\[ y = abc + abc + abc = ab(c + c) + abc \]
\[ = ab + abc = a(b + bc) \]
\[ = a[(b + b)(b + c)] = a(b + c) \]

\[ z = ab + ab = a \oplus b \]
Problem: 4.37:

Verilog:

module ex4_37(input clk, reset, a, b, output reg z);
reg [1:0] state, nextstate;
parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
parameter S3 = 2'b11;
// State Register
always @(posedge clk, posedge reset)
if (reset) state <= S0;
else state <= nextstate;
// Next State Logic
always @(*)
case (state)
  S0: case ({b,a})
    2'b00: nextstate = S0;
    2'b01: nextstate = S3;
    2'b10: nextstate = S0;
    2'b11: nextstate = S1;
  endcase
  S1: case ({b,a})
    2'b00: nextstate = S0;
    2'b01: nextstate = S3;
    2'b10: nextstate = S2;
    2'b11: nextstate = S1;
  endcase
  S2: case ({b,a})
    2'b00: nextstate = S0;
    2'b01: nextstate = S3;
    2'b10: nextstate = S2;
    2'b11: nextstate = S1;
  endcase
  S3: case ({b,a})
    2'b00: nextstate = S0;
    2'b01: nextstate = S3;
    2'b10: nextstate = S2;
    2'b11: nextstate = S1;
  endcase
  default: nextstate = S0;
endcase
// Output Logic
always @(*)
case (state)
  S0: z = a & b;
  S1: z = a | b;
  S2: z = a & b;
  S3: z = a | b;
  default: z = 1'b0;
endcase
endmodule
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity ex4_37 is
port(clk: in STD_LOGIC;
reset: in STD_LOGIC;
a, b: in STD_LOGIC;
z: out STD_LOGIC);
end;
architecture synth of ex4_37 is
type statetype is (S0, S1, S2, S3);
signal state, nextstate: statetype;
signal ba: STD_LOGIC_VECTOR(1 downto 0);
begin
-- state register
process(clk, reset) begin
if reset = '1' then state <= S0;
elsif clk'event and clk = '1' then
state <= nextstate;
end if;
end process;
-- next state logic
ba <= b & a;
process (state, a, b) begin
    case state is
        when S0 =>
            case (ba) is
                when "00" => nextstate <= S0;
                when "01" => nextstate <= S3;
                when "10" => nextstate <= S0;
                when "11" => nextstate <= S1;
                when others => nextstate <= S0;
            end case;
        when S1 =>
            case (ba) is
                when "00" => nextstate <= S0;
                when "01" => nextstate <= S3;
                when "10" => nextstate <= S2;
                when "11" => nextstate <= S1;
                when others => nextstate <= S0;
            end case;
        when S2 =>
            case (ba) is
                when "00" => nextstate <= S0;
                when "01" => nextstate <= S3;
                when "10" => nextstate <= S2;
                when "11" => nextstate <= S1;
                when others => nextstate <= S0;
            end case;
        when S3 =>
            case (ba) is
                when "00" => nextstate <= S0;
                when "01" => nextstate <= S3;
                when "10" => nextstate <= S2;
                when "11" => nextstate <= S1;
                when others => nextstate <= S0;
            end case;
        when others =>
            nextstate <= S0;
    end case;
end if;
end process;
-- output logic
process (state, a, b) begin
case state is
  when S0 => if (a = '1' and b = '1')
    then z <= '1';
    else z <= '0';
  end if;
  when S1 => if (a = '1' or b = '1')
    then z <= '1';
    else z <= '0';
  end if;
  when S2 => if (a = '1' and b = '1')
    then z <= '1';
    else z <= '0';
  end if;
  when S3 => if (a = '1' or b = '1')
    then z <= '1';
    else z <= '0';
  end if;
  when others => z <= '0';
end case;
end process;

Interview Question 4.2:

HDLs support blocking and nonblocking assignments in an always / process statement. A group of blocking assignments are evaluated in the order they appear in the code, just as one would expect in a standard programming language. A group of nonblocking assignments are evaluated concurrently; all of the statements are evaluated before any of the left hand sides are updated.

Verilog
In a Verilog always statement, = indicates a blocking assignment and <= indicates a nonblocking assignment. Assign statements are normally used outside always statements and are also evaluated concurrently.

VHDL
In a VHDL process statement, := indicates a blocking assignment and <= indicates a nonblocking assignment (also called a concurrent assignment). This is the first section where := is introduced. Nonblocking assignments are made to outputs and to signals. Blocking assignments are made to variables, which are declared in process statements. <= can also appear outside process statements, where it is also evaluated concurrently.