Session 4 Exercise

fares.sultan9@gmail.com Switch account

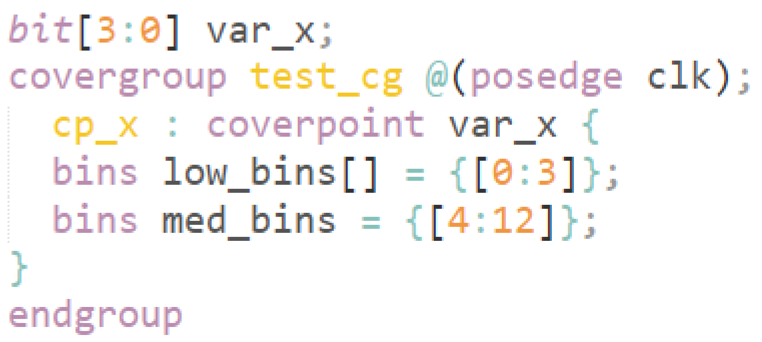
Not shared

* Indicates required question

Name *

Choose

How many bins are created in following examples for coverpoint cp_x ? * 1 point



```
bit[3:0] var_x;
covergroup test_cg @(posedge clk);
  cp_x : coverpoint var_x {
  bins low_bins[] = {[0:3]};
  bins med_bins = {[4:12]};
}
endgroup
```

- ◯ 4
- ◯ 3
- ◯ 5
- ◯ 13

ignore_bins are used to specify a set of values or transitions associated with a coverage point that can be marked as illegal * 1 point

- ◯ True
- ◯ False

What are the transitions covered by following coverpoint? * 1 point

```
coverpoint my_variable {
  bins trans_bin[] = ( a,b,c => x, y);
}
```

- ◯ a=>b, b=>c, c=>x, x=>y
- ◯ a=>x, a=>y, b=>x, b=>y, c=>x, c=>y
- ◯ None above

How many bins are created in following examples for cross cc_a_b ? * 1 point

```
bit [31:0] a_var;
bit [3:0] b_var;
covergroup cov3 @(posedge clk);
  cp_a: coverpoint a_var {
  bins yy[] = { [0:9] };
  }
  cp_b: coverpoint b_var;
  cc_a_b : cross cp_b, cp_a;
endgroup
```

- ○ 60
- ○ 160
- ○ 40
- ○ 320

Given the assertion X ##1 A [->3] ##1 Y, what does this assertion check for in SystemVerilog Assertions? * 1 point

- ○ Event X occurs, followed by 1 to 3 occurrences of event A (it's mandatory to be consecutive repetition), and then Y happens after clock cycle
- ○ Event X occurs, followed by at least 3 occurrences of event A (it's not mandatory to be consecutive repetition) , and then Y happens after 1 clock cycle but must follow immediately
- ○ Event X occurs, followed by at least 3 occurrences of event A (it's not mandatory to be consecutive repetition) , and then Y happens after 1 clock cycle but musn't follow immediately

What does the assertion X ##1 A [*3] ##1 Y check for? * 1 point

- ◯ Event X occurs, followed by exactly 3 consecutive cycles of event A, and then Y occurs one cycle later.
- ◯ Event X occurs, followed by at least 3 occurrences of event A (it's not mandatory to be consecutive repetition) , with one cycle delay between each, and then Y occurs one cycle later.
- ◯ Event X occurs, followed by exactly 3 consecutive cycles of event A, then Y happens after 1 clock cycle but musn't follow immediately

Consider the following sequence assertion: X ##1 A [=3] ##1 Y. * 1 point

What does this sequence assertion require

- ◯ Event X occurs, followed by exactly 3 consecutive occurrences of event A, and then Y occurs one cycle later but must follow immediately
- ◯ Event X occurs, followed by exactly 3 occurrences of event A (which may not be consecutive), with one cycle delay between each occurrence, and then Y occurs one cycle later but mustn't follow immediately
- ◯ Event X occurs, followed by at least 3 consecutive occurrences of event A, and then Y occurs with no delay
- ◯ Event X occurs, followed by exactly 2 to 3 occurrences of event A, and then Y occurs one cycle later but must follow immediately.

