Creating High Fidelity Cross Coverage Models

by

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

SynthWorks

High Fidelity Cross Coverage

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/downloads

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

www.SynthWorks.com

High Fidelity Cross Coverage

SynthWorks

Goal

- Understand why we need functional coverage
- Develop High Fidelity Cross Coverage Models
- Essential to tell us we have tested all requirements / features

Topics

- What about Code coverage?
- What is Functional Coverage?
- Writing Coverage
  - Tracking Transfer Sizes
  - Tracking UART Status Conditions
  - Tracking ALU Input Register Pairs
  - Tracking CPU Coverage
- Randomizing Stimulus Using Coverage
  - ALU Stimulus, Equal Weights
  - UART Stimulus, Different Weights

What about Code Coverage?

SynthWorks

- Code coverage is automatically collected by the simulator, however, ...

```
PrioritySel : process (SelA, SelB, SelC, A, B, C)
begin
  Y <= "00000000" ;
  if (SelC = '1') then
    Y <= C ;
  end if ;
  if (SelB = '1') then
    Y <= B ;
  end if ;
  if (SelA = '1') then
    Y <= A ;
  end if ;
end process ;
```

Issues:
- SelA = SelB = SelC = 1, Cov = 100%
- Delta cycle glitches are recorded as Cov

What we do know:
- < 100% coverage, need more tests
- = 100%, ???

Other Issues:
- Cannot detect missing features
- Cannot detect things not in code
  - Independent ports both at worst case

What is Functional Coverage?

SynthWorks

- Code that measures requirements and features (design and test)
  - Boundary conditions of different IP cores in worst case operation
- What to Measure?
  - Transfer size: 1, 2, 3, 4-127, 128-252, 253, 254, 255 (test requirement)
  - UART: All normal & error conditions observed in status register
  - ALU: Every source input pair has occurred
  - CPU: Mixture of sources and immediate values
- Classification
  - Point/Item coverage = covering a single value
  - Cross coverage = covering multiple different values
- When to Collect Coverage
  - At an event (rising_edge(Clk))
  - When a transaction completes
- 100 % Functional Coverage + 100 % Code Coverage = Done

Tracking Transfer Sizes

SynthWorks

- Worst case conditions occur when transfer sizes are smaller or larger

Transfer Sizes	Count
1	
2	
3	
4 to 127	
128 to 252	
253	
254	
255	

- Methods:
  - Manual
  - Using CoveragePkg

Tracking Transfer Sizes: Manual

SynthWorks

```
signal Bin : integer_vector(1 to 8) ;
. . .
process
begin
  wait until rising_edge(Clk) and DValid = '1' ;
  case to_integer(ActualData) is
    when   1 =>          Bin(1) <= Bin(1) + 1 ;
    when   2 =>          increment( Bin(2) ) ;
    when   3 =>          increment( Bin(3) ) ;
    when   4 to 127 =>   increment( Bin(4) ) ;
    when 128 to 252 =>   increment( Bin(5) ) ;
    when 253 =>          increment( Bin(6) ) ;
    when 254 =>          increment( Bin(7) ) ;
    when 255 =>          increment( Bin(8) ) ;
    when others =>       null ;
  end case ;
end process ;
```

Declaration of 8 Bins

Sampling

Bin Creation

Issues: Too much work, Specific to a problem, No reuse, No built-in reporting

CoveragePkg

SynthWorks

- Simplifies modeling and collection of coverage.
- Contains functions, procedures and a protected type.
  - The protected type encapsulates the data structure and configuration

