# Computer Architecture

Lab 4: Verilog Basics

# 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`

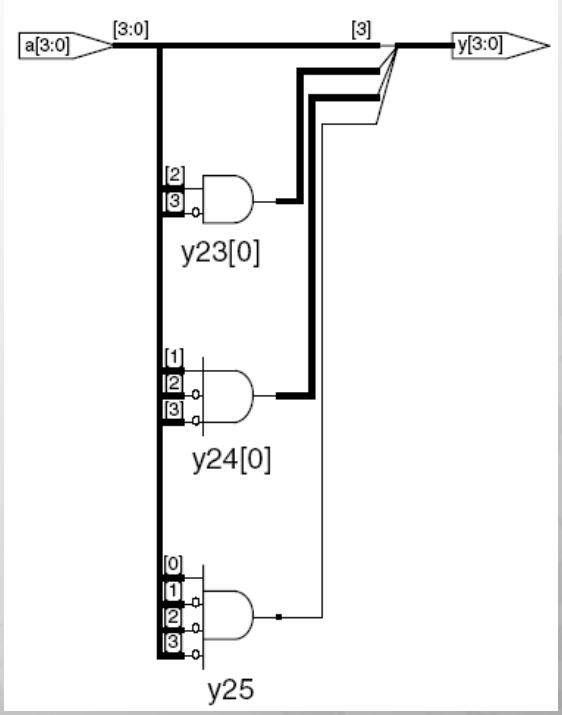
```
// combinational logic using an always statement
module mux(input  logic a, b, s
           output logic y);
  always_comb      // always @ (a, b, s) in Verilog
    if(s)
      y = a;
    else
      y = b;
endmodule
```

# 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
```

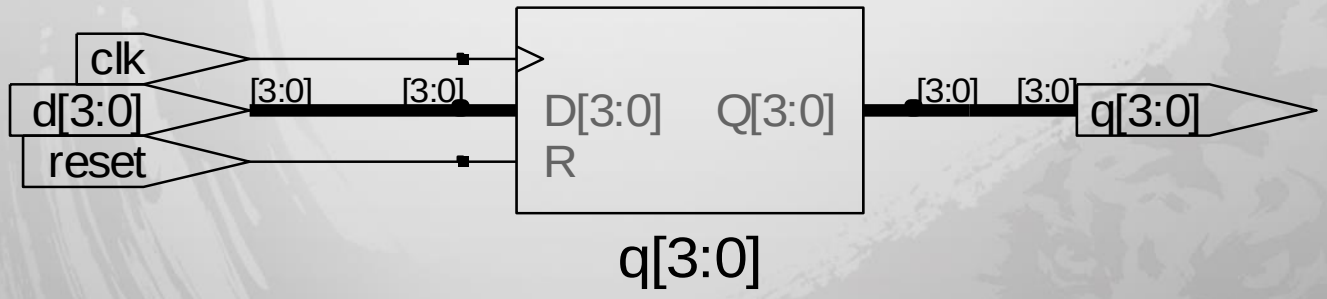
# Combinational Logic using casez

```
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
```



# Resettable D Flip-Flop

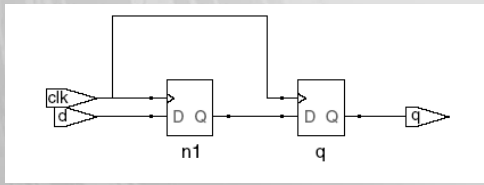
```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  // synchronous reset
  always_ff @(posedge clk)
    if (reset) q <= 4'b0;
    else       q <= d;

endmodule
```



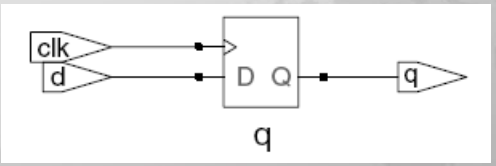
# Blocking vs. Nonblocking Assignment

- <= is nonblocking assignment
  - Occurs simultaneously with others
- = is blocking assignment
  - Occurs in order it appears in file

```
// Good synchronizer using
// nonblocking assignments
module syncgood(input  logic clk,
                input  logic d,
                output logic q);
  logic n1;
  always_ff @(posedge clk)
    begin
      n1 <= d;  // nonblocking
      q  <= n1; // nonblocking
    end
endmodule
```




