Computer Architecture

Lab 4: Verilog Basics 1

Getting Started

- Open a web browser and go to https://www.edaplayground.com/
- Create an account with your university email

Always Statement

General Structure:

```
always @(sensitivity list)
  statement;
```

Whenever the event in `sensitivity list` occurs, `statement` is executed

Combinational Logic using always

```
// combinational logic using an always statement
module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2, y3, y4, y5);
  always_comb          // need begin/end because there is
    begin              // more than one statement in always
      y1 = a & b;      // AND
      y2 = a | b;      // OR
      y3 = a ^ b;      // XOR
      y4 = ~(a & b);   // NAND
      y5 = ~(a | b);   // NOR
    end
endmodule
```

This hardware could be described with assign statements using fewer lines of code, so it's better to use assign statements in this case.

Other Behavioral Statements

- Statements that must be inside `always` statements:
  - `if / else`
  - `case, casez`

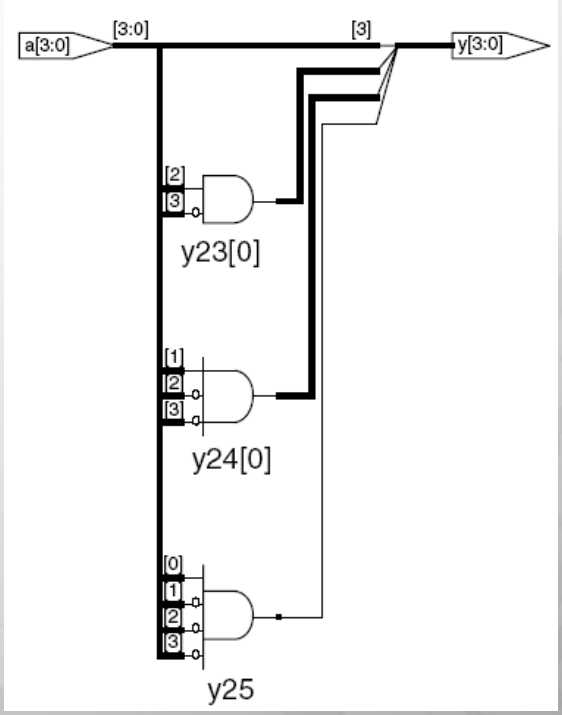
```
// combinational logic using an always statement
module mux(input  logic a, b, s
           output logic y);
  always_comb      // always @ (a, b, s) in Verilog
    if(s)
      y = a;
    else
      y = b;
endmodule
```

Combinational Logic using case

```
module sevenseg(input  logic [3:0] data,
                output logic [6:0] segments);
  always_comb
    case (data)
      //                        abc_defg
      0: segments =        7'b111_1110;
      1: segments =        7'b011_0000;
      2: segments =        7'b110_1101;
      3: segments =        7'b111_1001;
      4: segments =        7'b011_0011;
      5: segments =        7'b101_1011;
      6: segments =        7'b101_1111;
      7: segments =        7'b111_0000;
      8: segments =        7'b111_1111;
      9: segments =        7'b111_0011;
      default: segments = 7'b000_0000; // required
    endcase
endmodule
```

Combinational Logic using casez

```
module priority_casez(input  logic [3:0] a,
                      output logic [3:0] y);
  always_comb
    casez(a)
      4'b1???: y = 4'b1000;  // ? = don't care
      4'b01??: y = 4'b0100;
      4'b001?: y = 4'b0010;
      4'b0001: y = 4'b0001;
      default: y = 4'b0000;
   endcase
endmodule
```



Lab 4-1

- Copy and paste the following code to the left top window (testbench.sv)

```
module testbench4();
  logic [3:0] in;
  logic [1:0] out;
  // instantiate device under test
  priority_casez dut(in, out);

  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    in = 4'b1101; #20;
    in = 4'b0111; #20;
    in = 4'b0010; #20;
    in = 4'b0001; #20;
    in = 4'b0000; #20;
  end
endmodule
```

Lab 4-1 (cont'd)

