Module: SV for Verification
Section: Coverage **Task:** LRM Reference

Task 2

LRM Reference

➢ **Q1: Simulation Control System Tasks (LRM Page # 556):**

SystemVerilog provides simulation control tasks to manage the flow of the simulation, including terminating or pausing it. The three key control tasks are:

1. **$finish:** Terminates the simulation and exits the simulator. It can optionally print diagnostic messages.
2. **$stop:** Suspends the simulation, allowing you to examine the current state of the system. It can also print diagnostic messages.
3. **$exit:** Waits for all program blocks to complete before implicitly calling **$finish**.

Examples:

$finish: Ends the simulation

```
initial begin
  #100;  // Run simulation for 100 time units
  $finish;  // Terminate the simulation
end
```

$stop: Pauses the simulation

```
initial begin
  #50;   // Pause at 50 time units
  $stop; // Wait for user intervention
end
```

$exit: Ends the simulation and passes an exit code

```
initial begin
  #200;
  $exit(0);  // Exit with code 0 (successful)
end
```

➢ **Q2: Conversion functions (LRM Page # 561):**

SystemVerilog provides built-in functions for converting between different data types such as real numbers, integers, and bit representations. These functions are useful for handling mixed data types in simulation.

1. **$rtoi**: Converts a real number to an integer by truncation.
2. **$itor**: Converts an integer to a real number.
3. **$realtobits**: Converts a real number to its bit representation.

Examples:

$rtoi: Real to Integer Conversion

```
real real_num = 3.99;

int int_num;

initial begin

  int_num = $rtoi(real_num);  // Converts 3.99 to integer 3 (truncates)

  $display("Integer value: %0d", int_num);

end
```

$itor: Integer to Real Conversion

```
int int_num = 5;

real real_num;

initial begin

  real_num = $itor(int_num);  // Converts integer 5 to real 5.0

  $display("Real value: %f", real_num);

end
```

$realtobits: Real to Bit Representation Conversion

```
real real_num = 3.14;

int bit_rep;

initial begin

  bit_rep = $realtobits(real_num);  // Converts real 3.14 to its bit
representation

  $display("Bit representation: %h", bit_rep);

end
```

➢ **Q3: Bit vector functions (LRM Page # 568):**

SystemVerilog provides a variety of functions to manipulate and examine bit vectors. These functions are useful for analyzing bit patterns, detecting unknown states, and checking specific conditions within a vector.

1. **$countbits**: Counts the number of bits that are set to 1 in a bit vector.
2. **$isunknown**: Returns 1 if any bit in the vector contains an unknown (x or z) value, otherwise returns 0.
3. **$onehot**: Returns 1 if exactly one bit in the vector is set to 1, and 0 otherwise.

Examples:

$countbits: Counting the number of 1s in a bit vector

```
bit [7:0] data = 8'b10101011;

int count;

initial begin

  count = $countbits(data, ‘1);  // Count the number of 1's in the bit vector

  $display("Number of 1's: %0d", count);  // Output: 5

end
```

$isunknown: Checking if a bit vector contains unknown (x or z) values

```
bit [3:0] signal = 4'b01xz;

initial begin

  if ($isunknown(signal))  // Returns 1 if any bit is 'x' or 'z'

    $display("Signal contains unknown values");

  else

    $display("Signal is known");

end
```

$onehot: Verifying if a bit vector is one-hot (exactly one bit is 1)

```
bit [3:0] vector = 4'b0100;

initial begin

 if ($onehot(vector))  // Returns 1 if exactly one bit is set to 1

   $display("Vector is one-hot");

 else

   $display("Vector is not one-hot");

end
```

➢ **Q4: Severity System Tasks (LRM Page # 569):**

SystemVerilog provides severity system tasks to indicate the severity of issues during simulation. These tasks can report errors, warnings, or information messages, and optionally terminate the simulation.

1. **$fatal:** Terminates the simulation and reports a fatal error.
2. **$error:** Reports an error without terminating the simulation.
3. **$warning:** Reports a warning that indicates a potential issue but doesn't terminate the simulation.
4. **$info:** Displays an informational message during simulation.