Which of the following is NOT a type of SystemVerilog assertion? * 1 point

- ◯ Immediate Assertion
- ◯ Concurrent Assertion
- ◯ Functional Assertion

What is the primary purpose of an assertion in SystemVerilog? * 1 point

○ To provide a mechanism for data manipulation

○ To ensure design correctness and check for expected behavior

○ To perform arithmetic operations

○ To optimize simulation speed

Which SystemVerilog assertion type is used to validate behavior over a sequence of time? * 1 point

○ Concurrent Assertion

○ Immediate Assertion

○ Assertion Checker

○ Formal Assertion

Which assertion directive helps in identifying unreachable states in design verification? * 1 point

○ assert property

○ assume property

○ cover property

What is the primary goal of coverage convergence in SystemVerilog? * 1 point

- ○ To ensure all code paths are executed at least once
- ○ To verify that all possible assertions are checked
- ○ To achieve a high level of coverage and ensure thorough verification
- ○ To optimize simulation speed

What metric is to keep an eye on after all coverage types are 100% * 1 point

- ○ Working hours per day for the engineers
- ○ Bug rate per week
- ○ Customer feedback

If code coverage is high and functional coverage is low, then.... * 1 point

- ○ Need more FC points, including corner cases
- ○ Check Bug Rate
- ○ Test are not Exercising the full design
- ○ There is missing specs and features in the design

What does the fork-join construct do in SystemVerilog? * 1 point

- ○ Executes multiple processes concurrently and waits for all of them to complete
- ○ Executes a single process concurrently with itself
- ○ Starts multiple processes and waits for any one of them to complete
- ○ Starts multiple processes and waits for none of them to complete

What is the primary difference between fork-join and fork-join_any? * 1 point

- ○ fork-join waits for all processes to complete, while fork-join_any waits for any one process to complete
- ○ fork-join_any waits for all processes to complete, while fork-join waits for any one process to complete
- ○ fork-join executes processes sequentially, while fork-join_any executes processes concurrently
- ○ fork-join_any executes processes sequentially, while fork-join executes processes concurrently

When using fork-join_any, what happens if one of the processes completes? * 1 point

- ○ All other processes will be terminated immediately
- ○ The remaining processes continue to run, but the fork-join_any block exits
- ○ All other processes are suspended
- ○ The simulation stops immediately

How does fork-join_none differ from fork-join? * 1 point

- ○ fork-join_none waits for all processes to complete, while fork-join does not
- ○ fork-join_none does not wait for any of the processes to complete
- ○ fork-join_none executes all processes sequentially
- ○ fork-join executes processes without waiting for them to complete

Which construct is best suited for scenarios where you need to perform actions based on the first process that completes? * 1 point

- ○ fork-join
- ○ fork-join_any
- ○ fork-join_none
- ○ Sequential execution

In which scenario would fork-join be most appropriate? * 1 point

- ○ When you need to start several tasks and proceed as soon as any one completes
- ○ When you need to start multiple tasks and ensure that all of them have completed before proceeding
- ○ When you need to start tasks without caring about their completion
- ○ When you need to execute tasks in a specific order

What does the |-> implication operator specify in SystemVerilog assertions? * 1 point

- ○ The sequence on the right side must follow the sequence on the left side, allowing overlap
- ○ The sequence on the right side must occur immediately after the sequence on the left side, without overlap
- ○ The sequence on the right side must occur any time during the execution of the sequence on the left side
- ○ The sequence on the right side must not overlap with the sequence on the left side

In which scenario is the [->] repetition operator used in SystemVerilog? * 1 point

- ○ To specify that a sequence must occur exactly the given number of times consecutively
- ○ To specify that a sequence must occur at least a minimum number of times, but can skip some occurrences
- ○ To define that a sequence must not occur at all
- ○ To indicate that a sequence should occur only once

What is the primary purpose of an interface in SystemVerilog? * 1 point

- ○ To define a set of related signals and variables for use across multiple modules
- ○ To generate random test data for simulation
- ○ To implement a clocking block
- ○ To provide a mechanism for creating constraints

Which of the following is a correct way to use an interface within a module? * 1 point

- ○ Instantiate the interface as a local variable inside the module
- ○ Pass the interface as a port in the module declaration

What is the benefit of using modport in a SystemVerilog interface? * 1 point

- ○ To specify the access mode (input, output, or inout) for the interface signals
- ○ To define the data type of interface signals
- ○ To create multiple versions of the interface with different access permissions
- ○ To enforce a specific clocking behavior

How does the |=> implication operator differ from |-> in SystemVerilog assertions? * 1 point

- ○ |=> allows overlap between sequences, while |-> does not
- ○ |=> specifies a non-overlapping requirement, while |-> allows for overlapping
- ○ |=> specifies that the right-side sequence must occur before or during the left-side sequence, while |-> allows overlap
- ○ |=> is used for bidirectional implications, while |-> is for unidirectional implications

The meaning of cross coverage is * 1 point

- ◯ Uncovered bins affect covered bins so it is called cross
- ◯ Creation of bins interseted in covering more than one coverpoint bins
- ◯ Uncovered bins are labeled with a cross

