Nang Thiri Wutyi
20113
10/15/2024

Verilog Homework #3

1. **Design module**

```
module designModule (
    input a_i,
    input b_i,
    input c_i,
    output f_o
);

wire ab_n;   // Wire for (a + b)'

supply1 vdd; // Supply voltage (Vdd)
supply0 gnd; // Ground (Gnd)

// PMOS Transistors (for (a + b)')
pmos P1 (ab_n, vdd, a_i);  // PMOS pulls ab_n high when a_i = 0
pmos P2 (ab_n, vdd, b_i);  // PMOS pulls ab_n high when b_i = 0

// NMOS Transistors (for (a + b)')
nmos N1 (ab_n, gnd, a_i);  // NMOS pulls ab_n low when a_i = 1
nmos N2 (ab_n, gnd, b_i);  // NMOS pulls ab_n low when b_i = 1

// AND operation with c (f = ab_n AND c)
wire f_intermediate;       // Intermediate signal for AND operation

pmos P3 (f_intermediate, vdd, ab_n);  // PMOS pulls f_intermediate high if ab_n is 0
pmos P4 (f_o, vdd, c_i);              // PMOS for AND operation with c

nmos N3 (f_intermediate, gnd, ab_n);  // NMOS pulls f_intermediate low if ab_n = 1
nmos N4 (f_o, gnd, c_i);              // NMOS pulls f_o low if c_i = 1

nmos N5 (f_o, f_intermediate, gnd);   // Ensures final AND result with c

endmodule
```

Testbench

```
module testbench;
```

```
reg a_r, b_r, c_r;
wire f_w;

// Instantiate the design module
designModule u (
    .a_i(a_r),
    .b_i(b_r),
    .c_i(c_r),
    .f_o(f_w)
);

initial begin
    // Initialize and monitor the values
    $monitor("At time %0t, a = %b, b = %b, c = %b -> f = %b", $time, a_r, b_r, c_r, f_w);

    // Test all input combinations
    a_r = 1'b0; b_r = 1'b0; c_r = 1'b0; #5;
    a_r = 1'b0; b_r = 1'b0; c_r = 1'b1; #5;
    a_r = 1'b0; b_r = 1'b1; c_r = 1'b0; #5;
    a_r = 1'b0; b_r = 1'b1; c_r = 1'b1; #5;
    a_r = 1'b1; b_r = 1'b0; c_r = 1'b0; #5;
    a_r = 1'b1; b_r = 1'b0; c_r = 1'b1; #5;
    a_r = 1'b1; b_r = 1'b1; c_r = 1'b0; #5;
    a_r = 1'b1; b_r = 1'b1; c_r = 1'b1; #5;

    // Test with x and z values
    a_r = 1'bx; b_r = 1'b0; c_r = 1'b1; #5;
    a_r = 1'bz; b_r = 1'b1; c_r = 1'b0; #5;

    // Finish the simulation
    $finish;
end

endmodule
```

Results

```
[2024-10-15 16:44:55 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
At time 0, a = 0, b = 0, c = 0 -> f = 1
At time 5, a = 0, b = 0, c = 1 -> f = 0
At time 10, a = 0, b = 1, c = 0 -> f = 1
At time 15, a = 0, b = 1, c = 1 -> f = 0
At time 20, a = 1, b = 0, c = 0 -> f = 1
At time 25, a = 1, b = 0, c = 1 -> f = 0
At time 30, a = 1, b = 1, c = 0 -> f = 1
At time 35, a = 1, b = 1, c = 1 -> f = 0
At time 40, a = x, b = 0, c = 1 -> f = 0
At time 45, a = z, b = 1, c = 0 -> f = 1
testbench.sv:33: $finish called at 50 (1s)
Done
```

