INTERVIEW QUESTIONS

VERILOG

PART 1

JAIRAJ MIRASHI
DESIGN VERIFICATION ENGINEER

1. **Identify the error in the following code:**

b[7:0] = {2{5}};

The error in the code is the incorrect usage of the replication operator. The correct usage should be {n{value}} where n is the number of times the value should be replicated. In the given code, it should be:

b[7:0] = {2{5'b1}};

This would assign the value 8'b00100001 to the b signal.

Example:

```
module ReplicationExample;

   // Declare an 8-bit reg signal named 'b'
   reg [7:0] b;

   // Initial block for simulation initialization
   initial begin
      // Using the replication operator {n{value}} to replicate 5'b1 two
times
      // Resulting in the value 8'b11, which is assigned to signal 'b'
      b[7:0] = {2{5'b1}};

      // Display the value of signal 'b' using binary format
      $display("b = %b", b);
   end

endmodule
```

2. **When are instance names optional?**

In Verilog, instance names are optional when instantiating modules using the module's positional port connections. This means that if you instantiate a module and connect its ports based on their position in the module's port list, you can omit the instance names.

For example, consider a module named myModule with the following port list:

```
module myModule (
   input a,
   input b,
   output c
);
```

If you instantiate this module and connect ports based on their position, you can omit the instance names like this:

```
myModule u1 (.a(input_a), .b(input_b), .c(output_c));
```

Here, the instance names a, b, and c are omitted, and only the port connections are provided.

"In Verilog, instance names are optional when instantiating modules using positional port connections. This means that if you connect the module ports in the instantiation based on their position in the module's port list, you can omit specifying the instance names in the instantiation statement. This can be helpful in simplifying the instantiation code, especially when the port names are self-explanatory. However, using named connections is generally recommended for clarity and maintainability."

3. In the following program, what is the problem and how to avoid it?

```
task idriver;
input iread;
input i[7:0] write_data;
begin
  #30 idate_valid = 1'b1;
  wait(read == 1'b1);
  #20 icpu_data = write_data;
  $display("End of task");
end
endtask
```

The given program has a couple of issues:

Undeclared Variables: In the program, the variables idate_valid and icpu_data are being used without being declared. Verilog requires variable declarations before usage.

Non-blocking Assignment in a Task: The line #20 icpu_data = write_data; uses a non-blocking assignment (<=) inside a task. Non-blocking assignments are typically used in procedural blocks (always blocks) to model sequential behavior. Using them in a task can lead to unexpected behavior.

Here's a corrected version of the program:

```
module TestModule;
   reg idate_valid;
   reg iread;
   reg [7:0] icpu_data;

   task idriver;
      input iread;
      input [7:0] write_data;
   begin
      #30 idate_valid = 1'b1;
      wait(iread == 1'b1);
      #20;
      icpu_data <= write_data; // Using non-blocking assignment here
      $display("End of task");
   end
   endtask

   initial begin
      // Instantiate the task and provide input values
      idriver(iread, icpu_data);
   end
endmodule
```

In this corrected version:

idate_valid and icpu_data are declared as reg variables.

The non-blocking assignment operator (<=) is used for assigning icpu_data to write_data inside the task, as non-blocking assignments are suitable for modeling sequential behavior.

The idriver task is instantiated in the initial block with the appropriate inputs.

Please note that the instantiation of the task with actual values for iread and icpu_data is just a demonstration for testing purposes. In a real scenario, these values would come from your design or testbench environment.

4. How many levels can be nested using include?

The include directive in Verilog allows us to incorporate external files into our code, promoting modularity and organization within our design. In terms of nesting, it's worth noting that the include compiler directive can be utilized to nest files within files. Specifically, it's possible to nest the include directive to a depth of at least 16 levels. This implies that we can include files within other included files, forming a hierarchy of up to 16 layers. However, it's important to exercise caution when using this approach excessively, as excessive nesting can potentially complicate code maintenance and comprehension.

5. What is the use of $countdrivers?

The $countdrivers system function is used to determine the number of drivers connected to a net. It can be helpful in identifying potential issues with multiple drivers on the same net, which can lead to contention and unexpected behavior in the design.

6. What is the use of $getpattern?

The $getpattern system function is used to retrieve stimulus patterns from a pre-loaded memory, often used for testbenches to propagate patterns to multiple scalar inputs efficiently.

Here's a small code snippet demonstrating the use of $getpattern:

```
module GetPatternExample;
   reg [7:0] i;
   reg [7:0] in_mem [0:15];
   integer index;

   initial begin
      // Load stimulus patterns into 'in_mem' memory
      in_mem[0] = 8'b10101010;
      in_mem[1] = 8'b11001100;
      // ... Other pattern assignments ...

      index = 0; // Set initial pattern index

      // Use $getpattern to retrieve patterns from 'in_mem'
      i = $getpattern(in_mem[index]);

      // Display the retrieved pattern
      $display("Retrieved pattern: %b", i);
   end
endmodule
```

7. What is the functionality of &&& (not && or &)?

In Verilog, there is no &&& operator. It appears there might have been a misunderstanding or confusion. The correct operators for logical AND are && (logical AND) and & (bitwise AND). The && operator is used for logical AND operations, evaluating to true if both operands are true. The & operator performs a bitwise AND operation between corresponding bits of the operands.

