DAY 48 -111 DAYS VERIFICATION CHALLENGE

Topic: RTL Design considerations

Skill: Verilog, RTL Design

DAY 48 CHALLENGE:

1. What are some reusable coding practices for RTL Design?

- Modular Design: Break down large designs into smaller, manageable modules. Each module should have a well-defined interface.
- Parameterization: Use parameters for configurable widths and other characteristics to make your modules more flexible.
- Naming Conventions: Use consistent and descriptive naming conventions for signals, modules, and variables to make the code more readable and maintainable.
- Documenting Code: Include comments to explain complex logic and assumptions.
- Testing: Write testbenches and verify each module independently before integrating it into the larger design.

2. What are "snake" paths, and why should they be avoided?

- Definition: Snake paths refer to long, convoluted paths in combinational logic that can cause timing issues or make the design harder to understand.
- Avoidance: Keep paths as short and direct as possible to simplify timing analysis and ensure reliable operation. Use pipelining to break long combinational paths into smaller, manageable stages.

3. What are a few considerations while partitioning large designs?

- Clock Domains: Ensure that modules operating on different clock domains are correctly managed to avoid synchronization issues.
- Communication: Design clear and efficient interfaces between modules.
- Complexity: Balance the complexity of each partition to ensure no single module becomes a bottleneck.
- Testing: Verify partitions individually and ensure that integration does not introduce new issues.

4. How can I reliably convey control information across clock domains?

- Synchronizers: Use multi-stage flip-flop synchronizers to mitigate metastability issues.
- Handshake Protocols: Implement handshaking or FIFO-based protocols to safely transfer data and control signals between different clock domains.

5. What is a safe strategy to transfer data of different bus-widths and across different clock domains?

- Width Conversion: Use appropriate width conversion logic, such as padding or truncating data, to align with bus-width requirements.
- Clock Domain Crossing: Employ FIFO buffers or synchronizers to manage data transfer between different clock domains safely.

6. What are a few considerations while using FIFOs for posted writes or prefetched reads that influence the speed of the design?

- Depth: Ensure FIFO depth is adequate to handle data bursts and avoid overflows or underflows.
- Latency: Account for FIFO read and write latency in your design timing.
- Throughput: Optimize read and write pointers and manage data access to maximize throughput.

7. What will be synthesized from a module with only inputs and no outputs?

- Result: A module with only inputs and no outputs will typically be synthesized as a black-box or potentially optimized away, as it does not provide any outputs or functional behavior.

8. What are "combinatorial timing loops"? Why should they be avoided?

- Definition: Combinatorial timing loops occur when there are feedback paths within a combinational logic block, creating circular dependencies.
- Avoidance: These should be avoided because they can lead to unpredictable behavior and timing issues. Ensure that all feedback paths are appropriately managed by using sequential elements where necessary.

9. How does the sensitivity list of a combinatorial always block affect pre- and post- synthesis simulation? Is this still an issue lately?

- Effect: The sensitivity list determines which signals trigger the block's execution. An incorrect sensitivity list can lead to simulation mismatches between pre- and post-synthesis results.
- Current Practice: In modern tools, using the * (asterisk) in the sensitivity list is generally avoided. Instead, explicitly list all signals that the block depends on to ensure accurate synthesis and simulation.

10. Design an 8-bit counter by using forever loop. The counter starts counting at count=5 & finishes at count=67. The count is incremented at positive edge of clock. The clock has a time period of 10. The counter counts through the loop only once & then is disabled. (Hint: Use the disable statement).

```
module counter_8bit (
    input wire clk,      // Clock signal
    input wire reset,    // Asynchronous reset
    output reg [7:0] count // 8-bit counter output
);
    initial begin
        count = 8'd5;    // Initialize count to 5
    end
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            count <= 8'd5;  // Reset count to 5
        end else begin
            forever begin
                @(posedge clk);
                if (count < 8'd67) begin
                    count <= count + 1;  // Increment count
                end else begin
                    disable count_loop;   // Disable the loop when
count reaches 67
                end
            end
        end
    end
endmodule
```