OSVVM Testbenches
Level Up your Tests and Testbenches

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

1

OSVVM Testbenches

Copyright © 2020 - 2023 by SynthWorks Design Inc.
Distributing a pdf version of this document in whole for individual usage is permitted.
Printing a paper version of this document in whole for individual usage is permitted.
All other rights reserved.

In particular, without express written permission of SynthWorks Design Inc,
- You may not distribute powerpoint versions of this work,
- You may not alter, transform, or build upon this work,
- You may not use any material from this guide in a group presentation, tutorial, training class, or classroom,
- You must include this page in any printed copy of this document.

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

Copyright © 2020 -2023 by SynthWorks Design Inc.

2

SynthWorks

Level Up Your Tests and Testbench

- Agenda – Piece by Piece Approach
  - Use Transactions
  - Use Context Declarations
  - OSVVM Data Structures are Singletons
  - Use Error and Message Handling
  - Use Randomization
  - Use Functional Coverage
  - Use Scoreboards
  - Generate Reports
  - Use a Structured Testbench Framework
  - Use Verification Components
  - Use Structured Test Cases
  - Memory Modeling Made Easy with OSVVM

Copyright © 2020 -2023 by SynthWorks Design Inc.

3

SynthWorks

Background

- About Jim Lewis
  - 30 years: VHDL Design and Verification
  - 20+ years: Active in IEEE VHDL WGs
  - 15+ years: IEEE VHDL WG chair
  - Chief Architect of OSVVM
  - VHDL Consultant and Trainer for SynthWorks
- SynthWorks provides VHDL Training
  - Comprehensive VHDL Introduction
  - VHDL Coding for Synthesis
  - Advanced VHDL Testbenches and Verification – OSVVM Bootcamp

OSVVM is developed by the same VHDL experts who have helped develop VHDL standards.

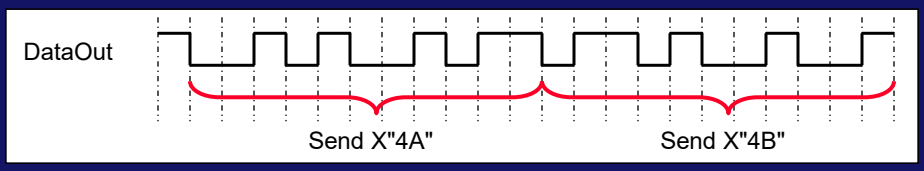
Copyright © 2020 -2023 by SynthWorks Design Inc.

4

SynthWorks

Use Transactions

- Transaction = Interface Operation (Send) or Directive (WaitForClock).



  - Start by identifying the operations that an interface supports

- Transactions Simplify writing tests

```
TxProc : process

begin
  Send (TxRec, X"10") ;

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

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

  Check(RxRec, X"11");
```

Benefit: Accelerates writing tests. Improves readability. Easy to maintain.

Copyright © 2020-2023 by SynthWorks Design Inc.

5

5

SynthWorks

Use OSVVM MIT Transaction API Library

- Address Bus / Memory Mapped Transaction API Library

```
Write(     AddrRec, iAddr, iData) ;
Read (     AddrRec, X"1111_1110", oData) ;
ReadCheck( AddrRec, X"AAAA_AAA0", X"55") ;
ReadPoll(  AddrRec, iAddr, [oData,] Index, BitValue) ;
WriteBurst(AddrRec, iAddr, NumBytes) ;
ReadBurst (AddrRec, iAddr, NumBytes) ;
. . .
```

- Stream Transaction API Library

```
Send (TxRec, iData [, iParam]) ;
Get  (RxRec, oData [, oParam]) ;
Check(RxStreamRec, iData [, iParam]);
SendBurst (StreamRec, NumFifoWords [, iParam]);
GetBurst  (StreamRec, NumFifoWords [, oParam]) ;
CheckBurst(StreamRec, NumFifoWords [, iParam]);
. . .
```

Benefit: Accelerates Test Writing, VC Development, and Documentation

Copyright © 2020-2023 by SynthWorks Design Inc.

6

6

SynthWorks

Use Context Declarations

- A context declaration (VHDL-2008) abstracts package references:

```
context OsvvmContext is
  library OSVVM ;
  use OSVVM.NamePkg.all ;
  use OSVVM.TranscriptPkg.all ;
  use OSVVM.TextUtilPkg.all ;
  use OSVVM.OsvvmGlobalPkg.all ;
  use OSVVM.AlertLogPkg.all ;
  use OSVVM.RandomPkg.all ;
  use OSVVM.CoveragePkg.all ;
  use OSVVM.MemoryPkg.all ;
  use OSVVM.ResolutionPkg.all ;
  use OSVVM.TbUtilPkg.all ;
end context OsvvmContext ;
```

- A context reference, references the set of packages in the context clause

```
Library OSVVM ;
context OSVVM.OsvvmContext ;
entity TestCtrl is
```

Important: Keyword = "context" and not "use"

Copyright © 2020 -2023 by SynthWorks Design Inc.