Editing the weight of coverage bins is useful for * 1 point

- ◯ Giving important bins higher effect on the final coverage results
- ◯ controlling the number of auto generated bins
- ◯ None of the above

You can't control the number of autogenerated bins * 1 point

- ◯ True
- ◯ False

System Verilog Assertions are * 1 point

- ◯ A methodology for verification competing with OVM and UVM
- ◯ A powerful tool acting as a substitution to the golden model in some test cases
- ◯ Another HDL language

Assertions faults always break the simulation and no other alternatives like displaying error or warning messages * 1 point

- True
- False

The difference between |-> and |=> is that the latter checks at the same clock edge while the first checks at the next clock edge * 1 point

- True
- False

The difference between
assert property .. seq_1 ##1 seq_2 and
assert property .. seq_1 |=> seq_2
is that the first one needs this condition to be true at all edges while the second one needs only property2 to be true given property1 is true * 1 point

- True
- False

$rose(a) |=> b[->1] means * 1 point

- when a rises b has to be high in the next clock cycle
- when a rises b has to be high in any time starting from the current clock cycle till the end of simulation
- when a rises b has to be high in any time starting from the next clock cycle till the end of simulation

How does SystemVerilog cross coverage help in verification? * 1 point

- ○ It helps track combinations of multiple coverage points.
- ○ It ensures 100% functional coverage.
- ○ It reduces testbench execution time.
- ○ It replaces assertion-based verification.

What is the purpose of **illegal bins** in cross coverage? * 1 point

- ○ Optimize the number of bins
- ○ Define expected test scenarios
- ○ Mark invalid value combinations
- ○ Force additional test vectors

What is the role of disable iff in assertions? * 1 point

- ○ It ignores specific coverage conditions
- ○ It ensures assertions always execute
- ○ It disables an assertion if a reset condition is met
- ○ It defines the precedence of assertions

What is the primary benefit of using SystemVerilog **interfaces**? * 1 point

- ○ Eliminates the need for clocking blocks
- ○ Increases simulation speed
- ○ Reduces redundant port declarations
- ○ Replaces the need for assertions

How do you connect a **modport** inside a module? * 1 point

- ○ Directly calling the interface without a modport
- ○ Using the interface_name.modport_name syntax
- ○ Using module.interface syntax
- ○ Using assign statements

How is an interface instantiated inside a testbench? * 1 point

- ○ Using a standard module instantiation syntax
- ○ Using bind to link modules
- ○ Using new() like in SystemVerilog classes

What happens if a module does not use modport when accessing an interface? * 1 point

- ○ It will synthesize but not simulate
- ○ It may access signals incorrectly
- ○ It will automatically map to correct signals
- ○ It will only work in RTL simulation

Which of the following is a valid reason to use cross_auto_bin_max = 0? * 1 point

- ○ To reduce simulation memory usage by avoiding automatic bin explosion
- ○ To enforce that only specific cross combinations are tracked
- ○ To improve coverage accuracy by manually defining relevant bins
- ○ All of the above

What is the effect of using the "strong" operator on a sequence? * 1 point

- ○ The sequence will fail if it did not complete before the simulation ends early
- ○ The sequence is checked with higher priority
- ○ The sequence is evaluated using formal verification
- ○ The sequence timing constraints are made stricter

What does this property check? * 1 point

```
property p2;
 @(posedge clk)
 $rose(en) |-> ##[1:3] $fell(en);
endproperty
```

- ○ en must rise and then fall within 1-3 cycles
- ○ Whenever en rises, it must fall within 1-3 cycles
- ○ en must rise, then stay high for 1-3 cycles before falling
- ○ en must rise and fall alternately every 1-3 cycles

What does this assertion ensure? * 1 point

```
property p_throughout;
 @(posedge clk)
 $rose(start) |-> (busy throughout (##[1:10] $fell(done)));
endproperty
```

- ○ When start rises, busy must stay high until done falls (within 1-10 cycles)
- ○ busy must be high for 1-10 cycles after start rises
- ○ done must fall within 1-10 cycles if busy is high
- ○ start and busy must rise together

An assertion bound via bind is failing unexpectedly. What is the **best** way to debug? * 1 point

- ○ Add the assetions to the waveform and debug the red markers for the failed assetions
- ○ Add $error statment printing the signal values in the assertion action block
- ○ All of the above

What is the difference between: *
a |-> ##[1:3] b And a |-> b[->1:3]

1 point

- ◯ The first checks for b in 1-3 cycles, the second checks for 1-3 occurrences of b
- ◯ The first requires consecutive b, the second allows non-consecutive b
- ◯ The second form is invalid SVA syntax
- ◯ Both behave identically

Submit Page 1 of 1 Clear form