2. **Design module**

```
// design.sv
primitive mux2to1 (y, i0, i1, sel);
   output y;
   input i0, i1, sel;

   table
   // i0  i1  sel :  y
     0   0    0  :  0;  // If sel = 0, output i0
     0   0    1  :  0;  // If sel = 1, output i1
     0   1    0  :  1;  // If sel = 0, output i0
     0   1    1  :  1;  // If sel = 1, output i1
     1   0    0  :  1;  // If sel = 0, output i0
     1   0    1  :  0;  // If sel = 1, output i1
     1   1    0  :  1;  // If sel = 0, output i0
     1   1    1  :  1;  // If sel = 1, output i1
     ?   ?    0  :  0;   // If sel = 0, output is i0 (don't care)
     ?   ?    1  :  1;   // If sel = 1, output is i1 (don't care)
   endtable
endprimitive
```

Testbench

```
module testbench;
   reg i0, i1, sel;
   wire y;

   // Instantiate the mux UDP
   mux2to1 my_mux (y, i0, i1, sel);

   initial begin
      $monitor("At time %0t, i0 = %b, i1 = %b, sel = %b -> y = %b", $time, i0, i1, sel, y);

      // Test cases
```

```
        i0 = 0; i1 = 0; sel = 0; #5; // Expect y = 0
        i0 = 0; i1 = 1; sel = 0; #5; // Expect y = 0
        i0 = 0; i1 = 1; sel = 1; #5; // Expect y = 1
        i0 = 1; i1 = 0; sel = 0; #5; // Expect y = 1
        i0 = 1; i1 = 0; sel = 1; #5; // Expect y = 0
        i0 = 1; i1 = 1; sel = 1; #5; // Expect y = 1
        i0 = 0; i1 = 1; sel = 1'bx; #5; // Testing X condition
        i0 = 1; i1 = 0; sel = 1'bx; #5; // Testing X condition
        i0 = 0; i1 = 1; sel = 1'bz; #5; // Testing Z condition
        i0 = 1; i1 = 0; sel = 1'bz; #5; // Testing Z condition
        $finish;
    end
endmodule
```

Results

```
[2024-10-15 17:13:48 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
At time 0, i0 = 0, i1 = 0, sel = 0 -> y = 0
At time 5, i0 = 0, i1 = 1, sel = 0 -> y = 0
At time 10, i0 = 0, i1 = 1, sel = 1 -> y = 1
At time 15, i0 = 1, i1 = 0, sel = 0 -> y = 0
At time 20, i0 = 1, i1 = 0, sel = 1 -> y = 0
At time 25, i0 = 1, i1 = 1, sel = 1 -> y = 1
At time 30, i0 = 0, i1 = 1, sel = x -> y = x
At time 35, i0 = 1, i1 = 0, sel = x -> y = x
At time 40, i0 = 0, i1 = 1, sel = z -> y = x
At time 45, i0 = 1, i1 = 0, sel = z -> y = x
testbench.sv:22: $finish called at 50 (1s)
Done
```

