DAY 55 -111 DAYS VERIFICATION CHALLENGE

Topic: Verilog trick questions

Skill: Verilog, RTL Design

DAY 55 CHALLENGE:

1. How can I abort execution of a task or a block of code?

We can abort the execution of a task or block using the disable statement. For example, within a block or a task, you can use disable block_name; to exit the execution of that block immediately.

```
initial begin : block_name
    // Some code here
    if (condition) disable block_name;
    // More code that won't execute if disabled
end
```

2. What does it mean to "short-circuit" the evaluation of an expression?

Short-circuit evaluation means that in a logical expression, evaluation stops as soon as the outcome is determined. For example, in the expression A && B, if A is false, B is not evaluated because the whole expression is already determined to be false. Similarly, in A || B, if A is true, B is not evaluated because the whole expression is true.

3. What is the difference between a constant part-select and an indexed part-select of a vectored net?

Constant Part-Select: A part-select where the bit positions are specified as constants. Example: net[15:8].

Indexed Part-Select: This is used when the part-select starts at a variable position, and the width is specified. Example: net[start_index +: width] or net[start_index -: width].

4. ilustrate how memory indirection is achieved in Verilog.

Memory indirection can be achieved by using pointers in higher-level languages, but in Verilog, memory indirection is simulated by indexing memory arrays with variables.

```
reg [7:0] memory [0:255]; // 256 bytes of memory
reg [7:0] index;
```

```
reg [7:0] data;
initial begin
    index = 5;
    data = memory[index]; // Indirect access
end
```

5. What is the logic synthesized when a non-constant is used as an index in a bit-select?

When a non-constant index is used for a bit-select, the synthesis tool generates a multiplexer (MUX). The inputs of the MUX correspond to each bit in the vector, and the select lines of the MUX are driven by the index.

6. How are string operands stored as constant numbers in a reg variable?

In Verilog, string literals are stored in reg variables by converting each character to its ASCII value and concatenating them.

```
reg [31:0] my_string = "ABCD"; // Stored as 32'h41424344
```

7. How can I typecast an expression to control its sign?

You can control the sign of an expression using the $signed or $unsigned functions:

```
reg signed [7:0] a = 8'hFF;
reg [7:0] b;
b = $unsigned(a); // Typecasts 'a' as unsigned
```

8. What are the pros and cons of using hierarchical names to refer to Verilog objects?

Pros of Using Hierarchical Names:

1. **Direct Access to Signals:**
    - Hierarchical names allow you to directly access signals, registers, or other objects deep within the design hierarchy from higher levels. This is particularly useful for debugging, monitoring, or applying specific overrides during simulation.
    - Example: Accessing a signal deep within a module: **top_module.sub_module.signal_name.**
2. **Flexibility in Testing:**
    - During verification, hierarchical names can be used to apply specific test conditions or to observe internal states without modifying the original design. This is handy for writing testbenches where certain internal signals need to be monitored or controlled.

3. **Ease of Debugging:**
    - When debugging complex designs, hierarchical names provide a straightforward way to trace signals and identify where issues might be occurring within the design hierarchy.
4. **Control over Internal Signals:**
    - It allows you to control or inject values into internal signals without exposing them at the module's interface. This can be useful for special cases during simulation.

Cons of Using Hierarchical Names:

1. **Reduced Modularity and Encapsulation:**
    - Relying on hierarchical names breaks the modularity of the design. It creates dependencies on the internal structure of modules, making it harder to change the design hierarchy or reuse modules in different contexts.
    - Example: If the internal structure of sub_module changes, any code referencing top_module.sub_module.signal_name will break.
2. **Maintenance Challenges:**
    - As the design evolves, maintaining code that uses hierarchical names can be cumbersome. If the module hierarchy or names change, all instances where these names are referenced need to be updated.
3. **Design Abstraction Violation:**
    - Using hierarchical names can violate the abstraction of the design by exposing internal details that should ideally remain hidden. This goes against the principles of good design practice, where modules should interact only through well-defined interfaces.
4. **Limited Synthesis Support:**
    - Hierarchical names are primarily useful during simulation. Most synthesis tools do not support the use of hierarchical names because the design hierarchy is often flattened during synthesis. This means that hierarchical references might not be portable across different tools or flows.
5. **Potential for Confusion:**
    - Using hierarchical names can lead to confusion, especially in large designs with deep hierarchies. It might not always be clear where a particular signal is coming from or how it fits into the overall design.

```
module sub_module;
    reg [7:0] internal_signal;
```

```
    initial begin
        internal_signal = 8'hA5; // Initialize the internal signal
    end
endmodule
module top_module;
    sub_module u_sub_module(); // Instantiate the sub_module
endmodule
module testbench;
    reg [7:0] observed_signal;
    initial begin
        // Accessing internal_signal using hierarchical name
        observed_signal = top_module.u_sub_module.internal_signal;
        #10; // Wait for simulation time
        // Display the observed signal value
        $display("Observed signal value: %h", observed_signal);
    end
endmodule
```

9. Explain fork-join in Verilog with an example.

The fork-join construct in Verilog allows you to execute multiple statements or tasks in parallel within a procedural block. Once all the parallel branches have completed, the control moves to the statements after the join.

