

- Send transaction on transmitter interface
- Check transaction in receiver

```
TxProc : process

begin
  Send (TxRec, X"10");

  Send (TxRec, X"11");

  . . .

  WaitForBarrier(TestDone);
end process TxProc ;
```

```
RxProc : process

begin
  Check(RxRec, X"10");

  Check(RxRec, X"11");

  . . .

  WaitForBarrier(TestDone);
end process RxProc ;
```

- Checkers in the model use the Model ID (UartRx_1):

```
%% Alert ERROR  In UartRx_1, Received: 08 /= Expected: 10 at 2150 ns
```

```
%% Log   PASSED In UartRx_1, Received: 10 at 2150 ns
```

OSVVM Logs Simplify Test Printing

SynthWorks

- Logs are for conditional test printing

```
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.
  - Logs are disabled by default, except ALWAYS

- Log output for above

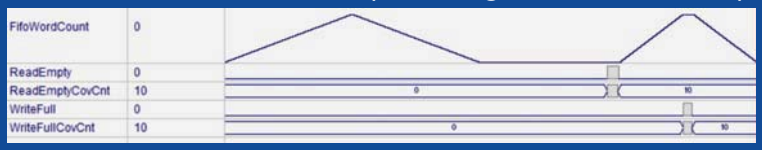
```
%% Log    ALWAYS In TB, Sequence 1 Starting at 2200 ns
```

  - Message with level DEBUG does not print since it is disabled

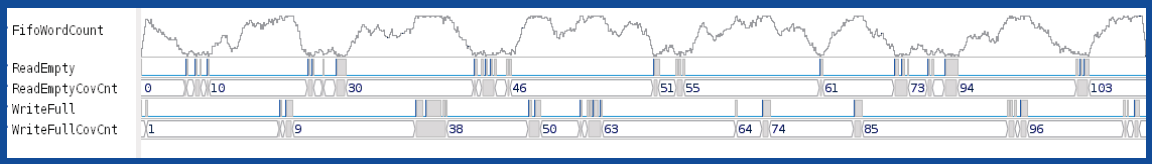
OSVVM Makes Randomization Easy

SynthWorks

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



  - Constrained Random test of a FIFO:



- Key Benefits:
  - Generates realistic stimulus in a timely fashion (to write)
  - Ideal for large variety of similar items
    - Modes, sequences, network packets, processor instructions, …

SynthWorks

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.

SynthWorks

OSVVM Constrained Random Tests

```
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.RandSlv(0, 255, Data'length) ;

    when 1  =>    -- Parity Error   10%

    when 2  =>    -- Stop Error   10%
      Operation := UARTTB_STOP_ERROR ;
      TxD:= RV.RandSlv(1, 255, Data'length) ;

    when  . . .  -- (2, 3, and 4)
  end case ;

  Send(TxRec, TxD, Operation) ;
end loop ;
 . . .
```

Randomize Operation

Nominal 70%

Stop Error 10%

Do Transaction

Constrained Random = Randomization + Code + Transaction Calls

Constrained Random and Checking?

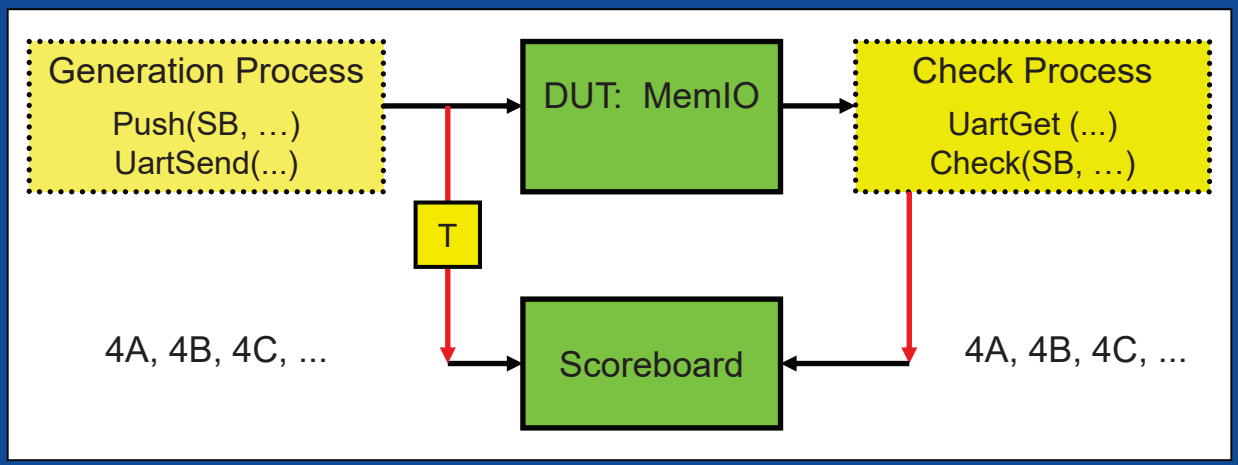
For checking, RxProc could repeat the randomization from TxProc, however, this is tedious and potentially error prone.