3. **Design module**

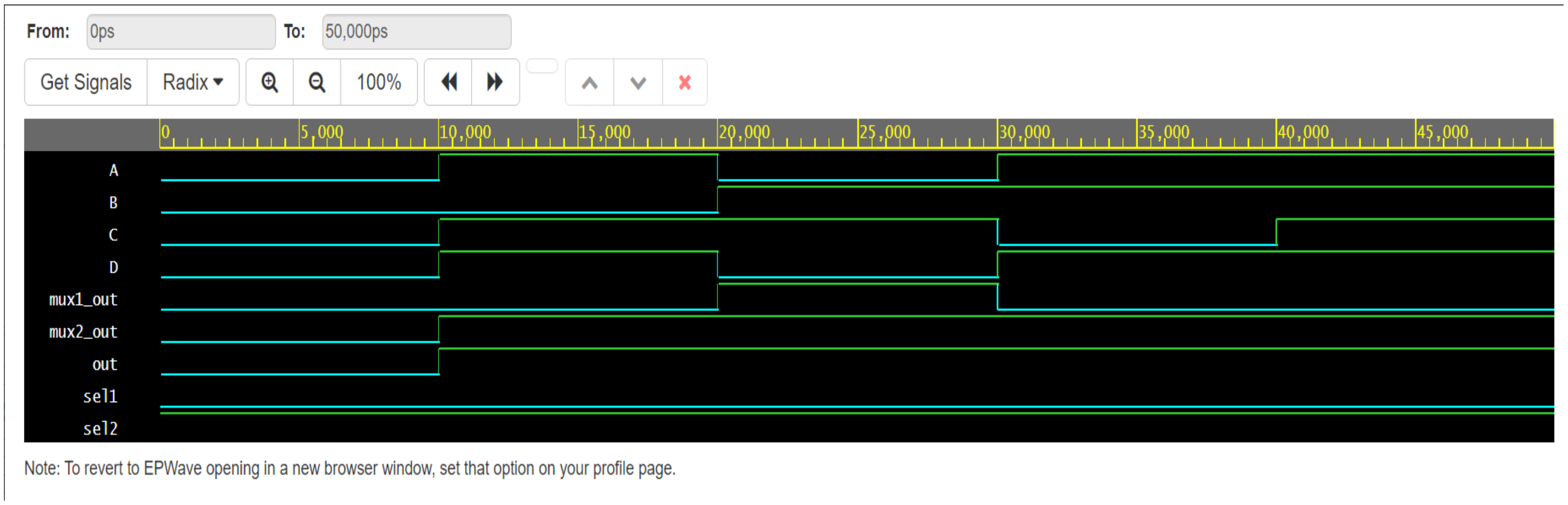
```
`timescale 1ns/1ps

// Define UDP for a 2-to-1 multiplexer
primitive mux2to1 (out, A, B, sel);
    input A, B, sel;
    output out;

    table
    // sel A  B  : out
       0   0  ?  : 0;
       0   1  ?  : 1;
       1   ?  0  : 0;
       1   ?  1  : 1;
    endtable
endprimitive

module top_module(
    input A,
    input B,
    input C,
    input D,
    output out
);
    wire mux1_out, mux2_out;
    reg sel1, sel2;

    // Select lines for multiplexers (2-to-1 muxes)
    assign sel1 = 0; // Choosing between A and B
    assign sel2 = 1; // Choosing between C and D

    // Instantiate two multiplexers
    mux2to1 mux1 (mux1_out, A, B, sel1);
    mux2to1 mux2 (mux2_out, C, D, sel2);

    // Final output multiplexer
    mux2to1 final_mux (out, mux1_out, mux2_out, sel2); // Using the same sel2 for final
selection
endmodule
```

Testbench module

```
`timescale 1ns/1ps

module tb_top_module;
    reg A, B, C, D;
    wire out;

    // Instantiate the top module
    top_module uut (
        .A(A),
        .B(B),
        .C(C),
        .D(D),
        .out(out)
    );

    initial begin
        $dumpfile("mux_waveform.vcd");
        $dumpvars(0, tb_top_module);
        $monitor("Time: %0t, A = %b, B = %b, C = %b, D = %b -> out = %b", $time, A, B,
C, D, out);

        // Apply test inputs
        A = 0; B = 0; C = 0; D = 0; #10;
        A = 1; B = 0; C = 1; D = 1; #10;
        A = 0; B = 1; C = 1; D = 0; #10;
        A = 1; B = 1; C = 0; D = 1; #10;
        A = 1; B = 1; C = 1; D = 1; #10;

        $finish;
    end
endmodule
```

Results

```
[2024-10-17 06:58:13 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
VCD info: dumpfile mux_waveform.vcd opened for output.
Time: 0, A = 0, B = 0, C = 0, D = 0 -> out = 0
Time: 10000, A = 1, B = 0, C = 1, D = 1 -> out = 1
Time: 20000, A = 0, B = 1, C = 1, D = 0 -> out = 1
Time: 30000, A = 1, B = 1, C = 0, D = 1 -> out = 1
Time: 40000, A = 1, B = 1, C = 1, D = 1 -> out = 1
testbench.sv:28: $finish called at 50000 (1ps)
Done
```



Note: To revert to EPWave opening in a new browser window, set that option on your profile page.