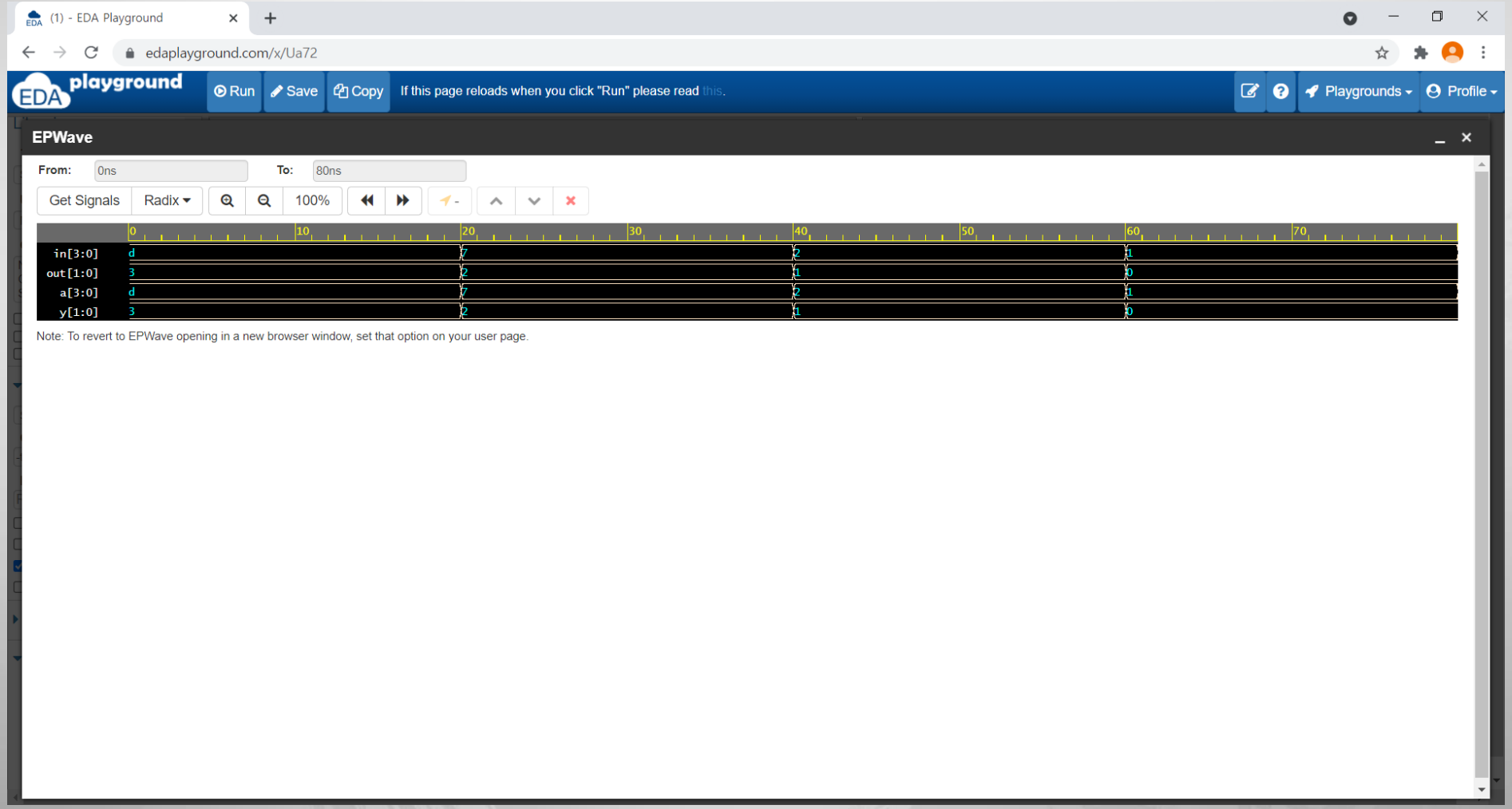
- Copy and paste the following code to the right top window (design.sv)

```
module priority_casez(input [3:0] a,
                      output reg [1:0] y);
  always @(a)
    casez(a)
      4'b1???: y = 2'b11;  // ? = don't care
      4'b01??: y = 2'b10;
      4'b001?: y = 2'b01;
      4'b0001: y = 2'b00;
      default: y = 2'b00;
   endcase
endmodule
```

Lab 4-1 (cont'd)

- Select "Synopsys VCS 2021.09" in Tools & Simulators
- Check "Open EPWave after run"
- Click "Run" on the top menu

Lab 4-1 (Result)



Lab Assignment

- Write a SystemVerilog code to implement the ALU
  - Its functionality is given in the table on the right.
  - Additional requirement: the ALU should generate a signal that indicates whether the result is zero or not
- The ALU has the following inputs and outputs
  - iA (32 bits): source operand 1
  - iB (32 bits): source operand 2
  - iF (3 bits): control signal
  - oY (32 bits): result
  - oZero (1 bit): 1 if oY==0, 0 otherwise
- Use the testbench in the following slide
- Save and submit the link of your design to the Blackboard.
  - Click 🔗 in the bottom window to copy the URL of your design.

$iF_{2:0}$	Function
000	iA & iB
001	iA | iB
010	iA + iB
011	not used
100	iA & ~iB
101	iA | ~iB
110	iA – iB
111	SLT

Testbench for ALU

```
module testbench_alu();
  logic [31:0] a, b, y;
  logic [2:0] f;
  logic zero;
  logic result;

  alu dut(
    .iA (a),
    .iB (b),
    .iF (f),
    .oY (y),
    .oZero (zero)
  );

  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    result = 1'b1;

    a = 32'h1234_5678; b = 32'h0000_ffff; f=3'b000; #10;
    if(y!=32'h0000_5678) begin $display("000 failed."); result=1'b0; end

    a = 32'h1234_5678; b = 32'h0000_ffff; f=3'b001; #10;
    if(y!=32'h1234_ffff) begin $display("001 failed."); result=1'b0; end

    a = 32'h1234_5678; b = 32'h1111_2222; f=3'b010; #10;
    if(y!=32'h2345_789a) begin $display("010 failed."); result=1'b0; end

    a = 32'h1234_5678; b = 32'h0000_ffff; f=3'b100; #10;
    if(y!=32'h1234_0000) begin $display("100 failed."); result=1'b0; end

    a = 32'h1234_5678; b = 32'h0000_ffff; f=3'b101; #10;
    if(y!=32'hffff_5678) begin $display("101 failed."); result=1'b0; end

    a = 32'h1234_5678; b = 32'h1111_2222; f=3'b110; #10;
    if(y!=32'h0123_3456) begin $display("110 failed."); result=1'b0; end

    a = 32'h0000_5678; b = 32'h0000_ffff; f=3'b111; #10;
    if(y!=32'h0000_0001) begin $display("111 failed."); result=1'b0; end

    a = 32'h0000_0000; b = 32'h0000_0000; f=3'b000; #10;
    if(zero!=1) begin $display("zero failed."); result=1'b0; end

    if(result)    $display("SUCCESS!");
    else                          $display("FAILURE!");
  end
endmodule
```