RTL CHALLENGE

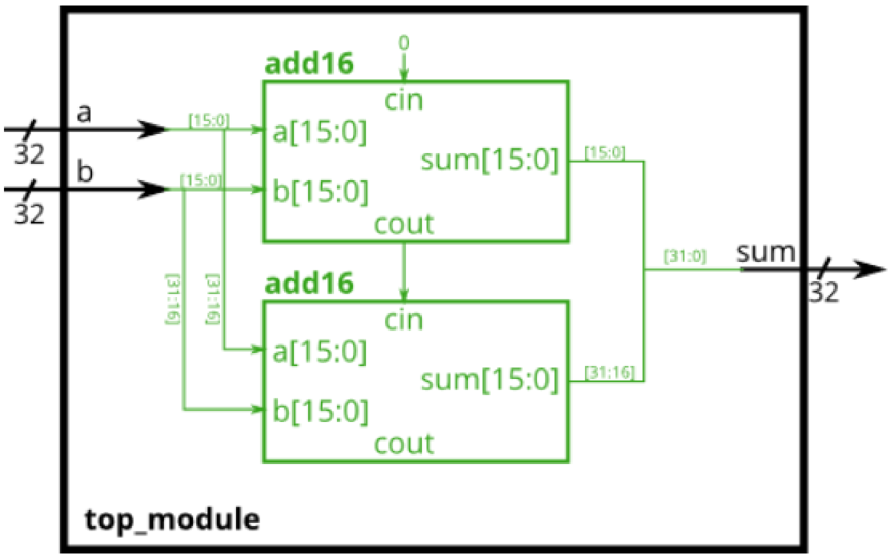
DAY – 14 :- Today I Practiced Some problems.

Website used:- HDL BITS

Problem Statements:-

You are given a module add16 that performs a 16-bit addition. Instantiate two of them to create a 32-bit adder. One add16 module computes the lower 16 bits of the addition result, while the second add16 module computes the upper 16 bits of the result, after receiving the carry-out from the first adder. Your 32-bit adder does not need to handle carry-in (assume 0) or carry-out (ignored), but the internal modules need to in order to function correctly.

module add16 (input[15:0] **a**, input[15:0] **b**, input **cin**, output[15:0] **sum**, output **cout**);



Write your solution here

[Load a previous submission] Load

```
module top_module(
    input [31:0] a,
    input [31:0] b,
    output [31:0] sum
);
    wire w1;
    add16 a1 (.a(a[15:0]),.b(b[15:0]),.cin(1'b0),.sum(sum[15:0]),.cout(w1));
    add16 a2 (.a(a[31:16]),.b(b[31:16]),.cin(w1),.sum(sum[31:16]));

endmodule
```

Submit | Submit (new window)

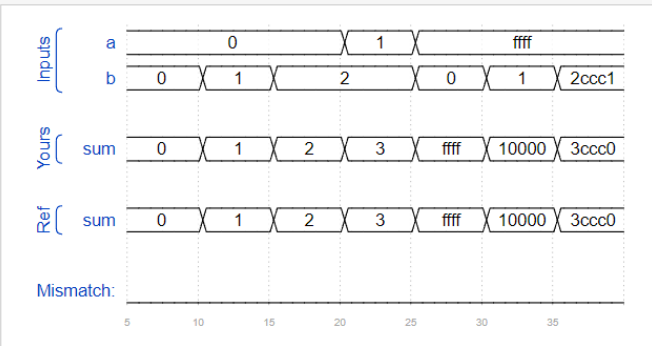
Status: Success!

You have solved 6 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

32-bit adder



 26/03/2024

RTL CHALLENGE

Problem Statement:-

Build an AND gate using both an assign statement and a combinational always block. (Since assign statements and combinational always blocks function identically, there is no way to enforce that you're using both methods. But you're here for practice, right?...)

