System Verilog

COVERAGE

Coverage

- Coverage is the metric of completeness of verification.

- Why we need coverage?
  - Direct Testing is not possible for complex designs.
  - Solution is constrained random verification but :
    - How do we make sure what is getting verified?
    - Are all importance design states getting verified?

- Types of Coverage's:
  - Code Coverage.
  - Functional Coverage.

Code Coverage

- Code Coverage is a measure used to describe how much part of code has been covered (executed).

- Categories of Code Coverage
  - Statement coverage
    - Checks whether each statement in source is executed.

  - Branch coverage
    - Checks whether each branch of control statement (if, case) has been covered.
    - Example: choices in case statements.

Code Coverage

- Condition coverage
  - Has Boolean expression in each condition evaluated to both true and false.
- Toggle coverage
  - Checks that each bit of every signal has toggled from 0 to 1 and 1 to 0.
- Transition coverage
  - Checks that all possible transitions in FSM has been covered.
- State coverage
  - Checks that all states of FSM has been covered.

Functional Coverage

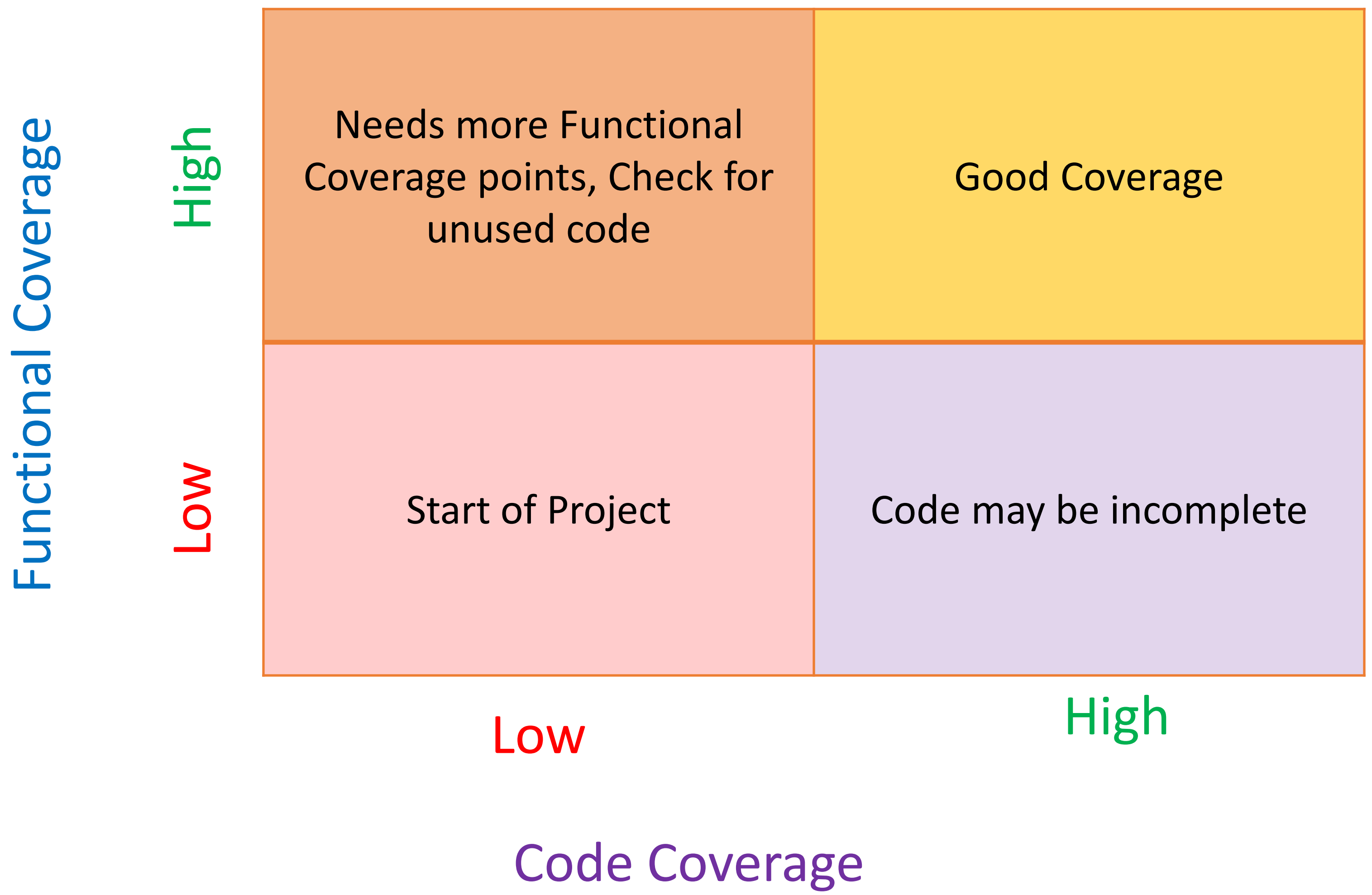
- Functional Coverage is used to verify that DUT meets all the described functionality.