7

7

SynthWorks

Use Context Declarations

- All OSVVM Libraries provide context declarations to simplify their usage

```
-- 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
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

8

8

SynthWorks

OSVVM Data Structures are Singletons

- Singleton = Dynamic array of things
  - Thing =Scoreboard, FIFO, Functional Coverage, and Alerts
  - Data Structure is created in a package
- ThingIDType is a handle to the data structure – can be a signal or variable

```
signal SB : ScoreboardIDType ;
```

- Singleton objects are constructed by calling NewID

```
SB <= NewID("AXI_SB1") ;
```

  - Can also search for an ID using its name – such as AXI_SB1 above
  - For SB, FIFO, and FC, this is disabled by default.
- Works like a built in language construct?
  - Yes with a VHDL 202X update that most simulators support now

```
Constant SB : ScoreboardIDType := NewID("AXI_SB1") ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

OSVVM Data Structures are Singletons

- ThingIDType is an ordinary type. Use it in records.
  - Can be used in records and entity interfaces

```
type StreamRecType is record
  . . .
  BurstFifo     : ScoreboardIdType ;
  . . .
end record StreamRecType ;
```

Stream Burst FIFO / Scoreboard

  - Can also use it on entity interfaces

Significantly easier than using shared variables and protected types.

Copyright © 2020 -2023 by SynthWorks Design Inc.

Use OSVVM Error and Message Handling

SynthWorks

- Setting Up
  - Name Your Test
  - Create Test Transcript
  - Use AlertLogIDs
- Signal Errors with Alerts
- Conditional Printing with Logs
- Self-Checking with Affirmations
- Simple Printing

Copyright © 2020 -2023 by SynthWorks Design Inc.

Name Test + Create Test Transcript

SynthWorks

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

  TranscriptOpen ; -- ("UartRx1.log")
  SetTranscriptMirror(TRUE) ;

  . . .
  TranscriptClose ;
```

Set Test Name

Open Transcript File + Write to Console

Close Transcript File

- TestName is required for OSVVM reporting
  - Used as a base name for file outputs
  - Expected to match TestName in scripts
- Transcript file saves ALL OSVVM test output
  - OSVVM Scripting saves it in directory <TestSuiteName>/<Name>
  - Without parameters, Name = <TestName>.log
- Close transcript when done

Copyright © 2020 -2023 by SynthWorks Design Inc.

Use AlertLogIDs

SynthWorks

```
signal TbID, RxID : AlertLogIDType ;
. . .

ControlProc : process
begin
  . . .
  TbID <= NewID("TB") ;
  RxID <= NewID("UartRx_1") ;
```

Construct AlertLogIDs

- AlertLogIDs associate test output with the ID name
  - Identifies source of errors
  - If know the name, can search for the ID (also with NewID)
  - IDs are optional

Copyright © 2020 -2023 by SynthWorks Design Inc.

13

13

Use Alerts to Signal Errors

SynthWorks

- Alerts signal errors and keep counts in the AlertLog data structure

```
AlertIf(SramAlertID, (iCE and iWE and iOE) = '1',
      "nCE, nWE, and nOE are all active") ;
```

  - Alert Levels: FAILURE, ERROR (default), WARNING
- Output: Here SramAlertID = "SRAM_1"

```
%% Alert ERROR  In SRAM_1, Received: 08 /= Expected: 10 at 2150 ns
```

- Controls: StopCount, PrintCount, Enable/Disable

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

- Alerts also have conditional forms, such as:

```
AlertIf   (Max < Min, "RandInt: Max < Min") ;
AlertIfNot(ReadValid, "Read Failed", FAILURE) ;
AlertIfDiff("test1.log", "../verified/test1.log");
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

14

14

Use Logs for Conditional 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.

- Controls: Enable/Disable

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

- 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

Copyright © 2020 -2023 by SynthWorks Design Inc.

15

15

Use Affirmations for Self-Checking

SynthWorks

```
AffirmIf(TbID, Rcv = Expect, "Received: " & to_string(Rcv),
                  " /= Expected: " & to_string(Expect)) ;
```

- AffirmIf prints with Log PASSED or Alert ERROR

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

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

- Additional Forms:

```
AffirmIfEqual(Rcv, Expect) ; -- Shorthand for above
AffirmIfNotDiff("test1.log", "../verified/test1.log");
```

- At end of test, check for self-checking with GetAffirmCount

```
AlertIf( GetAffirmCount < 1, "Test is not Self-Checking");
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

16

16

SynthWorks

Simple Printing

- Simple string based print to transcript

```
Print("A String") ;       -- String print, newline added
Print("") ;               -- Print a blank line
```

- Print + to_string (VHDL-2008) = general purpose printing

```
type   StateType is (S1, S2, S3, S4) ;
signal State : StateType ;
signal Count : integer ;
signal Data  : std_logic_vector(15 downto 0) ;
. . .
Print("State = "   & to_string(State) &
      ", Count = " & to_string(Count) &
      ", Data = "  & to_hstring(Data) &
      " at time "  & to_string(now, 1 ns) ) ; -- see note
```

```
State = S2, Count = 15, Data = A5A5 at time 2550 ns
```

- TextIO style printing to transcript