```
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 ;
```


OSVVM Generic Scoreboards

- Simplify self-checking when data is minimally transformed



- OSVVM scoreboards use package generics and have:
  - FIFO + Methods - can also be used as a FIFO
  - Handles small data transformations
  - Handles out of order execution
  - Handles dropped values

OSVVM Generic Scoreboards

SynthWorks

```
package ScoreBoardGenericPkg is
  generic(
    type ExpectedType ;
    type ActualType ;
    function Match( . . . ) return boolean ;
    function expected_to_string( . . . ) return string ;
    function actual_to_string  ( . . . ) return string
  ) ;
  type ScoreboardIDType is ... ;
  procedure NewID (…) ;
  procedure Push (…) ;
  procedure Check (…) ;
  procedure Pop (…) ;
  impure function Pop (…) return ExpectedType ;
  impure function Empty (…) return boolean ;
  . . .
  type ScoreBoardPType is protected
    . . .
  end protected ScoreBoardPType ;
end ScoreBoardGenericPkg ;
```

Parameterized with Generics

Scoreboard / FIFO API

OSVVM Scoreboards: Generic Instance

SynthWorks

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;

package ScoreBoardPkg_slv is new osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  std_logic_vector,
    ActualType          =>  std_logic_vector,
    match               =>  MetaMatch,
    expected_to_string  =>  to_hxstring,
    actual_to_string    =>  to_hxstring
  ) ;
```

```
package ScoreBoardPkg_int is new osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  integer,
    ActualType          =>  integer,
    match               =>  "=",
    expected_to_string  =>  to_string,
    actual_to_string    =>  to_string
  ) ;
```

Both in OSVVM Library

SynthWorks

Using OSVVM's Scoreboard is Easy

```
use osvvm.ScoreboardPkg_slv.all ;
signal SB : ScoreboardIDType ;
. . .
SB <= NewID("SB", ModelID) ; -- 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 ;
  . . .
```

```
RxProc : process
  variable RxD  : ByteType;
  variable RV   : RandomPType;
begin
  for I in 1 to 10000 loop

    Get(RxRec, RxD, RxOp);
    Check(SB,(RxD, RxOp));
  end loop ;
  . . .
  WaitForBarrier(TestDone);
```

Key Point:

- In TxProc, Always Push before Send.
- RxProc process does not need to know the expected data

SynthWorks

OSVVM 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 multiple 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 (features, bins, uncorrelated items)

SynthWorks

CoveragePkg

- CoveragePkg simplifies coverage definition, collection, and reporting
  - Implements a data structure and configuration parameters
  - API to all coverage features
  - Implemented as a singleton in CoveragePkg

```
function GenBin ( . . . ) return CovBinType ;

type CoverageIDType is . . . ;
impure function NewID(Name : string; ...)
  return CoverageIDType ;

procedure AddBins (ID : CoverageIDType; CovBin : CovBinType ) ;
procedure AddCross(ID : CoverageIDType;  Bin1, Bin2, ... : CovBinType );

procedure ICover (ID : CoverageIDType; val : integer ) ;
procedure ICover (ID : CoverageIDType; val : integer_vector ) ;

impure function IsCovered (ID : CoverageIDType) return boolean ;

procedure WriteBin        (ID : CoverageIDType) ;
procedure WriteCovHoles   (ID : CoverageIDType) ;
  . . .
```


SynthWorks

OSVVM Functional Coverage is Easy

- For the UART, we track the following items

	Status Register Values				
Condition	Break Error	Stop Error	Parity Error	Done Flag	Integer Value(s)
Normal Transfer	0	0	0	1	1
Parity Error	0	0	1	1	3
Stop Error	0	1	0	1	5
Parity & Stop Error	0	1	1	1	7
Break Error	1	-	-	1	9-15

SynthWorks

OSVVM Functional Coverage is Easy

```
architecture CR_1 of TestCtrl is
  signal RxCov : CoverageIdType ;
```

Coverage Object

```
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, RxOp ) ;
  end loop ;
  . . .
  RxCov.WriteBin ;
```

Initialize Data Structure

Define coverage model

Collect Coverage

Functional Coverage with OSVVM is as concise as language syntax.