Examples:

$fatal: Reporting a fatal error and terminating the simulation

```
initial begin
  if (some_critical_error) begin
    $fatal(1, "Fatal Error: Critical failure occurred!");
    // Simulation will terminate here with exit code 1
  end
end
```

$error: Reporting an error without stopping the simulation systemverilog

```
initial begin
  if (invalid_data) begin
    $error("Error: Invalid data detected!");
    // Simulation continues after logging the error
  end
end
```

$warning: Reporting a warning for a non-critical issue
systemverilog

```
initial begin
  if (unexpected_condition) begin
    $warning("Warning: Unexpected condition encountered.");
    // Simulation continues with the warning message
  end
end
```

$info: Displaying an informational message
Systemverilog

```
initial begin
  $info("Info: Simulation started.");
  // Logs an informational message without affecting simulation

end
```

➢ **Q5: Sampled value functions (LRM Page # 569):**

SystemVerilog provides sampled value functions to monitor changes or stability in signals across clock cycles. These functions are useful for detecting transitions, ensuring signal stability, and verifying specific timing behaviors in simulations.

1. **$stable:** Returns 1 if a signal has not changed during the current clock cycle.
2. **$rose:** Returns 1 if the signal transitioned from 0 to 1 in the current clock cycle.
3. **$fell:** Returns 1 if the signal transitioned from 1 to 0 in the current clock cycle.
4. **$changed:** Returns 1 if the signal changed state (either from 0 to 1 or 1 to 0) in the current clock cycle.

Examples:

$stable: Detecting if a signal has remained stable

```
always @(posedge clk) begin
  if ($stable(signal_in)) begin
    $display("Signal is stable at time %0t", $time);
  end
end
```

$rose: Detecting a rising edge (transition from 0 to 1)

```
always @(posedge clk) begin
  if ($rose(signal_in)) begin
    $display("Signal rose from 0 to 1 at time %0t", $time);
  end
end
```

$fell: Detecting a falling edge (transition from 1 to 0)

```
always @(posedge clk) begin
  if ($fell(signal_in)) begin
    $display("Signal fell from 1 to 0 at time %0t", $time);
  end
end
```

$changed: Detecting any change in the signal

```
always @(posedge clk) begin
  if ($changed(signal_in)) begin
    $display("Signal changed at time %0t", $time);
  end
end
```

➢ **Q6: Assertion control tasks (LRM Page # 571):**

SystemVerilog provides several system tasks to control the behavior of assertions during simulation. These tasks enable or disable assertions dynamically, which can be useful for controlling simulation flow and debugging.

1. **$asserton:** Enables all assertions.
2. **$assertoff:** Disables all assertions.
3. **$assertkill:** Disables all currently active assertions.
4. **$assertpasson:** Enables notifications for passing assertions.
5. **$assertpassoff:** Disables notifications for passing assertions.
6. **$assertcontrol:** Provides fine control over specific assertions by controlling their severity.

Examples:

$asserton: Enabling assertions during simulation

```
initial begin
```

```
  // Disable assertions at the start of simulation
  $assertoff;

  // After some time, enable all assertions
  #100 $asserton;
end
```

$assertoff: Disabling all assertions

```
initial begin
  // Disable assertions at the start of the simulation
  $assertoff;

  // Perform operations without assertion checks
  #50;

  // Assertions remain disabled until explicitly enabled again
end
```

$assertkill: Disabling only currently active assertions

```
initial begin
 // Assertions are running
 #50;

 // Disable any active assertions but allow new ones to start
 $assertkill;
end
```

$assertpasson: Enabling passing assertion notifications

```
initial begin
 // Enable notifications for passing assertions
 $assertpasson;
end
```

$assertcontrol: Controlling specific assertions with a certain severity

```
initial begin
  // Control assertion severity, enabling them as warnings
  $assertcontrol(2);  // Treat all assertions as warnings
end
```