1. What do conditional assignments get inferred into?
   Conditional assignments in Verilog (using ?:) typically infer multiplexers. The synthesized hardware will be a multiplexer that selects between different inputs based on the condition provided.
2. What is the logic that gets synthesised when conditional operators in a single continuous assignment are nested?
   When conditional operators are nested within a single continuous assignment, the synthesis tool will create a combinational logic network. This results in a structure similar to a multiplexer tree, where the conditions are evaluated hierarchically.
3. Why should a nonclocking assignment be used for sequential logic, and what would happen if a blocking assignment were used? Compare it with the same code in a combinational block.
   - Non-Blocking Assignments (<=): Used in sequential logic to allow for the evaluation of all assignments in a single simulation time step, ensuring that all updates occur simultaneously.
   - Blocking Assignments (=): If used in sequential logic, it can lead to race conditions or unintended behaviour because it evaluates and updates immediately, which can affect subsequent assignments in the same block.
   - Comparison with Combinational Block: In combinational logic, blocking assignments are suitable as they ensure that changes propagate immediately through the logic without timing issues.
4. What does the logic in a function get synthesised into? What are the area and timing implications of the calling function in RTL?
   - Synthesis: Logic within a function synthesizes into combinational logic. Functions typically return a value based on their inputs, resembling a combinational logic gate.
   - Area and Timing Implications: Calling a function in RTL can increase area and timing complexity because it may require additional logic to handle the function calls, impacting overall performance.
5. What does the logic in task get synthesised into? Explain with an example.
   Synthesis: Tasks do not synthesize into hardware in the same way as functions. Instead, they are more about procedural behaviour and control flow. If a task contains synthesizable logic, it can only be executed in simulation, not synthesized into hardware.
   Example: task my_task;
   ```
    input [3:0] a;
    output [3:0] b;
    begin
      b = a + 1; // This logic will not be synthesized directly
    end
   endtask
   ```
6. What are the differences between using a task, and defining a module for implementing reusable logic?
   Tasks:

   - Designed for procedural reuse without generating new hardware.
   - Can include timing controls and are useful for encapsulating complex behavior.

   Modules:
   - Generate new hardware instances.
   - Suitable for reusable logic that can be instantiated multiple times.
7. Can tasks and functions be declared external to the scope of module-endmodule?
   Both tasks and functions can be declared outside the scope of a module, typically in packages or libraries, but they must still be properly referenced within the module.
8. Design a 4-bit up-down counter which has input up-down. When up_down=1 it acts as an up counter, when up_down=0 it acts as a down counter.

```
module up_down_counter (
    input wire clk,
    input wire reset,
    input wire up_down,
    output reg [3:0] count
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            count <= 4'b0000;
        else if (up_down)
            count <= count + 1; // Up counting
        else
            count <= count - 1; // Down counting
    end
endmodule
```

9. Design a binary to Gray code converter in Verilog.

```
module binary_to_gray (
    input wire [3:0] binary,
    output wire [3:0] gray
);
    assign gray[3] = binary[3];          // MSB remains the same
    assign gray[2] = binary[3] ^ binary[2];
    assign gray[1] = binary[2] ^ binary[1];
    assign gray[0] = binary[1] ^ binary[0];
endmodule
```