SynthWorks

OSVVM Intelligent Coverage

- Intelligent Coverage = Randomize using functional coverage holes

```
architecture CR_1 of TestCtrl is
  signal StimCov : CoverageIdType ;
  . . .
```

Coverage Object

```
TxProc : process
   . . .
begin
  StimCov <=  NewID("StimCov", TB_ID) ;
  wait for 0 ns ;

  AddBins(StimCov, "NORMAL",        7000,  GenBin(1) ) ;
  AddBins(StimCov, "PARITY",        1100,  GenBin(3) ) ;
  AddBins(StimCov, "STOP",          1100,  GenBin(5) ) ;
  AddBins(StimCov, "PARITY + STOP",  600,  GenBin(7)) ;
  AddBins(StimCov, "BREAK",          200,  GenBin(9,15,1) ) ;

  for I in 1 to 10000 loop
    . . .
```

Initialize Data Structure

Add Name and Coverage Goals to coverage model

OSVVM Intelligent Coverage Randomization

SynthWorks

```
TxProc : process
   variable iOperation : integer ;
   . . .

   for I in 0 to 10000 loop
    iOperation := GetRandPoint(StimCov) ;
    case iOperation is

      when 1  =>    . . .  -- Nominal
      when 3  =>    . . .  -- Parity
       . . .
    end case ;

    . . .
    Push(SB, (Data, Operation) ) ;
    ICover(StimCov, iOperation ) ;
    Send(TxRec, Data, Operation) ;
    wait for Idle * UART_BAUD_PERIOD_115200 ;
  end loop ;
  WaitForBarrier(TestDone) ;
```

Randomize

Target values change
Actions do not.

Collect Coverage

Coverage driven randomization with coverage goals used as randomization weights

OSVVM Protocol and Parameter Checkers

SynthWorks

- Protocol Checks (in a Memory Model)

```
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 ;
```

- Parameter Checks (in OSVVM packages)

```
AlertIf (OSVVMID, Max < Min, "RandInt: Max < Min") ;
```

- Alerts can be enabled (default) or disabled

```
SetAlertEnable(ModelID, WARNING, FALSE) ; -- for a model
```

- Stop simulation after receiving 20 Errors

```
SetAlertStopCount(ERROR, 20) ; -- for the entire test
```

Test Finalization

SynthWorks

```
ControlProc : process
begin
  SetAlertLogName("UartRx1") ;

  . . . – Test Initialization

  WaitForBarrier(TestDone, 5 ms) ;

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

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

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

  AffirmIfNotDiff("MemTest1.txt", "Checked/MemTest1.txt");

  EndOfTestReports ;
  std.env.stop(GetAlertCount) ;
  wait ;
end process ControlProc
```

Stop until Test Done or 5 ms has passed

Report Errors

OSVVM Test Watch Dog

SynthWorks

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

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

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

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

```
TestProc2 : process
 . . .
 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 Test Wide Reporting

SynthWorks

- 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
%%    UartRx_1                 Failures: 0  Errors: 0  Warnings: 0  Passed: 0
%%    UartTx_1                 Failures: 0  Errors: 0  Warnings: 0  Passed: 0
```


Including the OSVVM Library is Easy

SynthWorks

- OSVVM Includes numerous packages.
  - To simplify this, OSVVM library provides context declarations

```
-- OSVVM Utility Library, Random, Coverage, …
library osvvm ;
  context osvvm.OsvvmContext ;           -- All OSVVM packages

-- AXI4 Model Library
library osvvm_axi4 ;
  context osvvm_axi4.Axi4Context ;       -- AXI4 Full VC
  context osvvm_axi4.Axi4LiteContext ;   -- AXI4 Lite VC
  context osvvm_axi4.AxiStreamContext ;  -- AXI Stream VC

-- UART Model Library
Library osvvm_uart ;
  context osvvm_uart.UartContext ;       -- UART VC

-- DPRAM Model Library
Library osvvm_dpram ;
  context osvvm_dpram.DpRamContext ;     -- DpRam VC
```