```
writeline( WriteBuf ) ;  -- Using textio
```

Copyright © 2020-2023 by SynthWorks Design Inc.

17

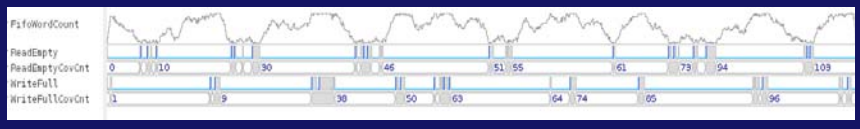
17

SynthWorks

Use Randomization

- 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, …

Copyright © 2020-2023 by SynthWorks Design Inc.

18

18

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

Copyright © 2020 -2023 by SynthWorks Design Inc.

19

19

SynthWorks

OSVVM Constrained Random Tests

= 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.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  . . .  -- (3 and 4)
  end case ;

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

Randomize Operation

Nominal 70%

Stop Error 10%

Do Transaction

Copyright © 2020 -2023 by SynthWorks Design Inc.

20

20

SynthWorks

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 ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

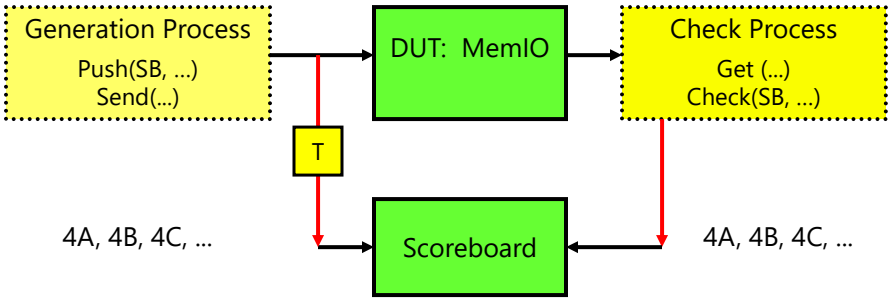
21

21

SynthWorks

Use 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

Copyright © 2020 -2023 by SynthWorks Design Inc.

22

22

OSVVM Generic Scoreboards

SynthWorks

- ScoreboardGenericPkg is a singleton for Scoreboards and FIFOs

```
package ScoreBoardGenericPkg is
  (
    type ExpectedType ;
    type ActualType ;
    function Match( . . . ) return boolean ;
    function expected_to_string( . . . ) return string ;
    function actual_to_string  ( . . . ) return string
  ) ;
  type ScoreboardIDType is ... ;
  type ScoreboardIdArrayType is ... ;
  type ScoreboardIdMatrixType is ... ;
  procedure NewID (…) ;
  procedure Push (…) ;
  procedure Check (…) ;
  procedure Pop (…) ;
  impure function Pop (…) return ExpectedType ;
  impure function Empty (…) return boolean ;
  impure function Find (…) return integer ;
  procedure Flush(…) ;
  . . .
```

Parameterized with Generics

Scoreboard / FIFO Singleton API

Note the API is not inside a protected type

Copyright © 2020 -2023 by SynthWorks Design Inc.

23

23

Scoreboards: Generic Instance

SynthWorks

```
use work.UartTbPkg.all ;

package ScoreBoardPkg_Uart is new osvvm.ScoreBoardGenericPkg
  generic map (
    ExpectedType        =>  UartCpuType,
    ActualType          =>  UartStimType,
    match               =>  match,
    expected_to_string  =>  to_string,
    actual_to_string    =>  to_string
  ) ;
```

- To create a new scoreboard,
  - In a package define (UartTbPkg):
    - Expected and Actual types to hold test and result values
    - Function match to compare Expected to Actual values
    - Functions to_string for the expected and actual types
  - In a separate file (ScoreboardPkg_Uart.vhd):
    - Create a package instance that maps the types and functions

Copyright © 2020 -2023 by SynthWorks Design Inc.

24

24

SynthWorks

Scoreboards: Instances in OSVVM Library

```
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

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Scoreboards

- Match function allows for corrections of small differences

```
function Match (
  constant  Actual      : in    UartStimType ;
  constant  Expected    : in    UartCpuType
) return boolean is
begin
  if Expected.ErrorMode(UARTTB_BREAK_INDEX) = '1' then
    return Actual.Status(RX_BREAK_ERROR) = '1' ;
  else
    return Actual = Expected ;
  end if ;
end function Match ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

Using OSVVM's Scoreboard is Easy

SynthWorks

```
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 ;
  . . .
```

```
RxProc : process
  . . .
begin
  for I in 1 to 10000 loop

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

Benefit:

- Same RxProc is used for either directed or constrained random tests

Copyright © 2020-2023 by SynthWorks Design Inc.

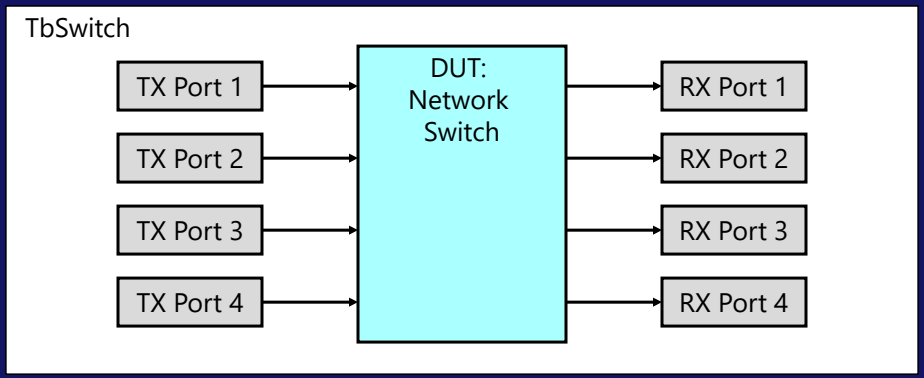
27

27

Arrays of Scoreboards

SynthWorks

- How do scoreboards work with a network switch?
  - TX Port 1 can send to the RX Port 1, 2, 3, or 4
  - Similarly, TX Ports 2, 3, or 4 can transmit to RX Port 1, 2, 3, or 4
  - We need a 2D array of scoreboards: RxPort x TxPort



Copyright © 2020-2023 by SynthWorks Design Inc.

