

## CoveragePkg Quick Ref

### 1. Item Coverage

Item coverage tracks relationships within a single object. For example, tracking bins of values for a transfer may look at the ranges, 1, 2, 3, 4 to 127, 128 to 252, 253, 254, and 255. The steps to model coverage are:

- Reference packages
- Declare the coverage ID
- Construct the Coverage Model
- Model coverage
- Accumulate coverage
- Detecting when done
- Print Results

### 2. Reference Packages

Using CoveragePkg requires the following references.

```
library osvvm ;
use osvvm.CoveragePkg.all ;
```

### 3. Declare the Coverage ID

A CoverageID is a handle to the coverage object that is created internal to CoveragePkg.

```
architecture Test1 of tb is
    signal Cov1 : CoverageIDType ;
```

### 4. Construct the Coverage Model

```
Cov1 <= NewID("Cov1") ;
wait for 0 ns ; -- Update Cov1
```

### 5. Model Coverage: Item Coverage

Coverage items are added to a coverage model by calling procedure AddBins and function GenBin one or more times. GenBin transforms a bin descriptor into a set of bins. AddBins inserts these bins into the coverage data structure. The version of GenBin shown below has three parameters: min value, max value, and number of bins. The call, GenBin(1,3,3), breaks the range 1 to 3 into the 3 separate bins with ranges 1 to 1, 2 to 2, 3 to 3.

```
AddBins(Cov1, GenBin(1, 3, 3)) ;
```

Additional calls to AddBins appends bins to the data structure. As a result, the call, GenBin(4, 252, 2), appends two bins with the ranges 4 to 127 and 128 to 252 respectively to the coverage model.

```
AddBins(Cov1, GenBin( 4, 252, 2)) ;
```

GenBin with two parameter, min value and max value, creates one bin per value. As a result, the call GenBin(253, 255) appends three bins with the ranges 253 to 253, 254 to 254, and 255 to 255.

```
AddBins(Cov1, GenBin(253, 255)) ;
```

### 6. Accumulate Coverage: Item Coverage

Coverage is accumulated using the method ICover.

```
ICover(Cov1, to_integer(Data_slv)) ;
```

### 7. Detecting when Done

The method IsCovered returns true when the functional coverage reaches 100%. Generally used in a loop:

```
exit when IsCovered(Cov1); -- done?
```

### 8. Print Results

WriteBin and WriteCovHoles print the coverage results.

```
WriteBin(Cov1) ; -- Print All Bins
WriteCovHoles(Cov1) ; -- Print Holes
```

### 9. Item Coverage, Complete Example

```
library osvvm ;
use osvvm.CoveragePkg.all ;
architecture Test1 of tb is
    signal Cov1 : CoverageIDType ;
begin
    TestProc : process
    begin
        Cov1 <= NewID("Cov1") ;
        wait for 0 ns ; -- update Cov1
        AddBins(Cov1, GenBin(1, 3)) ;
        AddBins(Cov1, GenBin(4, 252, 2)) ;
        AddBins(Cov1, GenBin(253, 255)) ;
        loop
            wait until rising_edge(Clk) ;
            ICover(Cov1, to_integer(RxData)) ;
            exit when IsCovered(Cov1); -- done?
        end loop ;
        WriteBin(Cov1) ; -- Print Results
        wait ;
    end process ;
```

### 10. Cross Coverage

Cross coverage examines the relationships between different objects, such as making sure that each register source has been used with an ALU.

Collecting cross coverage only differs from item coverage in the model and accumulate steps.

### 11. Model Coverage: Cross Coverage

The method AddCross creates the cross product of the set of bins (created by GenBin) on its inputs. The code below creates an 8 x 8 cross. Each call to GenBin(0,7) creates the 8 bins: 0, 1, 2, 3, 4, 5, 6, 7. The AddCross creates the 64 bin cross product of these bins.

```
AddCross(Cov, GenBin(0,7), GenBin(0,7)) ;
```

AddCross supports crossing from 2 to 20 items.

### 12. Accumulate Coverage: Cross Coverage

To accumulate cross coverage the ICover parameter becomes an integer\_vector, with one integer value per element in the cross product.

```
ICover(Cov, (Src1, Src2)) ;
```

### 13. Cross Coverage, Complete Example

In the following example, coverage is accumulated on a transaction basis.

```
architecture Test2 of tb is
    signal Cov : CoverageIDType ;
begin
    TestProc : process
        variable Src1, Src2 : integer ;
    begin
        Cov <= NewID("Cov") ;
        wait for 0 ns ; -- update Cov
        AddCross(Cov, GenBin(0,7), GenBin(0,7));
        loop
            -- Randomize test values,
            -- see 15 Intelligent Coverage
            (Src1, Src2) := GetRandPoint(Cov) ;
            DoAluOp(TRec, Src1, Src2) ;
            ICover(Cov, (Src1, Src2)) ;
            exit when IsCovered(Cov); -- done?
        end loop ;
        WriteBin(Cov) ; -- Print Results
        ReportAlerts ;
    end process ;
```

#### 14. Constrained Random is Slower

Constrained random uses uniform randomization to generate tests. For example in the last example, we could have used two calls RandInt to generate the test values.

```
Src1 := RV.RandInt(0, 7) ;
Src2 := RV.RandInt(0, 7) ;
```

Unfortunately, constrained random tests repeat numerous test cases before generating all test cases. In general, to generate N cases, it takes "N \* log N" randomizations. The "log N" represents repeated test cases and significantly adds to simulation run times. The above calls result in around 5X redundant vectors.