Mux1 (Inputs A and B):

Delay from A to mux1: Td = 5

Delay from B to mux1: Td = 5

Total Delay from mux1 output to final output: Td (mux1) = 8

Therefore, the total delay from inputs A or B through mux1 to the final output

Total Delay through mux1: Total Delay from A=5+8=13

Mux2 (Inputs C and D):

Delay from C to mux2: Td = 3

Delay from D to mux2: Td = 3

Total Delay from mux2 output to final output: Td (mux2) = 8

Therefore, the total delay from inputs C or D through mux2 to the final output

Total Delay through mux2: Total Delay from C=3+8=11

Longest Path Calculation

Final Output Delay:

The longest path from the inputs to the output, considering both mux outputs, will be:

From mux1: 13 (from either A or B)

From mux2: 11 (from either C or D)

Therefore, the overall longest path delay is: Longest Path Delay=max(13,11)=13

4. **Issues**
    - Improper Output Declaration: The output[1:0] a, r; declaration has a syntax error. Verilog allows scalar (1-bit) outputs in primitives, but you cannot declare a vector like [1:0] in primitives. Primitives only support single-bit outputs.
    - Redundant reg Declaration: In primitives, you don't need reg declarations. The output in a primitive behaves like a reg implicitly.
    - Invalid inout Declaration: The inout declaration for c is not allowed in Verilog primitives. Primitives only support inputs and outputs, not bidirectional signals.

- Improper Truth Table Format: The truth table has rows that are not properly formatted. Each row must clearly specify the input/output relation. Also, the values in the truth table must follow the allowed characters (0, 1, ?, and - for don't-care conditions).
- Invalid Include Directive: The #include directive (#include “basicPrimitive.v”) is not standard Verilog syntax. Verilog uses include ` (backticks) for including files.
- Misuse of basicPrimitive Instantiation: A primitive cannot instantiate another module or primitive directly. This line at the end of the primitive definition should be removed. Primitives are self-contained and cannot instantiate other modules.
  ```
  basicPrimitive u0(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r);
  ```

Fixed code

```
`include "basicPrimitive.v"  // Correct Verilog include syntax

primitive example(r, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q);
   output r;              // Primitives only support scalar outputs (1-bit)
   input  b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q;

   table
   // r : b c d e f g h i j k l m n o p q
     0 : 0 ? ? 1 ? 0 ? 1 ? 0 ? 1 ? 0 ? 1;  // Output is '0' for these inputs
     1 : 1 ? ? 1 ? 0 ? 1 ? 0 ? 1 ? 0 ? 1;  // Output is '1' for these inputs
     0 : ? ? ? 0 ? 1 ? 0 ? 1 ? 0 ? 1 ? 1;  // Another example case
     // Add more rows to fully describe the behavior.
   endtable

endprimitive
```

Explanation

- **Output is Single-Bit**: The primitive now has a single-bit output r. Primitives cannot have vector outputs like [1:0]. If you need a multi-bit output, you would have to design multiple primitives or use regular Verilog modules.
- **Removed reg Declaration**: The reg[1:0] a, r; declaration has been removed. In Verilog primitives, the output is implicitly treated as a reg, so there is no need for this declaration.
- **Removed inout**: The inout c; has been removed. Primitives in Verilog do not allow bidirectional (inout) ports. You should define c as either an input or output depending on its intended usage.
- **Truth Table Formatting**: The truth table has been properly formatted:

1. Each row of the table corresponds to a unique set of inputs (b, c, d, e, ..., q) and their associated output (r).
2. The ? symbol is used for "don't care" conditions, indicating that the specific input doesn't affect the output for that condition.
3. Only binary values (0 or 1) are allowed for inputs and outputs in primitives.

- **Removed Instantiation of basicPrimitive**: The instantiation of basicPrimitive u0 has been removed, as a primitive cannot instantiate another module or primitive. Primitives are standalone and cannot instantiate other components.
  **Proper Use of include **: The include directive now uses the correct Verilog syntax (include "basicPrimitive.v") to include the external file. Ensure that the basicPrimitive.v file exists and is in the correct location for this to work.