```
// Bad synchronizer using
// blocking assignments
module syncbad(input logic  clk,
               input  logic d,
               output logic q);
  logic n1;
  always_ff @(posedge clk)
    begin
      n1 = d;  // blocking
      q  = n1; // blocking
    end
endmodule
```



# Rules for Signal Assignment

- **Synchronous sequential logic**: use `always_ff @(posedge clk)` and nonblocking assignments (<=)

```
always_ff @ (posedge clk)
    q <= d; // nonblocking
```

- **Simple combinational logic**: use continuous assignments (`assign…`)

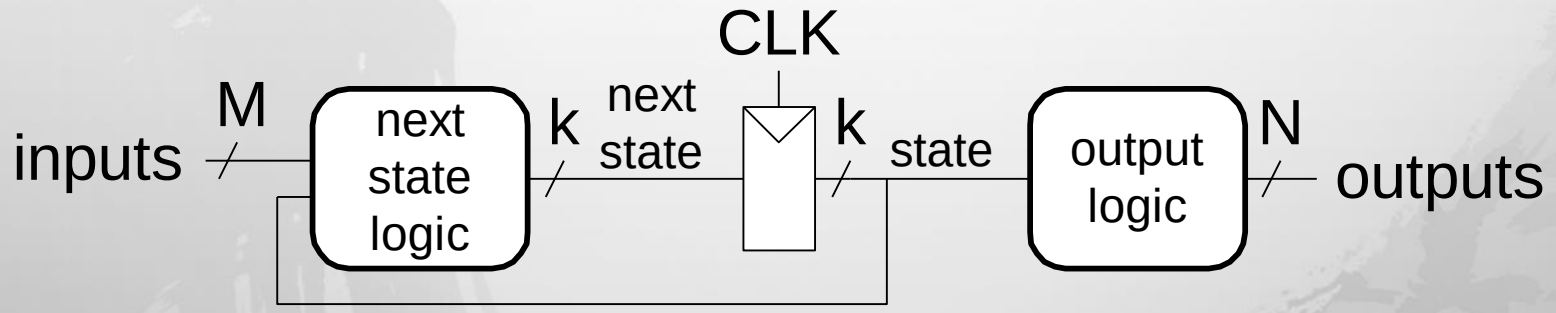
```
assign y = a & b;
```

- **More complicated combinational logic**: use `always_comb` and blocking assignments (=)

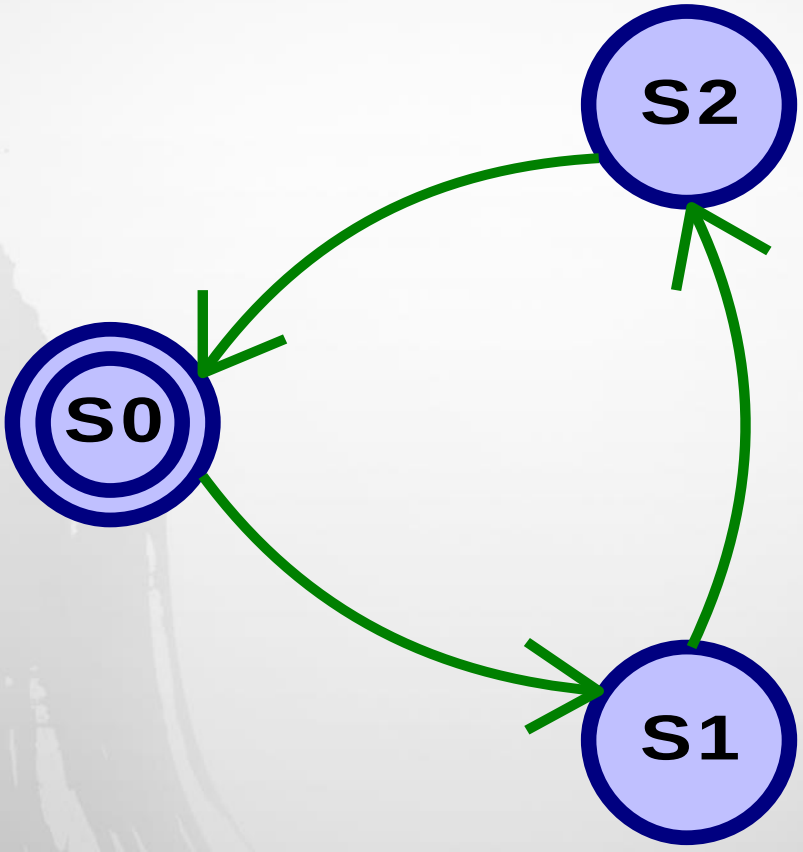
- Assign a signal in **only one** `always` statement or continuous assignment statement.

# Finite State Machines (FSMs)

- Three blocks:
  - next state logic
  - state register
  - output logic



# FSM Example: Divide by 3




The double circle indicates the reset state

# FSM in SystemVerilog