OSVVM Scripting

```
library  osvvm_TbUart

analyze  TestCtrl_e.vhd
analyze  TbUart.vhd
analyze  ../TestCases/TbUart_SendGet1.vhd

simulate TbUart_SendGet1
```

- Benefits
  - Simplistic
  - Reads like a text file,
    - but it is TCL, so the power is there if you need it
  - Works in – GHDL, Aldec, Siemens, Synopsys VCS, Cadence Xcelium
  - Remembers settings like library, code coverage enabled, language rev
  - Paths are relative to the directory from which the scripts run
- Run the Scripts using:

```
build ../OsvvmLibraries/OsvvmLibraries.pro
build ../OsvvmLibraries/RunDemoTestsWithCoverage.pro
```


OSVVM's Reports: Build Summary

OsvvmLibraries_RunAllTestsWithCoverage Build Summary Report

Build	OsvvmLibraries_RunAllTestsWithCoverage
Status	PASSED
PASSED	138
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:06:51
Elapsed Time (seconds)	411.044
Date	2022-04-24T08:13-0700
Simulator	RivieraPRO
Version	RivieraPRO-2021.10.114.8313
OSVVM YAML Version	1.0
Code Coverage	Code Coverage Results

▾ OsvvmLibraries_RunAllTestsWithCoverage Test Suite Summary

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Lite	PASSED	10	0	0	0 / 0	0	28.422
Axi4Full	PASSED	56	0	0	0 / 0	0	159.850
AxiStream	PASSED	63	0	0	0 / 0	0	165.551
Uart	PASSED	8	0	0	0 / 0	0	24.628
DoRam	PASSED	1	0	0	0 / 0	0	3.005

▾ Axi4Lite Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_BasicReadWrite	PASSED	60 / 60	0	0 / 0	-	0	1.450
TbAxi4_ReadWriteAsync1	PASSED	60 / 60	0	0 / 0	-	0	1.186
TbAxi4_ReadWriteAsync2	PASSED	60 / 60	0	0 / 0	-	0	1.144
TbAxi4_ReadWriteAsync3	PASSED	60 / 60	0	0 / 0	-	0	1.077
TbAxi4_RandomReadWrite	PASSED	3000 / 3000	0	0 / 0	-	0	1.298
TbAxi4_RandomReadWriteByte	PASSED	3000 / 3000	0	0 / 0	-	0	1.313
TbAxi4_TimeOut	PASSED	71 / 75	0	0 / 0	-	0	1.171
TbAxi4_WriteOptions	PASSED	72 / 72	0	0 / 0	-	0	2.590
TbAxi4_MemoryReadWrite1	PASSED	40 / 40	0	0 / 0	-	0	1.127
TbAxi4_AxiXResp	PASSED	36 / 36	0	0 / 0	-	0	1.178

▾ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_MemoryReadWriteDemo1	PASSED	334 / 334	0	0 / 0	43.75	0	1.341
TbAxi4_BasicReadWrite	PASSED	60 / 60	0	0 / 0	-	0	1.099
TbAxi4_RandomReadWrite	PASSED	3000 / 3000	0	0 / 0	-	0	1.496

Created by OSVVM Scripts + EndOfTestReports

Build Status

Link to code coverage when run

Test Suite Summary

Test Case Summaries

Has links to Test Case Detailed Reports

- Alert
- Functional Coverage
- Scoreboard

OSVVM's Reports: Alerts

SynthWorks

TbAxi4_RandomReadWrite Alert Report

▼ **TbAxi4_RandomReadWrite Alert Settings**

Setting		Value	Description
FailOnWarning		true	If true, warnings are a test error
FailOnDisabledErrors		true	If true, Disabled Alert Counts are a test error
FailOnRequirementErrors		true	If true, Requirements Errors are a test error
External	Failures	0	Added to Alert Counts in determine total errors
	Errors	0	
	Warnings	0	
Expected	Failures	0	Subtracted from Alert Counts in determine total errors
	Errors	0	
	Warnings	0	

