SynthWorks
VHDL Training Experts

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 object
- Model coverage
- Accumulate coverage
- Detecting when done
- Print Results

2. Reference Packages
Using CoveragePkg requires the following references. To compile these see section **Error! Reference source not found. Error! Reference source not found.**.

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

3. Declare the Coverge Object
Coverage is modeled using a data structure stored inside of a coverage object. The coverage object is a shared variable of type CovPType, such as Cov below.

```
architecture Test1 of tb is
  shared variable Cov : CovPType ;
```

4. Model Coverage: Item Coverage
A coverage model is constructed by using one or more calls to the method AddBins and the function GenBin. GenBin takes a range and breaks it into a set of bins. AddBins inserts these bins into the coverage model. 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.

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

Additional calls to AddBins appends additional 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.

```
Cov1.AddBins(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.

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

5. Accumulate Coverage: Item Coverage
Coverage is accumulated using the method ICover. The following example uses clock based sampling.

```
loop
  wait until rising_edge(Clk) and E='1';
  Cov1.ICover(to_integer(Data_slv)) ;
end loop ;
```

6. Detecting when Done
The method IsCovered returns true when the functional coverage reaches 100%. The example below uses IsCovered to exit when done

```
while not Cov1.IsCovered loop
  wait until rising_edge(Clk) and E='1';
  Cov1.ICover(to_integer(Data_slv)) ;
end loop ;
```

7. Print Results
WriteBin is used to print the coverage results.

```
Cov1.WriteBin ; -- Print Results
```

WriteCovHoles is used to print just the hole information.

```
Cov1.WriteCovHoles ; -- Print Holes
```

8. Item Coverage, Complete Example

```
library osvvm ;
use osvvm.CoveragePkg.all ;
architecture Test1 of tb is
  shared variable Cov1 : CovPType ;
begin
  TestProc : process
  begin
    Cov1.AddBins(GenBin(1, 3)) ;
    Cov1.AddBins(GenBin(4, 252, 2)) ;
    Cov1.AddBins(GenBin(253, 255 )) ;
    while not Cov1.IsCovered loop --Done?
      wait until rising_edge(Clk) ;
      Cov1.ICover(to_integer(RxData)) ;
    end loop ;
    Cov1.WriteBin ; -- Print Results
    wait ;
  end process ;
```

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

The steps for modeling cross coverage are the same steps used for item coverage: declare, model, accumulate, interact, and report. Collecting cross coverage only differs in the model and accumulate steps.

10. 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.

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

AddCross supports crossing from 2 to 20 items.

11. 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.

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

12. Cross Coverage, Complete Example

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

```
architecture Test2 of tb is
  shared variable Cov : CovPType ;
begin
  TestProc : process
    variable Src1, Src2 : integer ;
  begin
   Cov.AddCross(GenBin(0,7),GenBin(0,7));
    while not Cov.IsCovered loop
      -- Generate test values,
      -- see 14 Intelligent Coverage
      (Src1, Src2) := Cov.GetRandPoint ;
      DoAluOp(TRec, Src1, Src2) ;
      Cov.ICover( (Src1, Src2) ) ;
    end loop ;
    ACov.WriteBin ;  -- Report
    EndStatus(. . . ) ;
  end process ;
```

13. 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) ;
```

The problem with constrained random testbenches is that they repeat 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.

14. 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) := Cov.GetRandPoint ;
```

15. 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.

```
Cov.AddCross(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.

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

16. Randomization Weights
Randomization Weights help balance randomization when coverage goals are > 1. By default the coverage goal is used at the randomization weight. This behavior can be changed by the method SetWeightMode. There are currently two relevant values: AT_LEAST (default, use coverage goal as weight) and REMAIN (use coverage goal – current coverage as the weight).

```
Cov.SetWeightModel(REMAIN);
```

See also Thresholds.

17. Thresholds
Thresholds are designed to balance randomization when coverage goals are > 1. Rather than randomly selecting any bin whose coverage < 100%, only bins that are <= (MinCov + Threshold) are randomly selected.

```
Cov.SetCovThreshold(30.0); -- default 45
```

Thresholding can be disabled (FALSE) or enabled (TRUE) by using SetThresholding.

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

See also Randomization Weights.

18. Increasing / Decreasing run length
Adjusting coverage target increases or decreases run length. Coverage target is a multiplier that impacts each coverage goal. Default value is 100.0 which indicates use the coverage goal as it is. The following increases run length by 10X:

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

The following decreases run length by 10X:

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

19. 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 can reported using ReportAlerts.

20. Setting the Internal AlertLogID
Errors signaled by CoveragePkg use Alerts and the internal AlertLogIDVar. By default the value is OSVVM_ALERT_LOG_ID. It can be set to something different by SetAlertLogID. There are two forms of SetAlertLogID. The first form is intended to be used when an AlertLogID is shared with other items in the testbench. It is called as follows.

```
    Cov.SetAlertLogID(UartID) ;
```

The second form is intended to create an AlertLogID that is exclusive to the coverage model. It is called as follows.

```
    Cov.SetAlertLogID("UartCov", UartID);
```

21. 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) ;
```

22. Reports and Files
By default WriteBin and WriteCovHoles use the file specified by TranscriptPkg (either TranscriptFile, OUTPUT, or both if Mirroring is enabled).

They also support usage of local file by using a string based file parameter:

```
Cov1.WriteBin("Test1.txt", WRITE_MODE)  ;
```

See the user guide for more options.

23. Additional Features

See CoveragePkg_Users_Guide.pdf for more information.