5. **Design module**

```
primitive DFF_UDP (Q, D, clk, reset);
  output Q;
  reg Q;
  input D, clk, reset;

  table
  // D  clk  reset :  Q : Q_next
    ?   ?    1    :  ? : 0;   // When reset is 1, Q is set to 0
    0   (01) 0    :  ? : 0;   // On rising edge of clk, if D is 0, Q becomes 0
    1   (01) 0    :  ? : 1;   // On rising edge of clk, if D is 1, Q becomes 1
    ?   (??) 0    :  ? : -;   // Ignore other clock edges
  endtable
endprimitive
module three_bit_counter (
  input clk,        // Clock signal
  input reset,      // Synchronous reset
  output reg [2:0] out  // 3-bit output
);
  // Synchronous reset with counter logic
  always @(posedge clk) begin
    if (reset)
      out <= 3'b000;  // Reset the counter to 000
    else
      out <= out + 1;  // Increment counter on each clock cycle
  end
endmodule
```

Testbench module

```
// Testbench for 3-bit Counter
module tb_three_bit_counter;
   reg clk;
   reg reset;
   wire [2:0] out;

   // Instantiate the counter module
   three_bit_counter uut (
      .clk(clk),
      .reset(reset),
      .out(out)
   );

   // Clock generation (toggle every 5 time units)
   initial begin
      clk = 0;
      forever #5 clk = ~clk;  // Toggle clock every 5 time units
   end

   // Test sequence
   initial begin
      // Apply reset, then release after 10 time units
      reset = 1;  // Apply reset
      #10 reset = 0;  // Release reset after 10 time units

      // Observe the output for 300 time units, then finish
      #300 $finish;
   end

   // Monitor the output
   initial begin
      $monitor("Time: %0d, Counter Output: %b", $time, out);
   end

   // Display output on every positive clock edge
   always @(posedge clk) begin
      $display("At time %0t, Counter Value = %b", $time, out);
   end
```

endmodule

Results

```
[2024-10-15 17:53:15 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
Time: 0, Counter Output: xxx
At time 5, Counter Value = xxx
Time: 5, Counter Output: 000
At time 15, Counter Value = 000
Time: 15, Counter Output: 001
At time 25, Counter Value = 001
Time: 25, Counter Output: 010
At time 35, Counter Value = 010
Time: 35, Counter Output: 011
At time 45, Counter Value = 011
Time: 45, Counter Output: 100
At time 55, Counter Value = 100
Time: 55, Counter Output: 101
At time 65, Counter Value = 101
Time: 65, Counter Output: 110
At time 75, Counter Value = 110
Time: 75, Counter Output: 111
At time 85, Counter Value = 111
Time: 85, Counter Output: 000
At time 95, Counter Value = 000
Time: 95, Counter Output: 001
At time 105, Counter Value = 001
Time: 105, Counter Output: 010
At time 115, Counter Value = 010
Time: 115, Counter Output: 011
At time 125, Counter Value = 011
Time: 125, Counter Output: 100
At time 135, Counter Value = 100
Time: 135, Counter Output: 101
At time 145, Counter Value = 101
Time: 145, Counter Output: 110
At time 155, Counter Value = 110
Time: 155, Counter Output: 111
At time 165, Counter Value = 111
Time: 165, Counter Output: 000
At time 175, Counter Value = 000
Time: 175, Counter Output: 001
At time 185, Counter Value = 001
Time: 185, Counter Output: 010
At time 195, Counter Value = 010
Time: 195, Counter Output: 011
At time 205, Counter Value = 011
Time: 205, Counter Output: 100
At time 215, Counter Value = 100
Time: 215, Counter Output: 101
At time 225, Counter Value = 101
Time: 225, Counter Output: 110
At time 235, Counter Value = 110
Time: 235, Counter Output: 111
```

```
At time 245, Counter Value = 111
Time: 245, Counter Output: 000
At time 255, Counter Value = 000
Time: 255, Counter Output: 001
At time 265, Counter Value = 001
Time: 265, Counter Output: 010
At time 275, Counter Value = 010
Time: 275, Counter Output: 011
At time 285, Counter Value = 011
Time: 285, Counter Output: 100
At time 295, Counter Value = 100
Time: 295, Counter Output: 101
At time 305, Counter Value = 101
Time: 305, Counter Output: 110
testbench.sv:27: $finish called at 310 (1s)
Done
```