28

28

SynthWorks

Indexed Scoreboards

- ScoreboardIDMatrixType: A 2D array of ScoreboardIdType

```
type ScoreboardIdMatrixType is array
   (integer range <>, integer range <>) of ScoreboardIdType ;
```

- A 2D array of scoreboards

```
signal SB : ScoreboardIdMatrixType(1 to 4, 1 to 4) ;
```

- Initialize the scoreboards in a single step

```
SB <= NewID("SwitchSB", 4, 4, ParentID) ;
```

- Initialize the scoreboards individually (equivalent to above)

```
SB(1,1) <= NewID("SwitchSB(1, 1)", ParentID) ;
SB(1,2) <= NewID("SwitchSB(1, 2)", ParentID) ;
SB(1,3) <= NewID("SwitchSB(1, 3)", ParentID) ;
. . .
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Indexed Scoreboards

- In the test, each TX Port pushes to appropriate scoreboard

```
TxPort1Proc : process
  . . .
  while TestActive loop
    CreateStim (RxPort, Data, Len, ...) ;
    PushBurstVector(SB(RxPort, TX_PORT_1), Data(1 to Len), 8) ;
    DoTransaction(Tx1Rec, RxPort, TX_PORT_1, Data, Len);
  end loop ;
```

- In the test, each RX Port Checks the transactions it receives

```
RxPort1Proc : process
  . . .
  while TestActive loop
    GetTransaction(Rx1Rec, RX_PORT_1, TxPort, Data, Len);
    CheckBurstVector(SB(RX_PORT_1, TxPort), Data(1 to Len), 8) ;
  end loop ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Use 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)

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

CoveragePkg

- CoveragePkg simplifies coverage definition, collection, and reporting
  - Internally it has a data structure and configuration parameters
  - Implemented as a singleton in CoveragePkg
  - The singleton API defines the coverage capabilities

```
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) ;
  . . .
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

OSVVM Functional Coverage is Easy

SynthWorks

- For the UART, we track the following items

Condition	Break Error	Stop Error	Parity Error	Done Flag	Integer Value(s)
	Status Register Values				
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

Copyright © 2020 -2023 by SynthWorks Design Inc.

OSVVM Functional Coverage is Easy

SynthWorks

```
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, to_integer(RxOp));
  end loop ;
  . . .
```

Construct the Data Structure

Define coverage model

Collect Coverage

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

SynthWorks

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 ;
```

Coverage Object

Constructor

Coverage Goals = Randomization Weights

Randomize

Similar actions to constrained random

Do transaction & Collect coverage

Intelligent Coverage goes beyond what SV does

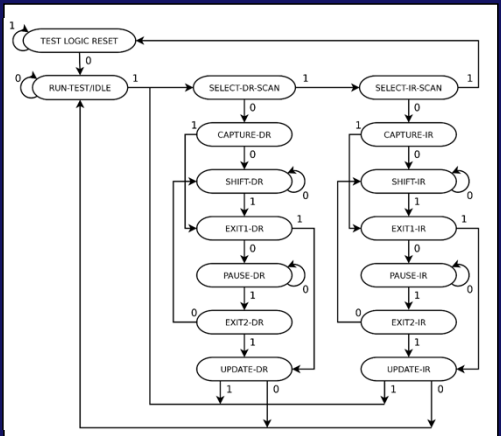
35

35

SynthWorks

Use Transition Coverage for Statemachines

- Transition Coverage is just Cross Coverage + History



CurState	=>	NextState
RESET	=>	RESET
RESET	=>	IDLE
IDLE	=>	IDLE
IDLE	=>	SELECT_DR
SELECT_DR	=>	CAPTURE_DR
SELECT_DR	=>	IR_SEL
CAPTURE_DR	=>	SHIFT_DR
CAPTURE_DR	=>	EXIT1_DR
EXIT1_DR	=>	PAUSE_DR
EXIT1_DR	=>	UPDATE_DR

. . .

All other transitions are illegal

- Original OSVVM example by Alex Grove and published on Aldec Website in 2016

Copyright © 2020 -2023 by SynthWorks Design Inc.

36

36

Transition Coverage: Accessing State Information

SynthWorks

- Need to do access the State to collect coverage
- However, signal and type declarations are in the RTL

```
type TapStateType is
  (RESET, IDLE,
   SELECT_DR, CAPTURE_DR, SHIFT_DR, EXIT1_DR,
   PAUSE_DR,  EXIT2_DR, UPDATE_DR,
   SELECT_IR, CAPTURE_IR, SHIFT_IR, EXIT1_IR,
   PAUSE_IR,  EXIT2_IR, UPDATE_IR ) ;

signal State, NextState : TapStateType ;
```

- Step1: Instrument RTL to access State (or put TapStateType in a package)

```
signal intState : integer ;
. . .
-- synthesis translate_off
intState <= TapStateType'pos(State) ;
-- synthesis translate_on
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

37

37

Transition Coverage: Defining Constants

SynthWorks

- Step 2: Define coverage constants

```
package TapTbPkg is
  -- type TapStateType is (. . . ) ;
  -- synthesis translate_off
  constant COV_RESET      : CovBinType := GenBin(0) ;
  constant COV_IDLE       : CovBinType := GenBin(1) ;
  constant COV_SELECT_DR  : CovBinType := GenBin(2) ;
  constant COV_CAPTURE_DR : CovBinType := GenBin(3) ;
  constant COV_SHIFT_DR   : CovBinType := GenBin(4) ;
  constant COV_EXIT1_DR   : CovBinType := GenBin(5) ;
  constant COV_PAUSE_DR   : CovBinType := GenBin(6) ;
  constant COV_EXIT2_DR   : CovBinType := GenBin(7) ;
  constant COV_UPDATE_DR  : CovBinType := GenBin(8) ;
  . . .
  -- synthesis translate_on
```

Note:
The numbers in GenBin must correlate with the position number in TapStateType

Copyright © 2020 -2023 by SynthWorks Design Inc.

38

38

Transition Coverage: Defining Coverage

SynthWorks

- Step 3: Define the State Transitions in Coverage Model

