DelayCoveragePkg

User Guide for Release 2025.04

By

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

http://www.SynthWorks.com

Table of Contents

1 Overview

DelayCoveragePkg implements a pattern for randomizing cycle-based delays such as AXI's Valid and Ready signals.

Cycle based delays may be a delay between each word sent, or they may be burst like and transfer a burst of values followed by a delay between the bursts (a burst delay). During a burst of values, there may be delay between each value transmitted (a beat delay).

To address this, DelayCoveragePkg implements a singleton that has three coverage models. A separate coverage model is used to randomize the burst length, burst delay, and beat delay. This supports complex as well as simple use models. For example, if we want a beat delay of 0, then we tell it to select a value between 0 and 0.

For the first release, the documentation is terse. Suggestions and clarifications are welcome.

2 Package Interface

The interface to the package is as follows.

```
package DelayCoveragePkg is

  type DelayCoverageIDType is record
      ID             : integer_max ;
      BurstLengthCov : CoverageIDType ;
      BurstDelayCov  : CoverageIDType ;
      BeatDelayCov   : CoverageIDType ;
  end record DelayCoverageIDType ;

  type DelayCoverageIDArrayType is array (integer range <>) of DelayCoverageIDType ;

  ------------------------------------------------------------
  impure function NewID (
    Name                : String ;
    ParentID            : AlertLogIDType          := OSVVM_COVERAGE_ALERTLOG_ID ;
    ReportMode          : AlertLogReportModeType  := DISABLED ;
    Search              : NameSearchType          := PRIVATE_NAME ;
    PrintParent         : AlertLogPrintParentType := PRINT_NAME_AND_PARENT
  ) return DelayCoverageIDType ;

  ------------------------------------------------------------
  impure function NewDelayCoverage (
    ID                  : Integer ;
    Name                : String ;
    ParentID            : AlertLogIDType          := OSVVM_COVERAGE_ALERTLOG_ID ;
    ReportMode          : AlertLogReportModeType  := DISABLED ;
```

```
    Search              : NameSearchType          := PRIVATE_NAME ;
    PrintParent         : AlertLogPrintParentType := PRINT_NAME_AND_PARENT
  ) return DelayCoverageIDType ;

  ------------------------------------------------------------
  impure function IsInitialized (ID : DelayCoverageIDType) return boolean ;

  impure function GetDelayCoverage(ID : integer) return DelayCoverageIDType ;
  procedure SetDelayCoverage ( ID : DelayCoverageIDType ) ;
  procedure SetBurstLength ( ID : DelayCoverageIDType ; BurstLength : integer ) ;
  impure function GetBurstLength ( ID : DelayCoverageIDType ) return integer ;

  ------------------------------------------------------------
  impure function GetRandDelay ( ID : DelayCoverageIDType ) return integer ;
  impure function GetRandDelay ( ID : DelayCoverageIDType ) return integer_vector ;

  ------------------------------------------------------------
  procedure DeallocateBins ( ID : DelayCoverageIDType ) ;

end package DelayCoveragePkg ;
```

The pattern used here is slightly different from other OSVVM singleton's in that the DelayCoverageIDType has the ID as well as the ID's for the coverage models.

3 Usage in a Verification Component

DelayCoveragePkg is being used by the OSVVM AxiStreamTransmitter and AxiStreamReceiver.

Step 1, declare an object of DelayCoverageIDType. The AxiStream VC need an object with global scope, so a signal is used.

```
signal BurstCov : DelayCoverageIDType ;
```

Step 2, construct the data structure. The wait for 0 is necessary since ModelID is constructed at time 0 and is not available immediately. This is done in TransactionDispatcher process since there are API commands that also update the BurstCov signal.

```
TransactionDispatcher : process
  . . .
begin
  wait for 0 ns ;
  BurstCov <= NewID("DelayCov", ModelID, ReportMode => DISABLED,
                    Search => NAME_AND_PARENT) ;
```

Step 3, implement the API in TransactionDispatcher.

```
case TransRec.Operation is
  . . .

  when SET_USE_RANDOM_DELAYS =>
    UseCoverageDelays      <= TransRec.BoolToModel ;

  when GET_USE_RANDOM_DELAYS =>
    TransRec.BoolFromModel <= UseCoverageDelays ;

  when SET_DELAYCOV_ID =>
    BurstCov          <= GetDelayCoverage(TransRec.IntToModel) ;
    UseCoverageDelays <= TRUE ;

  when GET_DELAYCOV_ID =>
    TransRec.IntFromModel <= BurstCov.ID ;
    UseCoverageDelays <= TRUE ;
```

