Up/Down Counter Design with Verilog

Overview

This project involves designing and implementing a versatile up/down counter using Verilog HDL. The counter supports counting both upwards and downwards based on control signals, with additional features like enable, reset, and load functionalities.

Features

- **Up/Down Counting**: The counter can count both up and down depending on the up_down control signal.
- **Enable Function**: Allows the counter to operate only when enabled.
- **Reset Function**: Resets the counter to zero when activated.
- **Load Function**: Loads a specific value into the counter when required.

Files

- up_down_counter.v: Verilog module for the up/down counter design.
- tb_up_down_counter.v: Testbench file to simulate and validate the functionality of the counter module.

How to Use

Simulation with ModelSim

1. **Open ModelSim** and create a new project.
2. **Add** up_down_counter.v and tb_up_down_counter.v to your project.
3. **Compile** both the Verilog module and testbench.
4. **Run Simulation** to observe the counter's behavior with different input signals.

Synthesis with Quartus Prime

1. **Open Quartus Prime** and create a new project.
2. **Add** up_down_counter.v as the top-level module.
3. **Assign Pins** for inputs and outputs:
    - clk for the clock input.
    - reset, enable, up_down, load, and load_val for control signals.
    - count for the counter output.
4. **Compile** the design and simulate using ModelSim-Altera within Quartus Prime.

Key Skills Demonstrated

- Verilog HDL
- Digital Design
- Sequential Logic

- Simulation and Synthesis

VERILOG CODE->

- **Up_down_counter.v**

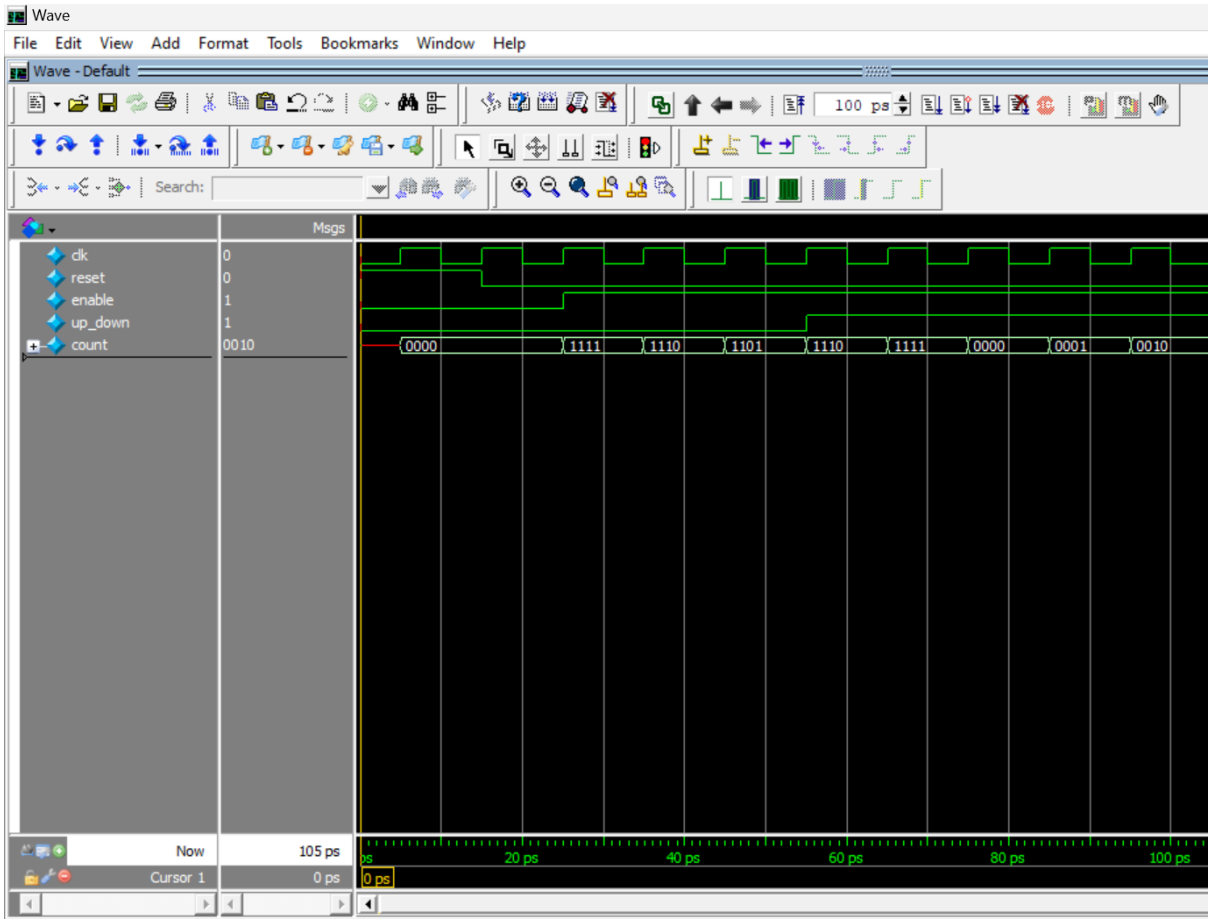
```
module up_down_counter (
input clk,          // Clock input
input reset,        // Synchronous reset
input enable,       // Enable for counting
input up_down,      // Up/Down control (1 = Up, 0 = Down)
input [3:0] load_val, // Value to load into the counter
input load,         // Load signal to load a specific value
output reg [3:0] count // 4-bit counter output
);

always @(posedge clk) begin
  if (reset) begin
    count <= 4'b0000;   // Reset counter to 0
  end else if (load) begin
    count <= load_val;  // Load specific value
  end else if (enable) begin
    if (up_down) begin
      count <= count + 1; // Count up
    end else begin
      count <= count - 1; // Count down
    end
  end
end

endmodule
```

Up_down_counter_tb.v->

```
module up_down_counter_tb();
   reg clk, reset, enable, up_down;
   wire [3:0] count;

   // Instantiate the counter
   up_down_counter dut (
      .clk(clk),
      .reset(reset),
      .enable(enable),
      .up_down(up_down),
      .count(count)
   );

   // Clock generation
   initial begin
      clk = 0;
      forever #5 clk = ~clk; // 10ns clock period
   end

   // Test sequence
   initial begin
      // Initialize signals
      reset = 1; enable = 0; up_down = 0;
      #15 reset = 0; // Deassert reset after 15ns
      #10 enable = 1; // Enable counting after 10ns
      #30 up_down = 1; // Switch to down counting
      #50 $stop; // End the simulation
   end
endmodule
```

RESULTS

MODELSIM SIMULATION OUTPUT->



QUARTUS PRIME SYNTHESIS OUTPUT->