Write your solution here

[Load a previous submission] Load

```
// synthesis verilog_input_version verilog_2001
module top_module(
    input a,
    input b,
    output wire out_assign,
    output reg out_alwaysblock
);
    assign out_assign = a&b;
    always @(*)
        out_alwaysblock = a&b;
endmodule

```

Submit | Submit (new window)

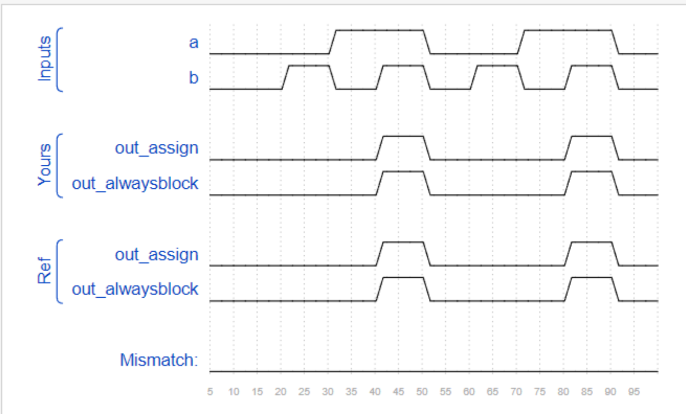
Status: Success!

You have solved 7 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

AND gate



RTL CHALLENGE

Problem Statement:-

Build an XOR gate three ways, using an assign statement, a combinational always block, and a clocked always block. Note that the clocked always block produces a different circuit from the other two: There is a flip-flop so the output is delayed.

Write your solution here

[Load a previous submission] Load

```
// synthesis verilog_input_version verilog_2001
module top_module(
    input clk,
    input a,
    input b,
    output wire out_assign,
    output reg out_always_comb,
    output reg out_always_ff   );
    assign out_assign = a^b;
    always@(*)
        out_always_comb = a^b;
    always@(posedge clk)
        out_always_ff = a^b;

endmodule

```

Submit | Submit (new window)

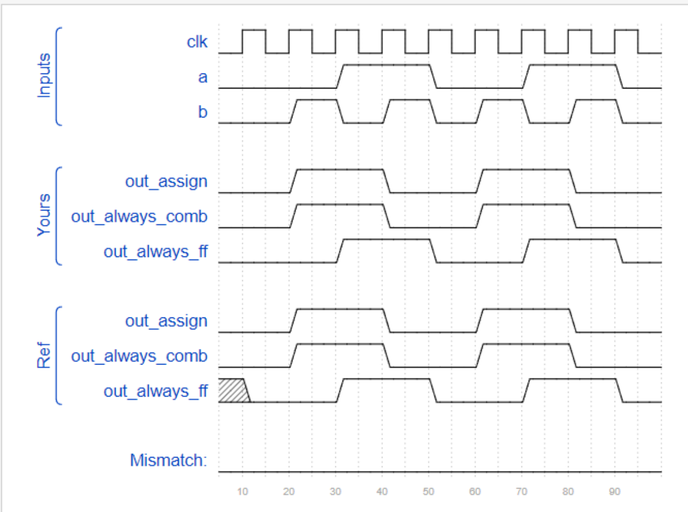
Status: Success!

You have solved 8 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

XOR gate



RTL CHALLENGE

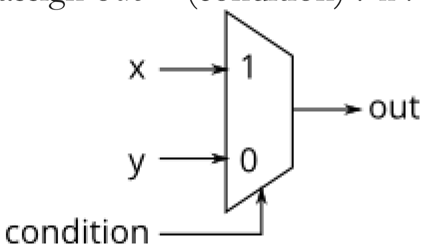
Problem Statement:-

An if statement usually creates a 2-to-1 multiplexer, selecting one input if the condition is true, and the other input if the condition is false.

```
always @(*) begin
    if (condition) begin
        out = x
       end
    else begin
        out = y;
    end
end
```

This is equivalent to using a continuous assignment with a conditional operator:

```
assign out = (condition) ? x : y;
```



Build a 2-to-1 mux that chooses between a and b.
Choose b if *both* sel_b1 and sel_b2 are true. Otherwise, choose a. Do the same twice, once using assign statements and once using a procedural if statement.