Step 4, set default values for the coverage models.

```
TransmitHandler : process
  . . .
begin
  . . .
  wait for 0 ns ; -- Allow Cov models to initialize
  wait for 0 ns ; -- Allow Cov models to initialize
  -- BurstLength - once per BurstLength, use BurstDelay, otherwise use BeatDelay
  AddBins (BurstCov.BurstLengthCov,  80, GenBin(3,11,1)) ;    -- 80% Small Burst Len
  AddBins (BurstCov.BurstLengthCov,  20, GenBin(109,131,1));  -- 20% Large Burst Len
  -- BurstDelay - happens at BurstLength boundaries
  AddBins (BurstCov.BurstDelayCov,   80, GenBin(2,8,1)) ;   -- 80% Small delay
  AddBins (BurstCov.BurstDelayCov,   20, GenBin(108,156,1)) ; -- 20% Large delay
  -- BeatDelay - happens between each transfer it not at a BurstLength boundary
  -- These are all small
  AddBins (BurstCov.BeatDelayCov,    85, GenBin(0)) ;       -- 85% 0 delay
  AddBins (BurstCov.BeatDelayCov,    10, GenBin(1)) ;       -- 10% 1 delay
  AddBins (BurstCov.BeatDelayCov,     5, GenBin(2)) ;       --  5% 2 delay
```

Step 5, wait for the delay cycles.

```
DelayCycles := GetRandDelay(BurstCov) ;
WaitForClock(Clk, DelayCycles) ;
```

Note that whether AddBins or AddCross is used is up to the VC. AxiStreamReceiver uses AddCross for BurstDelayCov and BeatDelayCov to also randomize the Ready Before Valid control (0 = Ready Before Valid). To see the differences explore the code of AxiStreamReceiver.

Usage of Delay Coverage in an interface like

4 Usage in a test case

TbStream_SendGetRandom1.vhd was written to demonstrate the capabilities of the delay models. It is in the directory OsvvmLibraries/AXI4/AxiStream/TestCases.

4.1 Part 1: Randomize using existing VC delay coverage

Turn on Delay Coverage randomization by calling SetUseRandomDelays. Note that whether randomization is on initially or not is up to a particular VC. AxiStreamTransmitter currently has it off to support historical modes of operation.

```
------------------------------------------------------------
-- AxiTransmitterProc
--   Generate transactions for AxiTransmitter
------------------------------------------------------------
TransmitterProc : process
  variable DelayCoverageID, DelayCoverageID_random : DelayCoverageIDType ;
  variable BaseWord, BurstWord : std_logic_vector(31 downto 0) := X"0000_0000" ;
begin

  wait until nReset = '1' ;
  WaitForClock(StreamTxRec, 2) ;
  -- Use Delay Coverage Defined in the VC
  SetUseRandomDelays(StreamTxRec) ;
```

Using the Delay Coverage settings from the VC, transfer 256 words individually and 256 words in 32 bursts of 8. Note that the burst length of SendBurst (here 8) is independent from the delays set in the Delay Coverage. The burst length of SendBurst says I want to transfer 8 words on the interface. The burst length of Delay Coverage models what happens when a sequence of words is put on the interface – independent of whether the API thinks of it as a single word or burst transfer.

```
  log("Transmit 256 words") ;
  BaseWord := BaseWord + X"0001_0000" ;
  for I in 1 to 256 loop
    Send( StreamTxRec, BaseWord + I ) ;
  end loop ;

  BurstWord := BaseWord ;
  log("SendBurstIncrement 8 bursts of size 8") ;
  for i in 1 to 32 loop
    BurstWord := BurstWord + X"0000_1000" ;
    SendBurstIncrement(StreamTxRec, BurstWord, 8) ;
  end loop ;

  WaitForClock(StreamTxRec, 4) ;
```

If you run the simulation, you can tell when it changes from transferring individual words to transferring bursts by checking for TLast = 1.