```
CaptureCoverage : process
 variable CovSt : CoverageIdType
 variable PrevState : integer ;
begin
  CovSt := NewID("CovST") ;
  SetItemBinNames(CovST,                          "PrevState",    "CurState") ;
  AddCross(CovSt, "RESET      -> RESET",      COV_RESET,      COV_RESET) ;
  AddCross(CovSt, "RESET      -> IDLE",       COV_RESET,      COV_IDLE) ;
  AddCross(CovSt, "IDLE       -> IDLE",       COV_IDLE,       COV_IDLE ) ;
  AddCross(CovSt, "IDLE       -> SELECT_DR",  COV_IDLE,       COV_SELECT_DR ) ;
  AddCross(CovSt, "SELECT_DR  -> SELECT_IR",  COV_SELECT_DR,  COV_SELECT_IR) ;
  AddCross(CovSt, "SELECT_DR  -> CAPTURE_DR", COV_SELECT_DR,  COV_CAPTURE_DR) ;
  AddCross(CovSt, "CAPTURE_DR -> SHIFT_DR",   COV_CAPTURE_DR, COV_SHIFT_DR) ;
  AddCross(CovSt, "CAPTURE_DR -> EXIT1_DR",   COV_CAPTURE_DR, COV_EXIT1_DR) ;
  AddCross(CovSt, "SHIFT_DR   -> SHIFT_DR",   COV_SHIFT_DR,   COV_SHIFT_DR) ;
  AddCross(CovSt, "SHIFT_DR   -> EXIT1_DR",   COV_SHIFT_DR,   COV_EXIT1_DR) ;
  . . .
  AddCross(CovST, "Illegal ",  ALL_ILLEGAL,  ALL_ILLEGAL ) ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

Transition Coverage: Capturing Coverage

SynthWorks

- Step 4: Capture Coverage (Previous and Current)

```
CaptureCoverage : process
  alias    CurState is <<signal .tb.VC1_1.intState : integer>>;
  variable PrevState : integer ;
begin
  wait until Clk = '1' ;
  PrevState := CurState ;
  loop
    wait until Clk = '1' ;
    ICover(CovST, (PrevState, CurState) ) ;
    PrevState := CurState ;
  end loop ;
end process CaptureCoverage ;
```

- TCover (Transition Cover) – Remembers previous value (BETA feature)

```
  loop
    wait until Clk = '1' ;
    TCover(CovST, CurState) ;
  end loop ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

Transition Coverage: Coverage Results

SynthWorks

Name	Type	PrevState	CurState	Count	AtLeast	Percent Coverage
TEST_LOGIC_RESET to TEST_LOGIC_RESET	COUNT	0	0	7	1	700.0
TEST_LOGIC_RESET to RUN_TEST_IDLE	COUNT	0	1	8	1	800.0
RUN_TEST_IDLE to RUN_TEST_IDLE	COUNT	1	1	20	1	2000.0
RUN_TEST_IDLE to SELECT_DR_SCAN	COUNT	1	2	20	1	2000.0
SELECT_DR_SCAN to SELECT_IR_SCAN	COUNT	2	9	15	1	1500.0
SELECT_DR_SCAN to CAPTURE_DR	COUNT	2	3	16	1	1600.0
CAPTURE_DR to SHIFT_DR	COUNT	3	4	8	1	800.0
CAPTURE_DR to EXIT1_DR	COUNT	3	5	8	1	800.0
SHIFT_DR to SHIFT_DR	COUNT	4	4	23	1	2300.0
SHIFT_DR to EXIT1_DR	COUNT	4	5	13	1	1300.0
EXIT1_DR to PAUSE_DR	COUNT	5	6	8	1	800.0
EXIT1_DR to UPDATE_DR	COUNT	5	8	13	1	1300.0
PAUSE_DR to PAUSE_DR	COUNT	6	6	11	1	1100.0
PAUSE_DR to EXIT2_DR	COUNT	6	7	8	1	800.0
EXIT2_DR to SHIFT_DR	COUNT	7	4	5	1	500.0
EXIT2_DR to UPDATE_DR	COUNT	7	8	3	1	300.0
UPDATE_DR to SELECT_DR_SCAN	COUNT	8	2	8	1	800.0
UPDATE_DR to RUN_TEST_IDLE	COUNT	8	1	8	1	800.0
SELECT_IR_SCAN to TEST_LOGIC_RESET	COUNT	9	0	7	1	700.0
SELECT_IR_SCAN to CAPTURE_IR	COUNT	9	10	8	1	800.0
CAPTURE_IR to SHIFT_IR	COUNT	10	11	1	1	100.0
CAPTURE_IR to EXIT1_IR	COUNT	10	12	7	1	700.0
SHIFT_IR to SHIFT_IR	COUNT	11	11	1	1	100.0
SHIFT_IR to EXIT1_IR	COUNT	11	12	1	1	100.0
EXIT1_IR to UPDATE_IR	COUNT	12	15	4	1	400.0
EXIT1_IR to PAUSE_IR	COUNT	12	13	4	1	400.0
PAUSE_IR to PAUSE_IR	COUNT	13	13	2	1	200.0
PAUSE_IR to EXIT2_IR	COUNT	13	14	4	1	400.0
EXIT2_IR to SHIFT_IR	COUNT	14	11	1	1	100.0
EXIT2_IR to UPDATE_IR	COUNT	14	15	3	1	300.0
UPDATE_IR to SELECT_DR_SCAN	COUNT	15	2	3	1	300.0
UPDATE_IR to RUN_TEST_IDLE	COUNT	15	1	4	1	400.0
JTAG TAP FSM: ILLEGAL Transition	ILLEGAL	ALL	ALL	0	0	100.0
Total Percent Coverage: 100.0						

Copyright © 2020 -2023 by SynthWorks Design Inc.

41

41

Generate Reports

SynthWorks

- EndOfTestReports + OSVVM scripts generate reports