sel_b1	sel_b2	out_assign out_always
0	0	a
0	1	a
1	0	a
1	1	b

Write your solution here

[Load a previous submission] Load

```
// synthesis verilog_input_version verilog_2001
module top_module(
    input a,
    input b,
    input sel_b1,
    input sel_b2,
    output wire out_assign,
    output reg out_always   );
    assign out_assign = (sel_b1 && sel_b2)?b:a;
    always@(*)
        begin
            if(sel_b1==1&& sel_b2 ==1)
                begin
                    out_always = b;
                end
            else
                begin

                   out_always = a;
                end
        end

endmodule

```

Submit | Submit (new window)

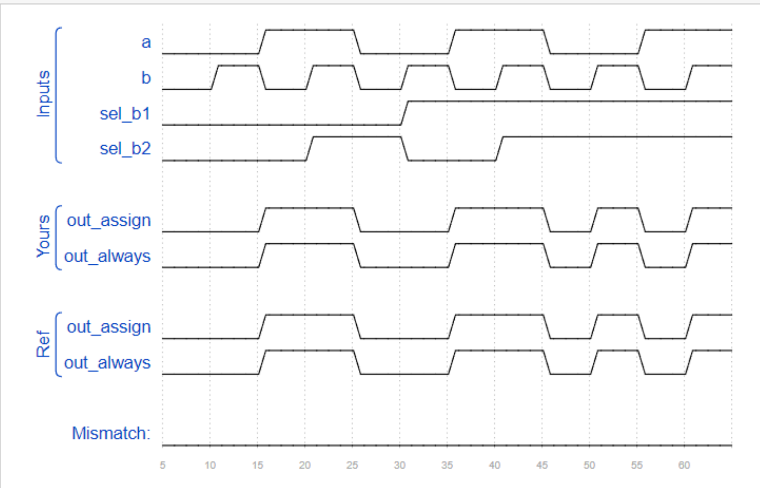
PP

Status: Success!

You have solved 9 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).



RTL CHALLENGE

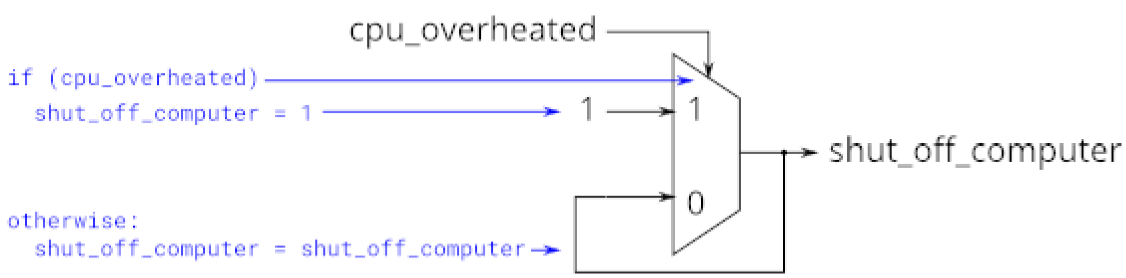
Problem Statement:-

The following code contains incorrect behaviour that creates a latch. Fix the bugs so that you will shut off the computer only if it's really overheated, and stop driving if you've arrived at your destination or you need to refuel.

This is the circuit described by the code, not the circuit you want to build.

```
always @(*) begin
   if (cpu_overheated)
      shut_off_computer = 1;
end

always @(*) begin
   if (~arrived)
      keep_driving = ~gas_tank_empty;
end
```



Write your solution here

[Load a previous submission] Load

```
// synthesis verilog_input_version verilog_2001
module top_module (
    input      cpu_overheated,
    output reg shut_off_computer,
    input      arrived,
    input      gas_tank_empty,
    output reg keep_driving  ); //

    always @(*) begin
        if (cpu_overheated)
           shut_off_computer = 1;
        else
           shut_off_computer = 0;
    end

    always @(*) begin
        if (~arrived)
           keep_driving = ~gas_tank_empty;
        else
            keep_driving = 0;
    end

endmodule

```

Submit | Submit (new window)

Status: Success!

You have solved 10 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).