How fork-join Works

- fork: Marks the beginning of parallel execution.
- join: Marks the end of parallel execution. The code execution will wait here until all the parallel branches started by the fork are finished.

```
module fork_join_example;
    reg [7:0] a, b, c;
    initial begin
        a = 0;
        b = 0;
        c = 0;
        // Start parallel execution using fork-join
        fork
            // First parallel block
            begin
                #5 a = 8'hAA; // After 5 time units, set a to 0xAA
```

```
            $display("Time: %0t | a = %h", $time, a);
        end
        // Second parallel block
        begin
            #10 b = 8'hBB; // After 10 time units, set b to 0xBB
            $display("Time: %0t | b = %h", $time, b);
        end
        // Third parallel block
        begin
            #15 c = 8'hCC; // After 15 time units, set c to 0xCC
            $display("Time: %0t | c = %h", $time, c);
        end
    join
    // This code runs only after all forked blocks are done
    $display("Time: %0t | All parallel blocks finished", $time);
end
endmodule
```

10. Can I return from a function without having it disabled?

Yes, We can return from a function in Verilog without disabling it. In Verilog, functions are used to compute and return a value. Unlike tasks, functions in Verilog cannot be disabled; they always return a value when they complete execution.

- A function in Verilog performs a specific computation and returns a value.
- You can use the return statement to exit the function and return a value explicitly.
- Functions do not have any "enable" or "disable" control; they simply execute when called and return a result.

```
module function_example;
    reg [7:0] result;
    // Function definition
    function [7:0] add;
        input [7:0] a, b;
        begin
            add = a + b;  // Perform addition
            return;       // Explicit return (optional in this case)
        end
    endfunction
```

```
    initial begin
        result = add(8'hA5, 8'h3C); // Call the function
        $display("Result of add: %h", result); // Display the result
    end
endmodule
```

11. What is strobing? How do I selectively strobe a net?

Strobing is capturing a signal value at a specific point in simulation time, often used for verification. You can selectively strobe using $strobe:

```
$strobe("Value of signal: %0d", signal);
```

12. How can I selectively enable or disable monitoring?

In Verilog, monitoring refers to the use of the `$monitor` system task to continuously display the values of variables or signals whenever they change. If you want to selectively enable or disable monitoring, you can do so by controlling when the `$monitor` task is called and when it is disabled using the `$monitoroff` and `$monitoron` system tasks.

```
module monitor_example;
    reg clk;
    reg [7:0] data;
    // Generate a clock signal
    initial begin
        clk = 0;
        forever #5 clk = ~clk;  // Clock with a period of 10 time
units
    end
    // Stimulus block
    initial begin
        data = 8'h00;
        #10 data = 8'hAA; // Change data after 10 time units
        #10 data = 8'h55; // Change data after another 10 time units
        #10 data = 8'hFF; // Change data after another 10 time units
        #10 data = 8'h00; // Change data after another 10 time units
    end

    // Monitor changes in clk and data
    initial begin
        $monitor("Time: %0t | clk = %b, data = %h", $time, clk, data);
        #15 $monitoroff; // Disable monitoring after 15 time units
```

```
        #15 $monitoron;  // Re-enable monitoring after 30 time units
    end
endmodule
```

13. How can I specify arguments on the Verilog simulator's command line?

Arguments can be passed using +define+ options or by using the -g flag for variables:

```
vlog +define+MY_DEFINE=1  (in shell)
```

In Verilog, access them with:

```
`ifdef MY_DEFINE
    // Code here
`endif
```

14. Can the 'define be used for text substitution through variable instead of literal substitution only?

In Verilog, the \definedirective is used for macro definition and text substitution. However,`define` performs literal substitution and doesn't directly support dynamic substitution through variables at runtime. This means that the substitution happens at the pre-processing stage, before simulation, and the substitution text is fixed.

```
`define WIDTH 8
module define_example;
    reg [`WIDTH-1:0] data; // `WIDTH will be replaced by 8 during preprocessing
    initial begin
        data = 8'hFF;
        $display("Data: %h", data);
    end
endmodule
```