```
EndOfTestReports ;
```

- EndOfTestReports produces PASSED / FAILED message for each test

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

- 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
```

- When a build finishes, reports are created for:
  - Build Summary (HTML and JUnit XML)
  - Test Case Report (HTML)
  - Log File (HTML and text)

Copyright © 2020 -2023 by SynthWorks Design Inc.

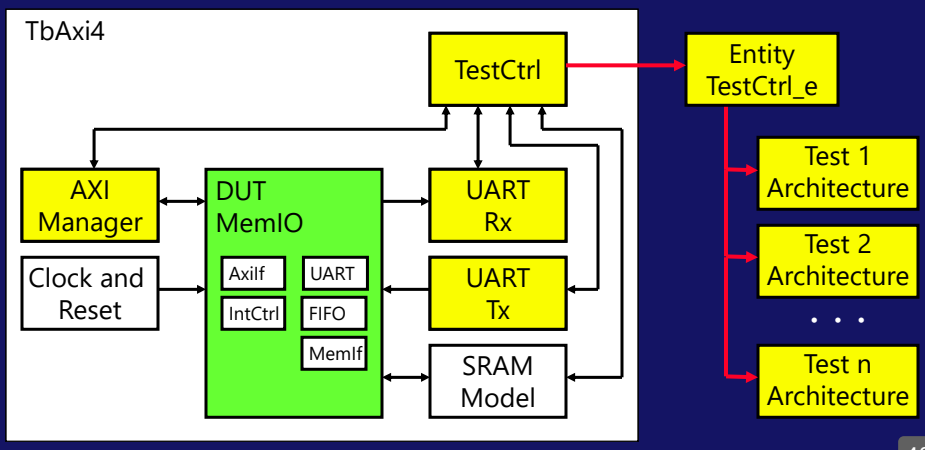
42

42

SynthWorks

Use a Structured Testbench 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



Copyright © 2020-2023 by SynthWorks Design Inc.

43

43

SynthWorks

Framework Implementation

Structural Code
Plugs together just like RTL

```
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 ;
```

DUT

Verification Components

Test Sequencer

44

44

SynthWorks

Creating Clock and Reset

- OSVVM package TbUtilPkg has procedures to create clock and reset

```
-- Create Clock
osvvm.TbUtilPkg.CreateClock(
  Clk        => Clk,
  Period     => tperiod_Clk,
  DutyCycle  => 0.5                  -- default, not required
) ;

-- Create Reset
osvvm.TbUtilPkg.CreateReset(
  Reset        => nReset,
  ResetActive  => '0',
  Clk          => Clk,
  Period       => 7 * tperiod_Clk,   -- length of reset
  tpd          => tpd
) ;
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

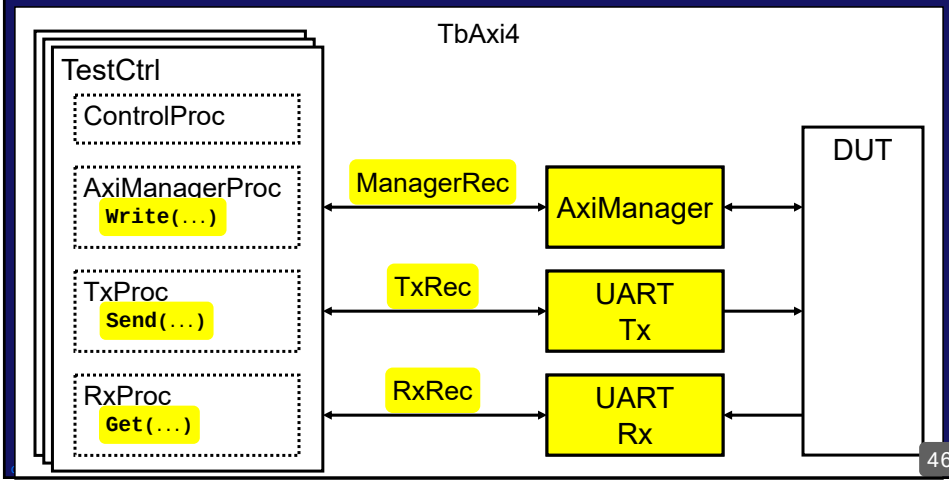
45

45

SynthWorks

Use Verification Components

- 3 Steps to VC Development
  - Step 1: Transaction Interface (ManagerRec, …) – provided by MIT library
  - Step 2: Transaction API (Write, Send, …) – provided by MIT library
  - Step 3: Verification components



46

46

Use Verification Components

SynthWorks

```
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 Manager Functional Interface
  AxiBus       : inout Axi4RecType ;