```
module divideby3FSM (input  logic clk,
                     input  logic reset,
                     output logic q);
   typedef enum logic [1:0] {S0, S1, S2} statetype;
   statetype [1:0] state, nextstate;

   // state register
   always_ff @ (posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;

   // next state logic
   always_comb
      case (state)
         S0:      nextstate = S1;
         S1:      nextstate = S2;
         S2:      nextstate = S0;
         default: nextstate = S0;
      endcase

   // output logic
   assign q = (state == S0);
endmodule
```

# Lab 4

- 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 – 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 – cont'd

- Select "Synopsys VCS 2021.09" in Tools & Simulators
- Add "+vcs+finish+100" to Run Options
- Check "Open EPWave after run"
- Click "Run" on the top menu

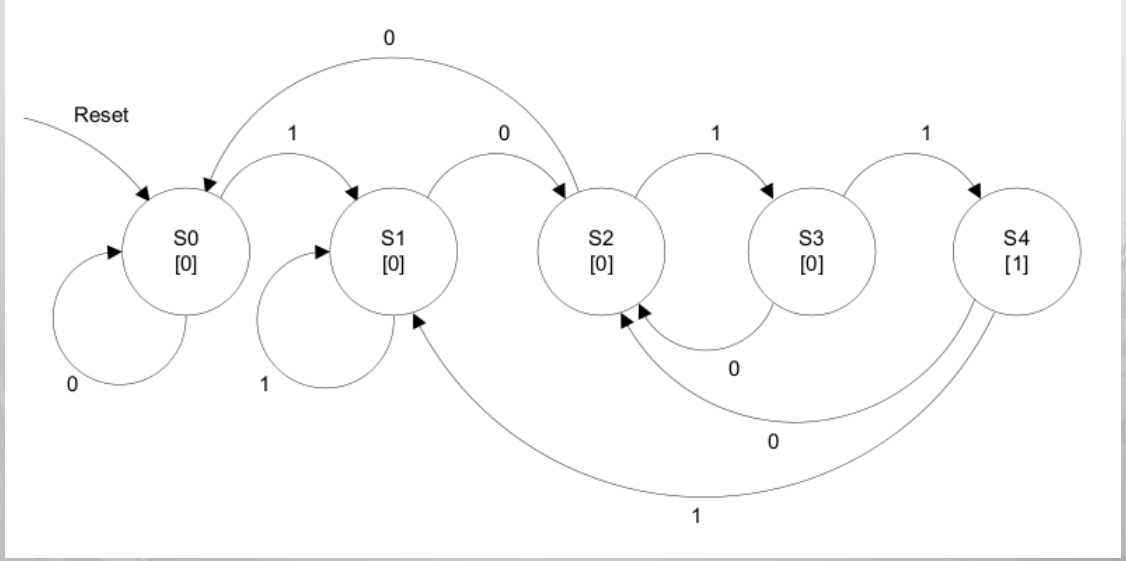
# Lab 4 – Expected Result

# Lab Assignment

- Write a SystemVerilog module that implements the following FSM, which is a 1011 pattern recognizer.
- It has three inputs, `clk`, `reset`, and `in` and an one-bit output `out`.
- Use the testbench given in the next slide.

- Select "Synopsys VCS 2021.09" in Tools & Simulators
- Check "Open EPWave after run"
- Add "+vcs+finish+200" to Run Options
- Run

- Save and submit the link of your design to the Blackboard.
  - Click 🔗 in the bottom window to copy the URL of your design.



# Lab Assignment – cont'd

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

```
module testbench();
  logic clk, rst, in;
  logic out;
  // instantiate device under test
  patterndet dut(clk, rst, in, out);

  always     // no sensitivity list, so it always executes
    begin
      clk = 1; #5; clk = 0; #5;
    end

  initial begin
    $dumpfile("dump.vcd"); $dumpvars;
    in = 0; rst = 1; #21; rst = 0; #10
    in = 1; #30
    in = 0; #40
    in = 1; #10
    in = 0; #10
    in = 1; #40
    in = 0;
  end
endmodule
```

# Expected Result