- Functional Coverage is derived from design specifications.
  - ○ DUT Inputs : Are all interested combinations of inputs injected.
  - ○ DUT Outputs : Are all desired responses observed from every output port.
  - ○ DUT internals : Are all interested design events verified. e.g. FIFO full/empty, bus arbitration.

Examples

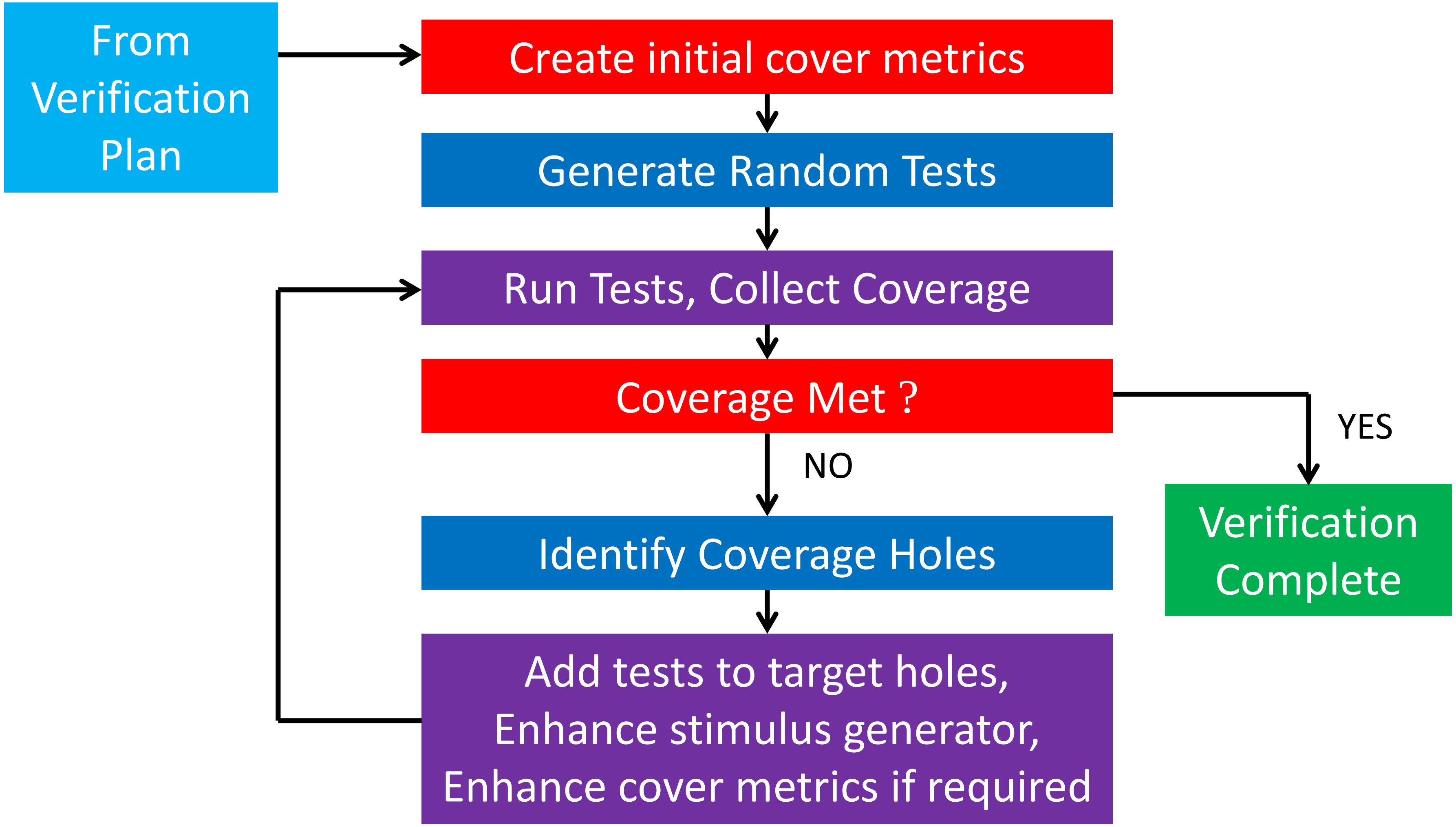
- Have I exercised all the protocol request types and combinations?
  - Burst reads, writes etc.
- Have we accessed different memory alignments?
  - Byte aligned, word aligned, dword aligned, etc.
- Did we verify sequence of transactions?
  - Reads followed by writes.
- Did we verify queue full and empty conditions?
  - input and output queues getting full and new requests getting back pressured.