4.2 Part 2: Replace the old delay coverage with new delay coverage

Get the DelayCoverageID from the VC using GetDelayCoverageID. Note that the DelayCoverageID variable has both the singleton ID as well as copy of the CoverageIDs that are also in the DelayCoveragePkg singleton.

```
GetDelayCoverageID(StreamTxRec, DelayCoverageID) ;
```

Remove all existing bins Delay Coverage bins referenced by the DelayCoverageID variable using DeallocateBins. Since DelayCoverageID has the same CoverageIDs as the singleton, this effectively removes the Delay Coverage bins from the singleton (and the VC).

```
DeallocateBins(DelayCoverageID) ;
```

Create new Delay Coverage bins using AddBins. In this case, use a constant burst length of 4, a burst delay of 4, and a beat delay of 1.

```
AddBins(DelayCoverageID.BurstLengthCov, GenBin(4)) ;
AddBins(DelayCoverageID.BurstDelayCov, GenBin(4)) ;
AddBins(DelayCoverageID.BeatDelayCov,  GenBin(1)) ;
```

Using these settings, transfer 32 words individually and transfer 64 words in bursts of 8.

```
log("Transmit 32 words") ;
BaseWord := BaseWord + X"0001_0000" ;
for i in 1 to 32 loop
  Send( StreamTxRec, BaseWord + I ) ;
end loop ;

log("SendBurstIncrement 8 bursts of size 8") ;
BurstWord := BaseWord ;
for i in 1 to 8 loop
  BurstWord := BurstWord + X"0000_1000" ;
  SendBurstIncrement(StreamTxRec, BurstWord, 8) ;
end loop ;

WaitForClock(StreamTxRec, 4) ;
```

4.3 Part 3: Create new Delay Coverage and Swap it in for the Existing One

Create Delay Coverage information in variable DelayCovID_Random using NewDelayCoverage. Note that these CoverageIDs are only in the variable and not in the DelayCoveragePkg singleton. Also note that DelayCovID_Random.ID is set to the same singleton ID used by DelayCoverageID.

```
DelayCovID_Random := NewDelayCoverage(DelayCoverageID.ID, "TxRandom", TbID);
```

Add bins to the CoverageIDs referenced in DelayCovID_Random using AddBins.

```
-- BurstLength - once per BurstLength, use BurstDelay, otherwise use BeatDelay
AddBins (DelayCovID_Random.BurstLengthCov, 80, GenBin(3,11,1)) ;   -- 80% Small
AddBins (DelayCovID_Random.BurstLengthCov, 20, GenBin(109,131,1)); -- 20% Large
-- BurstDelay - happens at BurstLength boundaries
AddBins (DelayCovID_Random.BurstDelayCov,  80, GenBin(2,8,1)) ; -- 80% Small
AddBins (DelayCovID_Random.BurstDelayCov,  20, GenBin(108,156,1)) ; -- 20% Large
-- BeatDelay - happens between each transfer it not at a BurstLength boundary
AddBins (DelayCovID_Random.BeatDelayCov,    85, GenBin(0)) ; -- 85% no delay
AddBins (DelayCovID_Random.BeatDelayCov,    10, GenBin(1)) ; -- 10% 1 cycle delay
AddBins (DelayCovID_Random.BeatDelayCov,     5, GenBin(2)) ; --  5% 2 cycle delay
```

Copy the CoverageIDs in variable DelayCovID_random to the DelayCoveragePkg singleton. Since DelayCovID_Random.ID = DelayCoverageID.ID, this copies over the original CoverageIDs in the singleton (and used by VC) – don't despair we still have a reference to the original CoverageIDs in the DelayCoverageID variable.

```
SetDelayCoverage(DelayCoverageID_Random) ;
```

Using these settings, transfer 256 words individually and 256 words in 32 bursts of 8.

```
-- Send
log("Transmit 256 words") ;
BaseWord := BaseWord + X"0001_0000" ;
for I in 1 to 256 loop
  Send( StreamTxRec, BaseWord + I ) ;
end loop ;

BurstWord := BaseWord ;
log("SendBurstIncrement 8 bursts of size 8") ;
for i in 1 to 32 loop
  BurstWord := BurstWord + X"0000_1000" ;
  SendBurstIncrement(StreamTxRec, BurstWord, 8) ;
end loop ;

WaitForClock(StreamTxRec, 4) ;
```