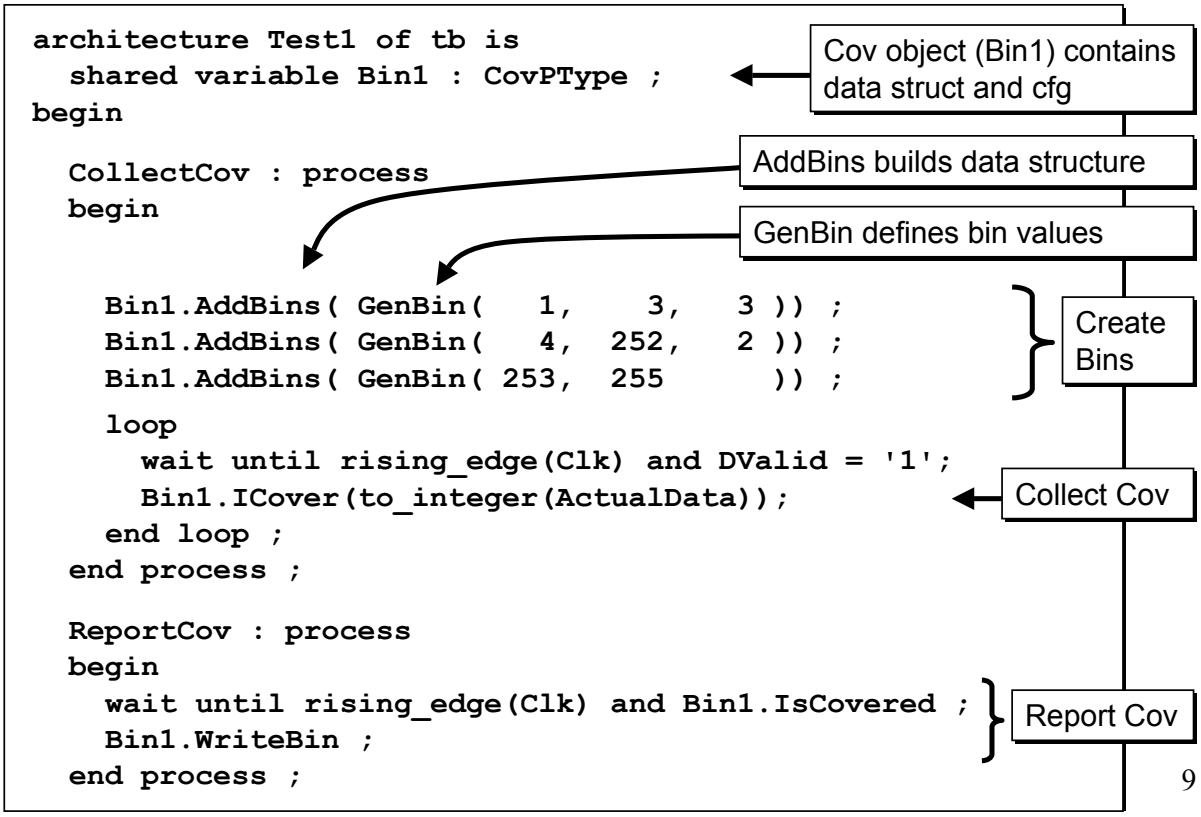
```
function GenBin ( . . . ) return CovBinType ;

type CovPType is protected
  procedure AddBins ( CovBin : CovBinType ) ;
  procedure AddCross ( Bin1, Bin2, ... : CovBinType ) ;
  procedure ICover  ( val : integer_vector ) ;
  procedure ICover  ( val : integer ) ;
  impure function IsCovered ( ... ) return boolean ;
  procedure WriteBin ;
  procedure ReadCovDb     ( FileName : string ) ;
  procedure WriteCovDb    ( FileName : string; ... ) ;
  . . .
end protected CovPType ;
```

- Usage of the methods and functions replace the need for language syntax

Tracking Transfer Sizes: CoveragePkg

SynthWorks



Bin Construction: AddBins + GenBin

SynthWorks

- Method AddBins: Creates internal data structure in CovPType.
- GenBin: Creates an array of bins, the input to AddBins

```
--                          min, max, #bins
CovBin1.AddBins( GenBin(  1,   3,   3   ));
```

  - Create 3 bins with ranges: 1 to 1, 2 to 2, and 3 to 3 .
- Consecutive calls to AddBins creates additional bins

```
CovBin1.AddBins( GenBin( 4, 252, 2 ) );
```

  - Creates 2 additional bins with ranges: 4 to 127, 128 to 252.
- GenBin without NumBins creates one bin per value

```
CovBin1.AddBins( GenBin( 253, 255 ) );
```

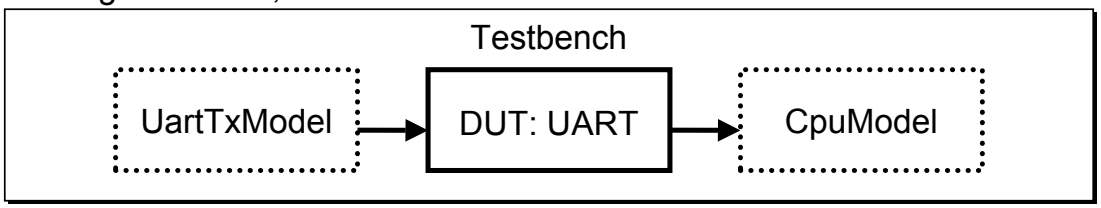
  - 3 additional bins with ranges: 253 to 253, 254 to 254, and 255 to 255.
- GenBin a single parameter creates one bin: 1 to 1

```
CovBin1.AddBins( GenBin( 1 ) );
```