Alert Settings

Alert Report

▼ **TbAxi4_RandomReadWrite Alert Results**

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbAxi4_RandomReadWrite	PASSED	3000	3000	0	0	0	0	0	0	0	0	0
Default	PASSED	1750	1750	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: Data Check	PASSED	0	0	0	0	0	0	0	0	0	0	0
subordinate_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: Data Check	PASSED	250	250	0	0	0	0	0	0	0	0	0
manager_1: No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
manager_1: WriteResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: ReadResponse Scoreboard	PASSED	500	500	0	0	0	0	0	0	0	0	0
manager_1: WriteBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0

OSVVM's Reports: Coverage

SynthWorks

Uart7_Random_part3 Coverage Report

Total Coverage: 100.00

▼ **UART_RX_STIM_COV Coverage Model** **Coverage: 100.0**

▼ UART_RX_STIM_COV Coverage Settings

CovWeight	1
Goal	100.0
WeightMode	at_least
Seeds	824213985 792842968
CountMode	count_first
IllegalMode	illegal_on
Threashold	45.0
ThresholdEnable	0
TotalCovCount	100
TotalCovGoal	100

Coverage Model Settings

▼ UART_RX_STIM_COV Coverage Bins

Name	Type	Mode	Data	Idle	Count	AtLeast	Percent Coverage
NORMAL	Count	1 to 1	0 to 255	0 to 0	63	63	100.0
NORMAL	Count	1 to 1	0 to 255	1 to 15	7	7	100.0
PARITY	Count	3 to 3	0 to 255	2 to 15	11	11	100.0
STOP	Count	5 to 5	1 to 255	2 to 15	11	11	100.0
PARITY_STOP	Count	7 to 7	1 to 255	2 to 15	6	6	100.0
BREAK	Count	9 to 15	11 to 30	2 to 15	2	2	100.0
Total Percent Coverage:		100.0					

Coverage Results

▼ **UART_RX_COV Coverage Model** **Coverage: 100.0**

▶ UART_RX_COV Coverage Settings

▼ UART_RX_COV Coverage Bins

Getting OSVVM & Running Scripts

- Documentation starts at:

```
https://osvvm.github.io/
```

- 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
```

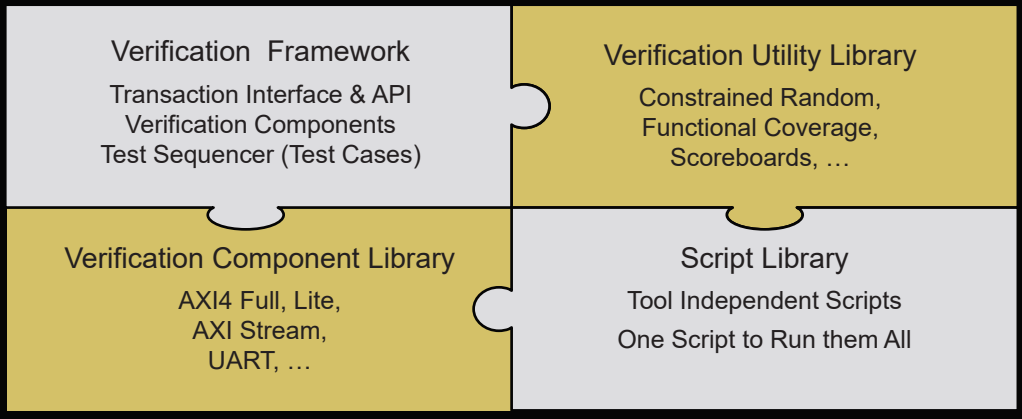
- Build all OSVVM and Run AXI4 Tests

```
build ../OsvvmLibraries/OsvvmLibraries.pro
build ../OsvvmLibraries/AXI4/Axi4/RunAllTests.pro
```

  - There is a RunAllTests for each VC and the OsvvmLibraries
  - There is also a RunDemoTests

OSVVM Benefits

All you need is … OSVVM



- Benefits
  - Upgrade your current VHDL testbench and models incrementally
  - Tests and Verification Components can be written by RTL designers
  - Tests are Readable and Reviewable by All
  - Powerful and concise - rivals other verification languages
  - Supports mixed approaches (directed, algorithmic, file, CR, IT)