4.4 **Part 4: Swap in the Delay Model from Part 2**

The Delay Coverage from part 2 is still in the variable DelayCoverageID. Copy the CoverageIDs in DelayCoverageID to the DelayCoveragePkg singleton. This will use a constant burst length of 4, a burst delay of 4, and a beat delay of 1.

```
SetDelayCoverage(DelayCoverageID) ;
```

Using these settings, transfer 32 words individually and transfer 64 words in bursts of 8.

```
log("Transmit 32 words") ;
BaseWord := BaseWord + X"0001_0000" ;
for i in 1 to 32 loop
  Send( StreamTxRec, BaseWord + I ) ;
end loop ;
```

```
    log("SendBurstIncrement 8 bursts of size 8") ;
    BurstWord := BaseWord ;
    for i in 1 to 8 loop
      BurstWord := BurstWord + X"0000_1000" ;
      SendBurstIncrement(StreamTxRec, BurstWord, 8) ;
    end loop ;

    -- Wait for outputs to propagate and signal TestDone
    WaitForClock(StreamTxRec, 2) ;
    WaitForBarrier(TestDone) ;
    wait ;
  end process TransmitterProc ;
```

5 Setting BurstLength at Start up

Nominally the default BurstLength is 0. This means the first call to GetRandDelay in a VC will use a burst delay. If RandomSalt is set, then the BurstLength will be a random value between 0 and 5. A VC or test case can set the first value to be a particular using SetBurstLength.

```
-- Get Delay Coverage ID
GetDelayCoverageID(StreamTxRec, DelayCoverageID) ;
-- Set the BurstLength so that the next 5 delays use Beat Delays
SetBurstLength(DelayCoverageID, 5) ;
```

6 Running TbStream_SendGetRandom1.vhd

Build the OsvvmLibraries per the directions in the Script User Guide. Change RunDemoTests.pro that is in the directory OsvvmLibraries/AXI4/AxStream so that it reads as follows:

```
TestSuite AxiStream
library   osvvm_TbAxiStream
include ./testbench
RunTest ./TestCases/TbStream_SendGetRandom1.vhd
```

Now run this test case using:

```
build <Path-To-OsvvmLibraries>/OsvvmLibraries/AXI4/AxiStream/RunDemoTests.pro
```

Note that you can tell when it changes from transferring individual words to transferring bursts by checking for TLast = 1.

7 About DelayCoveragePkg

DelayCoveragePkg is part of the OSVVM Utility library.

DelayCoveragePkg was created by Jim Lewis of SynthWorks. Please support our work by buying your VHDL Training from SynthWorks.

DelayCoveragePkg.vhd is a work in progress and will be updated from time to time. Caution, undocumented items are experimental and may be removed in a future version.

8 Compiling and Using OSVVM Utility Library

Reference all packages in the OSVVM Utility library by using the context declaration:

```
library OSVVM ;
  context osvvm.OsvvmContext ;
```

Compilation order for OSVVM Utility Library is in OSVVM_release_notes.pdf. Rather than learning this, we recommend using the OSVVM compilation scripts.

OSVVM Utility library is released under the Apache open source license. It is free (both to download and use - there are no license fees). You can download it from osvvm.org or from our development area on GitHub (https://github.com/OSVVM/OSVVM).

If you add features to the package, please donate them back under the same license as candidates to be added to the standard version of the package. If you need features, be sure to contact us.

We also support the OSVVM user community and blogs through http://www.osvvm.org. Interested in sharing about your experience using OSVVM? Let us know, you can blog about it at osvvm.org.

9 About the Author - Jim Lewis

Jim Lewis, the founder of SynthWorks, has thirty plus years of design, teaching, and problem solving experience. In addition to working as a Principal Trainer for SynthWorks, Mr Lewis has done ASIC and FPGA design, custom model development, and consulting.

Mr. Lewis is chair of the IEEE 1076 VHDL Working Group (VASG) and is the primary developer of the Open Source VHDL Verification Methodology (OSVVM.org) packages. Neither of these activities generate revenue.

Please support our volunteer efforts by buying your VHDL training from SynthWorks.

If you find bugs these packages or would like to request enhancements, you can reach me at jim@synthworks.com.