Tracking UART Status Conditions

SynthWorks

- Testing interfaces, here a UART:



- For the UART, we track the following items

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

- Result: List of conditions that must be covered

Tracking UART Status Conditions

SynthWorks

```
architecture Test2 of tb is
  shared variable UCov : CovPType ; -- Cov Object
begin

 CollectCov : process
 begin
  -- Create Coverage Bins, Binary
  UCov.AddCross(GenBin(0),GenBin(0),GenBin(0),GenBin(1));--Normal
  UCov.AddCross(GenBin(0),GenBin(0),GenBin(1),GenBin(1));--Parity
  UCov.AddCross(GenBin(0),GenBin(1),GenBin(0),GenBin(1));--Stop
  UCov.AddCross(GenBin(0),GenBin(1),GenBin(1),GenBin(1));--SE+PE
  UCov.AddCross(GenBin(1),  ALL_BIN,  ALL_BIN,GenBin(1));--Break

  while not TestDone loop

    CpuUartGet(RxRec, Data, Status) ;

    UCov.ICover(to_integer_vector(status)) ;

  end loop ;

  UCov.WriteBin ; -- Report Coverage
  EndStatus(. . . ) ;
end process ;
```

Cross Coverage Bins

Rx Transaction

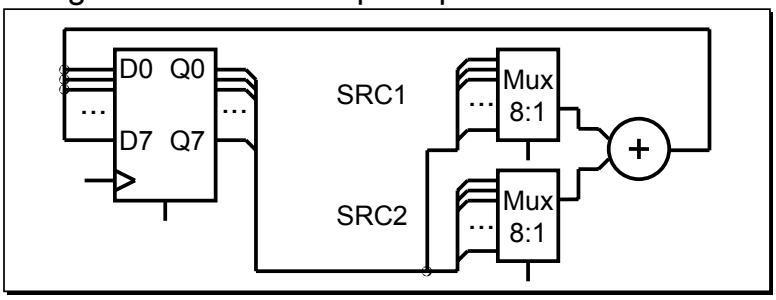
Collect Cov

Coverage sampling is transaction based.

Tracking ALU Input Register Pairs

SynthWorks

- Testing an ALU with Multiple Inputs:



- Need to test every register in SRC1 with every register in SRC2

SRC2

SRC1	R0	R1	R2	R3	R4	R5	R6	R7
R0								
R1								
R2								
R3								
R4								
R5								
R6								
R7								

- Result: Matrix of conditions that must be covered

Tracking ALU Input Register Pairs

SynthWorks

```
architecture Test3 of tb is
  shared variable ACov : CovPType ;      -- Cov Object
begin
 CollectCov : process
 begin
   -- Create Coverage Bins
   ACov.AddCross( GenBin(0,7), GenBin(0,7) );

  while not Done loop
    -- Code to generate RegIn1 & RegIn2 - more later
    DoAluOp(TRec, RegIn1, RegIn2) ;
    ACov.ICover( (RegIn1, RegIn2) ) ;
  end loop ;
  ACov.WriteBin ;                        -- Report Coverage
  EndStatus(. . . ) ;
end process ;
```

Create Bins 8x8 Matrix

Do Transaction

Collect Cov

Matrix coverage creates many bins in a quick simple format.

CPU Coverage

SynthWorks

- Testing an CPU

Instruction	Mnemonic	Parameters
Load Word	LW	Register Target, Register Base, 16-bit offset
Store Word	SW	Register Target, Register Base, 16-bit offset
Add	ADD	Register Target, Register Src1, Register Src2
Subtract	SUB	Register Target, Register Src1, Register Src2
Branch Equal	BEQ	Register Src1, Register Src2, 16-bit offset
Branch Not Equal	BNE	Register Src1, Register Src2, 16-bit offset

- Coverage
  - LW/SW: test two register variations with a set of offset values
  - ADD/SUB: test three register variations
  - BEQ/BNE: test two register variations with a set of offset values
- Result: Hybrid coverage = Mixture of lists and matricies
  - Not all instructions use all of the fields.