  -- Testbench Transaction Interface
  TransRec     : inout AddressBusRecType
) ;
end entity Axi4Manager ;
```

AXI4 Interface

Transaction Interface
Defined by OSVVM MIT

OSVVM VC Library includes:
Axi4, Axi4Lite, AxiStream, UART, Ethernet xMII, DpRam

Copyright © 2020-2023 by SynthWorks Design Inc.

Use Verification Components

SynthWorks

Architecture Model of AxiManager is

Initialize Alerts	• Declare and allocate AlertIDs
Preprocess Inputs	• Remove H and L from inputs
Transaction Handler / Transaction Dispatcher	• Decode transactions • Handlers also execute transactions
{ Interface Handler }	• Execute* transactions
Protocol Checker	• Detect interface signaling errors
Timing Checker	• Check timing
Log Results	• Print interface values

Copyright © 2020-2023 by SynthWorks Design Inc.

Use Structured Test Cases: Test Sequencer

SynthWorks

```
entity TestCtrl is
port (
  TxRec         : InOut StreamRecType ;
  RxRec         : InOut StreamRecType ;

  ManagerRec    : InOut AddressBusRecType;

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

Ports = Transaction Interfaces

Copyright © 2020-2023 by SynthWorks Design Inc.

49

49

```
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

50

50

Test Initialization in ControlProc

SynthWorks

```
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
```

Set Test Name

Open Transcript File + Write to Console

Construct AlertLog and Scoreboard data structures

Enable Logs Message Filtering

Stop until Test Done or 5 ms has passed = WatchDog timer

Copyright © 2020 -2023 by SynthWorks Design Inc.

51

51

Test Finalization

SynthWorks

```
ControlProc : process
begin
  SetTestName("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("UartRx1.log", "Checked/UartRx1.log") ;

  EndOfTestReports ;
  std.env.stop ;
  wait ;
end process ControlProc
```

Stop until Test Done or 5 ms has passed

Create Reports

Copyright © 2020 -2023 by SynthWorks Design Inc.

52

52

SynthWorks

Get Synchronized

- Barrier Synchronization

```
WaitForBarrier ( TestDone ) ;
WaitForBarrier ( TestDone, 5 ms ) ;
```

Types: std_logic, integer_barrier

- Toggle / Increment Synchronization

```
Toggle ( DoRead ) ;
Toggle ( DoRead, 5 * tperiod_Clk ) ;
WaitForToggle ( DoRead ) ;
```

Types: bit, std_logic

```
Increment ( TransactionCount_int ) ;
WaitForToggle (TransactionCount_int ) ;
```

Types: integer

- Wait for Clock

```
WaitForClock ( Clk, 5 * tperiod_Clk ) ;
WaitForClock ( Clk, 5 ) ;
WaitForClock ( Clk, Enable ) ;
```

Types: std_logic

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Get Synchronized

- Purpose: Coordinate test finalization

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

```
ControlProc : process
begin
  SetTestName("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

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Memory Modeling with OSVVM

- Why is my Memory Model Slow?

```
type MemArrayType is array(0 to 65535) of
   std_logic_vector(7 downto 0);

signal MemArray     : MemArrayType ;
```

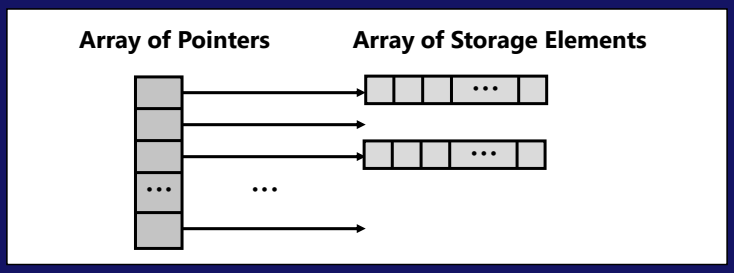
- Allocation
  - Not all of memory is used in most simulations
  - However, a signal (or variable) declaration allocates the whole memory.
- Objects
  - Signals are convenient for implementation
  - Signals use more workstation memory than variables
- Types
  - Std_logic has 9 values (U, X, 0, 1, Z, W, L, H, -) - 4 bits per value
  - Integer uses one memory location

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Memory Modeling with OSVVM

- MemoryPkg is a singleton data structure for memory models
- Each singleton object implements a sparse memory data structure



  - On a write a block of storage is allocated (like cache)
  - Internally it uses variables
  - Internally it uses integer for storage

Copyright © 2020 -2023 by SynthWorks Design Inc.

MemoryPkg

SynthWorks

```
architecture Model of Sram is
  signal MemoryID : MemoryIDType ;
  . . .
begin
  MemoryID <= NewID(
      Name       =>  Axi4Memory'instance_name,
      Addrwidth  =>  Addr'length,
      DataWidth  =>  Data'length,
      Search     =>  NAME ) ;      -- Private
  . . .
  MemWrite(MemoryID, Address, Data) ;
  . . .
  MemRead(MemoryID, Address, Data) ;
   . . .
  FileReadH(MemoryID, VerilogMemory.File) ;
  . . .
  FileWriteH(MemoryID, VerilogMemory.File) ;
  . . .
```

Declare ID

Initialize Structure

Memory Write

Memory Read

Read File into Memory

Write Memory into File

Easier to use than an array object

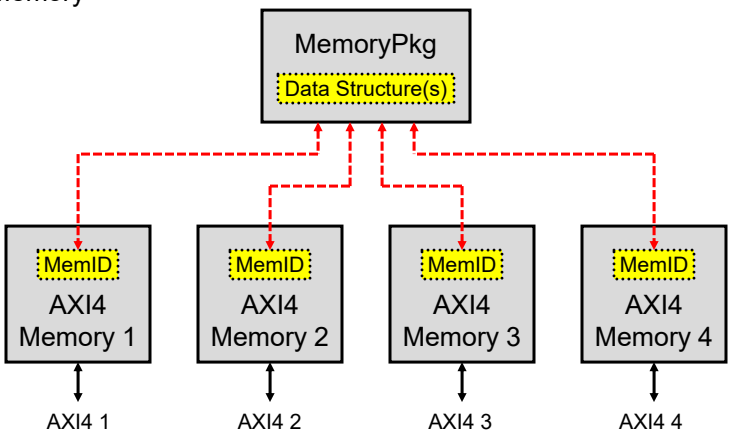
57

57

Implementing ZynqMemory

SynthWorks

ZynqMemory



- Each AXI4 Memory subordinate can either
  - share one memory or
  - have a separate memory

Copyright © 2020 -2023 by SynthWorks Design Inc.