8. How to get a copy of all the text that is printed to the standard output in a log file?

You can redirect the standard output to a log file using the $display system function.

By using the $display system function with file I/O syntax, you can direct the output to both the console and a log file.

Code:

```
initial begin
   $fopen("log.txt");
   $display("This will be written to the log file");
   $fclose;
end
```

9. What is the use of PATHPULSE$?

PATHPULSE$ is used to control pulse handling within a module path.

10. In the statement (a==b) && (c==d), what is the expression coverage if a=0, b=0, c=0, d=0?

The expression (a==b) && (c==d) evaluates to 1'b1 (True).

Since both (a==b) and (c==d) are true (both are equal to 0), the entire expression is true.

11. Difference between Reduction and Bitwise operators?

Reduction operators operate on bits of a single vector operand, while bitwise operators operate on corresponding bits of two operands.

Reduction operators (e.g., &, |, ^) reduce a vector into a single bit result. Bitwise operators (e.g., &, |, ^) perform operations between corresponding bits of two vectors.

Code:

```
wire [3:0] a, b;
wire r_and, bw_and;
assign r_and = &a;        // Reduction AND
assign bw_and = a & b;    // Bitwise AND
```

12. What is the difference between the following two lines of Verilog code?

#5 a = b;

a = #5 b;

The first line delays the assignment by 5 time units, while the second line delays the assignment until 5 time units after b changes.

In the first line, the assignment is delayed by 5 time units from the point of execution. In the second line, the assignment is delayed until 5 time units after b changes.

13. What is the difference between c = (foo) ? a : b; and if (foo) c = a; else c = b;?

Both lines produce the same result, but the first line uses the ternary operator for conditional assignment.

The ternary operator (condition) ? value_if_true : value_if_false is a compact way to perform conditional assignment, similar to an if-else statement.

Both expressions achieve conditional assignment of values to c, but they differ in how they handle "x" and "z" values.

In (foo) ? a : b;, the ternary operator maintains "x" and "z" values, resulting in "x" or "z" if foo is "x" or "z".

In the if-else construct, if foo is "x" or "z", it treats the condition as false and assigns the value of b to c.

Code:

```
reg [7:0] a, b, c;
reg foo;

// Using ternary operator
assign c = (foo) ? a : b;

// Using if-else
always @* begin
   if (foo)
      c = a;
   else
      c = b;
end
```

14. How is Verilog implementation independent and why is this an advantage?

Verilog is implementation independent as it describes behavior, not specific hardware. This portability allows designs to be simulated on different tools and technologies.

15. What level of Verilog is used in: a. Testbenches b. Synthesized designs c. Netlists

a. Testbenches use behavioral and structural Verilog.

b. Synthesized designs use structural and gate-level Verilog.

c. Netlists use gate-level Verilog.

16. What is the difference between $fopen("filename"); and $fopen("filename","w");?

The first opens the file in read mode, while the second opens the file in write mode.

$fopen("filename"); opens the file in read mode, allowing you to read from it. $fopen("filename", "w"); opens the file in write mode, allowing you to write to it.

17. What is the difference between multi-channel descriptors (mcd) and file descriptors (fd)?

mcd is a 32-bit reg indicating opened files. fd is a file descriptor returned by $fopen.

mcd is a single-bit reg indicating opened files, while fd is a descriptor that refers to a specific opened file.

18. How to generate a random number?

Use $random to generate a 32-bit pseudo-random number.

Code:

```
module RandomNumberExample;
   reg [31:0] random_number;

   initial begin
      random_number = $random;
     $display("Random Number: %d", random_number);
   end
endmodule
```

19. How to generate a random number which is less than 100?

Use $random % 100.

Code:

```
module RandomNumberExample;
   reg [31:0] random_number;

   initial begin
      random_number = $random % 100;
     $display("Random Number: %d", random_number);
   end
endmodule
```

20. How to generate a random number which is between 40 and 50?

Use MIN + {$random} % (MAX - MIN) will_generate random numbers between MIN and MAX

Code:

```
module Tb();
   integer add;

   initial
   begin
      repeat(5) // Repeat the following block 5 times
      begin
         #1; // Wait for 1 time unit before proceeding

         // Generate a random number between 40 and 49
         add = 40 + {$random} % (50 - 40);

         // Display the generated random number
         $display("add = %0d", add);
      end
   end
endmodule
```