Nang Thiri Wutyi
20113
11/5/2024

Verilog Lab 5

1. **DFF Module**

```
`timescale 1ns/100ps

module dff(input D_i, clock_i, output reg Q_o);
   always @(posedge clock_i)
      Q_o <= D_i;
endmodule
```

2. **Latch with XOR logic**

```
`timescale 1ns/100ps

module xor3 (input a_i, b_i, c_i, clock_i, output reg Q_o);
   always @(posedge clock_i) begin
      if (a_i)
         Q_o <= c_i;
      else
         Q_o <= b_i;
   end
endmodule
```

3. **Original Counter Module**

```
module counter (input clock, input [3:0] in, input latch, input dec, output zero);
   reg [3:0] value;

   always @(posedge clock) begin
      if (latch)
         value <= in;
      else if (dec && !zero)
         value <= value - 1'b1;
   end

   assign zero = (value == 4'b0000);
endmodule
```

Modified Counter Module

```
// counter_modified.sv
```

```
module counter_modified (
    input clock,
    input [3:0] in,
    input latch,
    input dec,
    output zero
);
    reg [3:0] value;

    always @(posedge clock) begin
        if (latch)
            value <= in;                // Load initial value when latch is high
        else if (dec && !zero)
            value <= value - 2'b10;     // Decrement by 2 if dec is high and zero flag is not set
    end

    assign zero = (value == 4'b0000 || value == 4'b0001); // Set zero flag high at 0000 or
0001
endmodule
```

Modified Testbench

```
// counter_modified_tb.sv
`timescale 1ns/100ps

module counter_modified_tb;
    reg clock;
    reg latch;
    reg dec;
    reg [3:0] in;
    wire zero;

    // Instantiate the modified counter
    counter_modified uut (
        .clock(clock),
        .in(in),
        .latch(latch),
        .dec(dec),
        .zero(zero)
    );
```

```
    // Generate clock signal with a period of 10 time units
    initial begin
        clock = 0;
        forever #5 clock = ~clock;
    end

    // Test sequence
    initial begin
        // Initialize inputs
        in = 4'b0101;  // Starting value
        latch = 1;     // Load initial value
        dec = 0;       // Decrement disabled initially
        #10;

        // Start decrementing by setting latch to 0 and dec to 1
        latch = 0;
        dec = 1;

        // Monitor the value and zero flag for a few cycles
        repeat (10) begin
            #10; // Wait for one clock cycle
            $display("Time: %0t | Value: %b | Zero Flag: %b", $time, uut.value, zero);
        end

        $stop; // Stop simulation
    end
endmodule
```

Results

```
2024-11-06 05:24:25 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
warning: Some design elements have no explicit time unit and/or
       : time precision. This may cause confusing timing results.
       : Affected design elements are:
       :   -- module counter_modified declared here: design.sv:2
Time: 200 | Value: 0011 | Zero Flag: 0
Time: 300 | Value: 0001 | Zero Flag: 1
Time: 400 | Value: 0001 | Zero Flag: 1
Time: 500 | Value: 0001 | Zero Flag: 1
Time: 600 | Value: 0001 | Zero Flag: 1
Time: 700 | Value: 0001 | Zero Flag: 1
Time: 800 | Value: 0001 | Zero Flag: 1
Time: 900 | Value: 0001 | Zero Flag: 1
```

```
Time: 1000 | Value: 0001 | Zero Flag: 1
Time: 1100 | Value: 0001 | Zero Flag: 1
testbench.sv:44: $stop called at 1100 (100ps)
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 1100 ticks.
Execution interrupted or reached maximum runtime.
Exit code expected: 0, received: 137
Done
```

4. **Race Condition Handling**

   To avoid race conditions in sequential designs, the following guidelines were implemented:

   - **Nonblocking Assignments for Sequential Logic**: Used nonblocking (`<=`) for sequential operations.
   - **Blocking Assignments for Combinational Logic**: Applied blocking (`=`) for purely combinational `always` blocks.
   - **Consistent Assignment Practice**: Avoided mixing blocking and nonblocking assignments within the same `always` block.
   - **Single Assignment Block for Each Variable**: Ensured that each variable is assigned only in one `always` block.
   - **Avoiding #0 Delays**: #0 delays were not used to prevent timing issues.

5. **XOR Circuit Implementation**

```
`timescale 1ns/100ps

module xor3 (input B_i, D_i, sel_i, clock, output reg E_o);
  always @(posedge clock) begin
    case (sel_i)
      0: E_o <= D_i ^ B_i;
      1: E_o <= B_i;
    endcase
  end
endmodule
```

Testbench

```
`timescale 1ns/100ps

module xor3_tb;
```

```
    reg B_i, D_i, sel_i, clock;
    wire E_o;

    xor3 uut (.B_i(B_i), .D_i(D_i), .sel_i(sel_i), .clock(clock), .E_o(E_o));

    initial begin
        clock = 0;
        forever #5 clock = ~clock;
    end

    initial begin
        // Test cases
        B_i = 0; D_i = 1; sel_i = 0; #10;
        B_i = 1; D_i = 1; sel_i = 1; #10;
        B_i = 0; D_i = 0; sel_i = 0; #10;
        $stop;
    end
endmodule
```