58

58

SynthWorks

Axi4Memory

- Sharing Memory AXI subordinate – it is about the MEMORY_NAME

```
entity Axi4Memory is
generic (
  MODEL_ID_NAME   : string := "" ;
  MEMORY_NAME     : string := "" ;
```

- Set the MEMORY_NAME generic the same to share the same memory

```
architecture structural of ZynqMemory is
begin
  Axi4Memory_1 : Axi4Memory
    generic map ( MEMORY_NAME => "SHARED" )
    port map    ( . . . ) ;
  Axi4Memory_2 : Axi4Memory
    generic map ( MEMORY_NAME => "SHARED" )
    port map    ( . . . ) ;
  Axi4Memory_3 : Axi4Memory
    generic map ( MEMORY_NAME => "SHARED" )
    port map    ( . . . ) ;
  Axi4Memory_4 : . . .
```

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

Axi4Memory

- Creating the Memory Name

```
constant LOCAL_MEMORY_NAME : string :=
    IfElse( MEMORY_NAME /= "", MEMORY_NAME,
            to_lower(Axi4Memory'PATH_NAME) & ":memory") ;
```

- With "Search => NAME", NewID returns the same ID if names match.

```
constant MemoryID : MemoryIDType := NewID(
      Name       => LOCAL_MEMORY_NAME,
      AddrWidth  => AXI_ADDR_WIDTH,  -- Byte Address
      DataWidth  => 8,               -- Data stored as bytes
      Search     => NAME
    ) ;
```

- Same mechanism is available for Coverage and Scoreboards

Copyright © 2020 -2023 by SynthWorks Design Inc.

SynthWorks

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
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2187
  - OSVVM's Test Reports and Simulator Independent Scripting
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2188
  - Advances in OSVVM's Verification Data Structures
    - https://www.aldec.com/en/support/resources/multimedia/webinars/2190
- Jump start your VHDL verification effort with training
  - Advanced VHDL Testbenches and Verification – OSVVM Boot Camp
  - https://synthworks.com/vhdl_testbench_verification.htm

Copyright © 2020 -2023 by SynthWorks Design Inc.

61

SynthWorks

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

Copyright © 2020 -2023 by SynthWorks Design Inc.

62

SynthWorks

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.

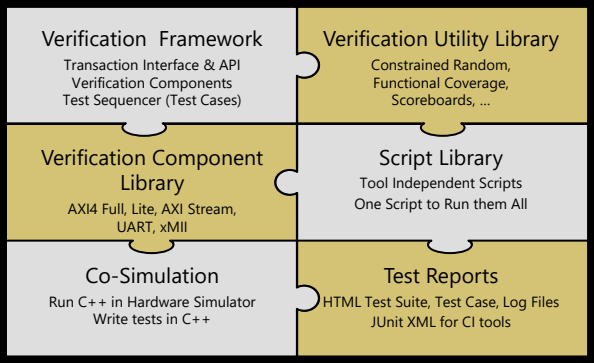
Advanced VHDL Testbenches and Verification - OSVVM Boot Camp - 5 days
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction based modeling, self-checking, scoreboards, memory modeling, functional coverage, directed, algorithmic, 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.

Copyright © 2020 -2023 by SynthWorks Design Inc.

63

All you need is ... OSVVM



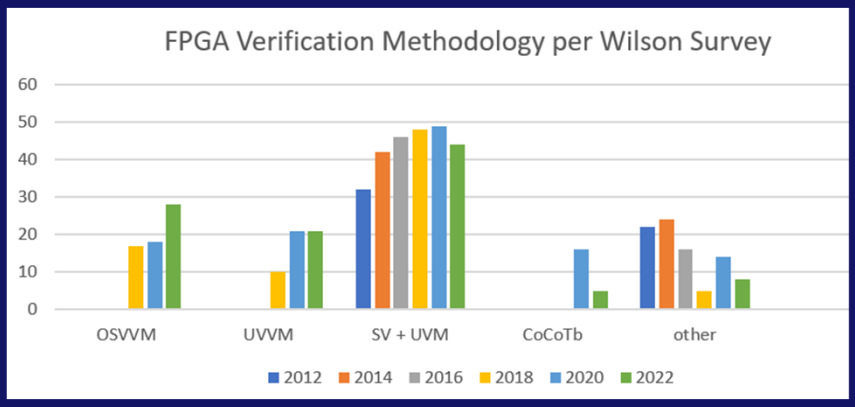
- Benefits
  - Powerful and Concise – rivals other verification languages
  - Unmatched reuse through the entire verification process
  - Unmatched report capability with HTML for humans and JUnit XML for CI
  - Tests are Readable and Reviewable by All
  - Adopt incrementally as needed
  - Tests and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

64

SynthWorks

Why OSVVM?

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



- © Siemens 2022 https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/

Copyright © 2020 -2023 by SynthWorks Design Inc.

65

65