#### 15. Intelligent Coverage

Ideally when we randomize a test we would like to only generate the N test cases we need. Intelligent Coverage does this by doing a random walk across the coverage model.

The method GetRandPoint randomly selects a hole in the coverage model and passes this to the stimulus generation. For item coverage it returns an integer value and for cross coverage it returns an integer\_vector value. The previous cross coverage example uses the following call to generate stimulus.

```
(Src1, Src2) := GetRandPoint(Cov) ;
```

#### 16. Weighted Intelligent Coverage

By default, the coverage goal is 1. AddBins and AddCross are overloaded to support coverage goals larger than 1. For example, the following sets each bin in the cross to have a coverage goal of 10.

```
AddCross(Cov, 10, GenBin(0, 7), GenBin(0, 7)) ;
```

The coverage goal is specified as the first parameter to AddBins or AddCross.

The coverage goal is also used as the randomization weight. Hence, bins with larger coverage goals are generated more often. This can be used to generate some conditions, transactions, or sequences more than others. Such as more normal transactions than error transactions. The following generates the value 0 70% of the time, a 1 20% of the time, and a 2 10% of the time.

```
AddBins(Cov1, 70, GenBin(0) ) ;
AddBins(Cov1, 20, GenBin(1) ) ;
AddBins(Cov1, 10, GenBin(2) ) ;
```

#### 17. Randomization Weight Mode

Weight Mode controls whether randomization uses the coverage goal for a bin as the randomization weight (AT\_LEAST mode = default) or the remaining coverage for a bin as the randomization weight (REMAIN mode = coverage goal – current coverage).

```
SetWeightMode (Cov, REMAIN);
```

#### 18. Thresholds

Rather than randomly selecting any bin whose coverage < 100%, only randomly select bins whose coverage is <= (MinCov + Threshold).

```
SetCovThreshold(Cov, 30.0);
```

Use SetThresholding to disable or enable Thresholding.

```
Cov.SetThresholding(FALSE);
```

#### 19. Increasing / Decreasing run length

By default the coverage target is 100.0. Changing the coverage target changes the coverage goals by that amount. The following increases coverage goals (and hence run length) by 10X:

```
SetCovTarget(Cov, 1000.0 ); -- = 100 * 10
```

The following decreases coverage goals by 10X:

```
SetCovTarget(Cov, 10.0 ); -- = 100 / 10
```

#### 20. Illegal Bins

Illegal bins signal the receipt of illegal (invalid) values. Illegal bins are not for handled errors (such as parity errors). Illegal bins are never generated by Intelligent Coverage.

The method IllegalBin (similar to GenBin) designates a bin to be an illegal bin. The following creates the range 1 to 9 in a single illegal bin.

```
IllegalBin(1, 9)
```

The method SetIllegalMode controls signaling when a value is in an illegal bin. See the CoveragePkg User Guide for details

Errors detected by the coverage model are reported as alerts and are reported by ReportAlerts.

#### 21. GetCov

Returns the total coverage for all coverage models:

```
TotalCov := GetCov ;
```

#### 22. Setting Coverage Weights

The coverage weight is a multiplying factor in determining the total coverage. CovWeight is 1 by default. Set it to 0 to ignore the coverage model in total coverage calculations

```
SetCovWeight (Cov, 0);
```

#### 23. Set / Get the Internal AlertLogID

The internal AlertLogID is set using the name specified by NewID. Read it with GetAlertLogID.

```
Cov <= NewID("Cov") ;
. . .
```

```
CovAlertID <= GetAlertLogID(Cov) ;
```

#### 24. Constants Coverage Type

The return value from GenBin is of type CovBinType. It is defined as follows.

```
type CovBinType is array
  (natural range <>) of CovBinBaseType;
```

This allows the definition of constants.

```
constant REG_COV : CovBinType :=
  GenBin(0, 7) & GenBin(9) ;
```

#### 25. Naming Individual Bins

Individual bin names are specified as follows:

```
AddBins(Cov1, "PARITY_ERROR", . . . ) ;
AddCross(Cov2, "S1 -> S2", . . . ) ;
```

#### 26. Item Bin Names (Columns in the report)

Specify one name per item bin.

```
SetItemBinNames(Cov1, "Bin1", . . . ) ;
```

#### 27. Test Case Detailed Reports

Call EndOfTestReports (see AlertLogPkg) at the end of a test to get a report for each coverage model in the test case in Reports/<TestName>.html.

#### 28. More Detailed Information

See: CoveragePkg\_Users\_Guide.pdf.

© 2015 - 2021 by SynthWorks Design Inc. Reproduction of entire document in whole permitted. All other rights reserved.

**SynthWorks Design Inc.**  
VHDL Intro, RTL, and Verification Training

11898 SW 128<sup>th</sup> Ave. Tigard OR 97223 +1-503-590-4787

<http://www.SynthWorks.com> [jim@synthworks.com](mailto:jim@synthworks.com)