Code vs. Functional Coverage



Functional Coverage		Code Coverage: Low	Code Coverage: High
	High	Needs more Functional Coverage points, Check for unused code	Good Coverage
	Low	Start of Project	Code may be incomplete

Coverage Driven Verification



From Verification Plan
Create initial cover metrics
Generate Random Tests
Run Tests, Collect Coverage
Coverage Met ?
YES
NO
Verification Complete
Identify Coverage Holes
Add tests to target holes, Enhance stimulus generator, Enhance cover metrics if required

SV Functional Coverage Support

- The System Verilog functional coverage constructs provides:
    - Coverage of variables and expressions, as well as cross coverage between them.
    - Automatic as well as user-defined coverage bins.
    - Associate bins with sets of values, transitions, or cross products.
    - Events and sequences to automatically trigger coverage sampling.
    - Procedural activation and query of coverage.
    - Optional directives to control and regulate coverage.

covergroup

- covergroup construct encapsulates the specification of a coverage model.

- covergroup is a user defined type that allows you to collectively sample all variables/transitions/cross that are sampled at the same clock (or sampling) edge.

- It can be defined inside a package, module, interface, program block and class.

- Once defined, a covergroup instance can be created using new() - just like a class.

Example

Syntax:

```
covergroup cg_name [(port_list)]
[coverage_event];
//coverage_specs;
//coverage_options;
endgroup  [ : cg_name]
```

Example:

```
covergroup cg;
……
endgroup

cg cg1=new;
```

Coverpoint

- A coverage point (coverpoint) is a variable or an expression that functionally covers design parameters.
- Each coverage point includes a set of bins associated with its sampled values or its value-transitions.
- The bins can be automatically generated or manually specified.
- A covergroup can contain one or more coverpoints.

Example

Syntax:

```
[label : ] coverpoint expression [ iff (expression)]
[{
//bins specifications;
}] ;
```

Example:

```
covergroup cg;
coverpoint a iff (!reset);
endgroup
```

bins

- bins are buckets which are used to collect number of times a particular value/transaction has occurred.
- bins allows us to organize coverpoint sample values in different ways.
  - Single value bins.
  - Values in a range, multiple ranges.
  - Illegal values, etc.
- If bins construct is not used inside coverpoint then automatic bins are created based on the variable type and size.

- For a n-bit variable, 2 ^ n automatic bins are created.

Example1

```
bit [3:0] temp;
covergroup cg;
coverpoint temp;       //16 - Automatic bins created
endgroup

cg cg1;
initial cg1=new;
```

bin[0] to bin[15] are created where each bin stores information of how many times that number has occurred.

Example2

```
bit [3:0] temp;
covergroup cg;
coverpoint temp
{
bins a= { [0 : 15] };      //creates single bin for values 0-15
bins b [ ]= { [0 : 15] };  //creates separate bin for each
                           //value 0-15
}
endgroup
```

Example3

```
bit [3:0] temp;
covergroup cg;
coverpoint temp
{
bins a [ ]= { 0, 1, 2 };          //creates three bins 0, 1, 2
bins b [ ]= { 0, 1, 2, [1:5] };   //creates eight bins 0, 1, 2,
                                   //1, 2, 3, 4, 5

}
endgroup
```

Example4

```
bit [3:0] temp;
covergroup cg;
coverpoint temp
{
bins a [4]= { [1:10], 1, 5, 7 };
//creates four bins with distribution <1, 2, 3>   <4, 5, 6>
<7, 8, 9>   <10, 1, 5, 7>
}
endgroup
```

Example5

```
bit [9:0] temp;
covergroup cg;
coverpoint temp
{
bins a = { [0:63], 65 };                // single bin
bins b [ ]={ [127:150], [148:191] }; // overlapping multiple bins
bins c [ ]={ 200, 201, 202 };        // three bins
bins d [ ]={ [1000:$] };               // multiple bins from 1000
                                   // to  $(last value:1023)
bins others [ ] = default;              // bins for all other value
}
endgroup
```

Questa (How to obtain coverage)

```
vlog +cover filename.sv
vsim –c modulename –do “run time; coverage report –details;”
//Provides Function coverage, -details switch is used to observe
bins

vsim –c –cover modulename –do “run time; coverage report
–details; ”              // -cover switch enables code coverage

vsim –c  –cover modulename –do “run time; coverage report
–details –html;”              //create html report for coverage
```

Sample code (dUT)

- module chk_cvg(input clk,input [2:0] a,b, output reg [3:0]out);

- always @(posedge clk) begin
- out = a+b;
- end
- endmodule

Sample code (tB)

- module tb_cvg;

- reg [2:0] a,b;
- wire [3:0]out;
- bit clk =0;

- chk_cvg C1(clk,a,b,out); //dut

- always #5 clk = ~clk;
- covergroup cg @(posedge clk); // covergroup
- //option.per_instance =1;

- coverpoint a;
- coverpoint b;
- coverpoint out;
- endgroup

```
initial begin
cg cg1;
cg1  =new();
end

initial begin

repeat(10) begin
a= $random;
b= $random;
#10;
end
end
endmodule
```