CPU Coverage

SynthWorks

- Constants to simplify coverage capture

```
-- Individual bins for each instruction
constant LW_INST  : CovBinType := GenBin(0) ;
constant SW_INST  : CovBinType := GenBin(1) ;
constant ADD_INST : CovBinType := GenBin(2) ;
constant SUB_INST : CovBinType := GenBin(3) ;
constant BEQ_INST : CovBinType := GenBin(4) ;
constant BNE_INST : CovBinType := GenBin(5) ;

-- 1 bin per register
constant REG_BIN  : CovBinType := GenBin(0,7) ;

-- Offset values in bins with
-- individual small & large values
constant SIGN_OFF : CovBinType :=
    GenBin(-4,4,9) &                -- small
    GenBin(5,2**15-5,4) &           -- medium positive
    GenBin(-5,-2**15-4,4) &         -- medium negative
    GenBin(2**15-4,2**15-1,4) &     -- large positive
    GenBin(-2**15-3,-2**15,4) ;     -- large negative
```

Instruction Bins

Register Bins

Offset Bins

CPU Coverage

SynthWorks

```
architecture Test3 of tb is
  shared variable CCov : CovPType ;  -- Cov Object
  . . .
begin
 CollectCov : process
 begin
  --              Inst       Reg1      Reg2      Reg3      Offset
  CCov.AddCross( LW_INST,  REG_BIN, REG_BIN, ALL_BIN, SIGN_OFF );
  CCov.AddCross( SW_INST,  REG_BIN, REG_BIN, ALL_BIN, SIGN_OFF );
  CCov.AddCross( ADD_INST, REG_BIN, REG_BIN, REG_BIN, ALL_BIN  );
  CCov.AddCross( SUB_INST, REG_BIN, REG_BIN, REG_BIN, ALL_BIN  );
  CCov.AddCross( BEQ_INST, REG_BIN, REG_BIN, ALL_BIN, SIGN_OFF );
  CCov.AddCross( BNE_INST, REG_BIN, REG_BIN, ALL_BIN, SIGN_OFF );

  while not Done loop
    GetCpuInst(MonitorRec, Inst, Reg1, Reg2, Reg3, Offset) ;
    CCov.ICover((Inst, Reg1, Reg2, Reg3, Offset)) ;
  end loop ;
  CCov.WriteBin ;  -- Report Coverage
  . . .
```

Cross Coverage Bins

Collect Cov

Randomizing Stimulus Using Coverage

SynthWorks

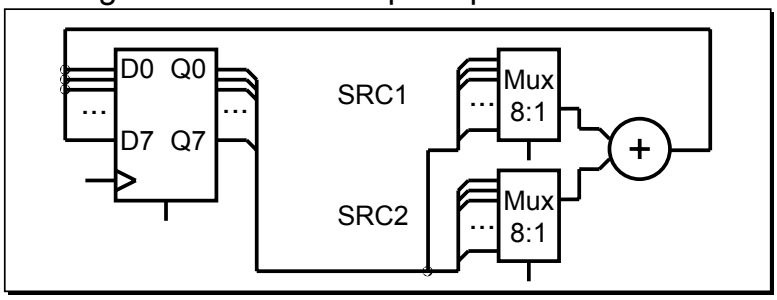
- Constrained random tests generate some conditions many times
  - To cover n conditions requires O(n*logn) in time
  - With additional EDA tools ($$$$$), it can approach O(n)
- To approach O(n) with CoveragePkg, we add goals and randomization

```
type CovPType is protected
  . . .
  procedure AddBins (
    AtLeast : integer ;
    CovBin : CovBinType
  );
  procedure AddCover (
    AtLeast : integer ;
    Bin1, Bin2, ... : CovBinType
  ) ;
  impure function IsCovered (...) return boolean ;
  impure function RandCovPoint (...) return integer_vector ;
  . . .
end protected CovPType ;
```

ALU Stimulus, Equal Weights

SynthWorks

- Testing an ALU with Multiple Inputs:



- Goal: only cover each test case one time or same number of times

SRC1 \ SRC2	R0	R1	R2	R3	R4	R5	R6	R7
R0								
R1								
R2								
R3								
R4								
R5								
R6								
R7								

ALU Stimulus, Equal Weights

SynthWorks

```
architecture Test3 of tb is
  shared variable ACov : CovPType ; -- Cov Object
begin
 CollectCov : process
   variable Reg1, Reg2 : integer ;
 begin
  ACov.AddCross(2, GenBin(0,7), GenBin(0,7) );

  while not ACov.IsCovered loop

    (Reg1, Reg2) := ACov.RandCovPoint ;

    DoAluOp(TRec, Reg1, Reg2) ;

    ACov.ICover( (Reg1, Reg2) ) ;
  end loop ;
  ACov.WriteBin ; -- Report Coverage
  EndStatus(. . . ) ;
end process ;
```

Weighted Coverage Matrix 8x8

Covered = Done

Rand Uncovered

Do Transaction

Mark it Covered

RandCovPoint only generates uncovered values

UART Stimulus, Different Weights

SynthWorks

- For the UART, we want more normal transactions than others:

Condition	Frequency	Data Value	Idle Time
Normal Transfer	63	0 to 255	0
Normal Transfer	10	0 to 255	1 to 15
Parity Error	10	0 to 255	2 to 15
Stop Error	10	1 to 255	2 to 15
Parity & Stop Error	5	1 to 255	2 to 15
Break Error	2	11 to 30*	2 to 15

* note for a break error, the value we will receive is 0

- Alternate approach, represent status/conditions as integers

```
constant NORMAL : CovBinType := GenBin(1) ;       -- 0001
constant PARITY : CovBinType := GenBin(3) ;       -- 0011
constant STOP   : CovBinType := GenBin(5) ;       -- 0101
constant PE_SE  : CovBinType := GenBin(7) ;       -- 0111
constant BREAK  : CovBinType := GenBin(9,15,1) ; -- 1--1
```


UART Stimulus, Different Weights

SynthWorks

```
architecture Test2 of tb is
  shared variable UCov : CovPType ; -- Cov Object
  . . .
begin
  CollectCov : process
  begin
    UCov.AddCross(63, NORMAL, GenBin(0,255,1), GenBin(0));
    UCov.AddCross(10, NORMAL, GenBin(0,255,1), GenBin(1,15,1));
    UCov.AddCross(10, PARITY, GenBin(0,255,1), GenBin(2,15,1));
    UCov.AddCross(10, STOP,   GenBin(1,255,1), GenBin(2,15,1));
    UCov.AddCross( 5, PE_SE,  GenBin(1,255,1), GenBin(2,15,1));
    UCov.AddCross( 2, BREAK,  GenBin(0),       GenBin(2,15,1));

    while not UCov.IsCovered loop

      CpuUartGet(RxRec, Data, Status) ;

      UartSboard.CheckActualData((Data, Status)) ;

      UCov.ICover((Status, Data, IdleTime)) ;
    end loop ;
    . . .
  end process ;
```

Create Coverage Bins with AtLeast

Covered = Done

Get Transaction

Accumulate Cov

UART Stimulus, Different Weights

SynthWorks

```
-- continued architecture Test2 of tb is

 GenStim : process
   variable Status, Data, Idle : integer ;
   . . .
 begin

  while not UCov.IsCovered loop

    (Status, Data, Idle) := UCov.RandCovPoint;

    if Status = BREAK_ERROR then
      Data := RV.RandInt(11,30) ;
    end if ;

    IdleTime <= Idle ; -- passed to RX side

    UartSboard.PutExpectedData((Data, Status)) ;

    UartSend(TxRec, Data, Idle, Status) ;
  end loop ;

  wait ;
end process ;
```

Covered = Done

Rand Uncovered

Adjust Randomization

Do Transaction

RandCovPoint uses AtLeast to generate weights for randomization.

Summary

SynthWorks

- In VHDL we build functional coverage using data structures.
  - Data structure is hidden within a protected type
  - Incrementally build a high fidelity coverage model
- Basic methods provide similar capability to SystemVerilog and 'e'
- Advanced methods provide better than SystemVerilog and 'e' capability.
  - Randomly select coverage holes, forwarding to stimulus generation
  - Focus on generating missing stimulus
  - Test time approaches O(n) (significant speed-up)
  - Gives us "Intelligent Testbench" like capability using a basic simulator
- CoveragePkg is available as Open Source
  - http://www.SynthWorks.com/downloads
  - Presentation is based on version 2.2 of CoveragePkg.
- Since it is open source, the code is reviewable.
- Use with your current VHDL testbench
- No additional licenses required