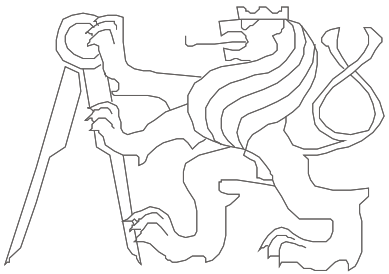
Advanced Computer Architectures

Verilog – introduction

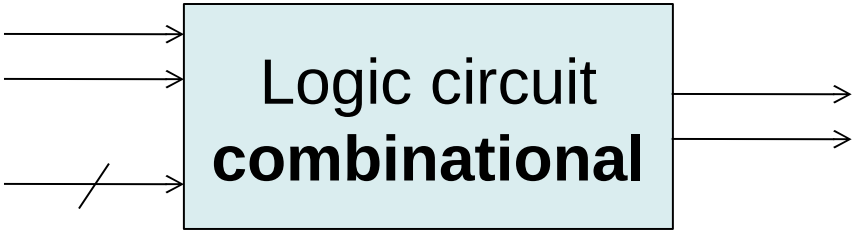


Czech Technical University in Prague, Faculty of Electrical Engineering
Slides authors: Michal Štepanovský, update Pavel Píša

HDL – *Hardware Description Language*

- Two widespread HDL languages:
  - **VHDL** (*VHSIC Hardware Description Language; VHSIC = Very High Speed Integrated Circuits)*
  - **Verilog**
  - Both include even constructs/commands **not** synthetizable into hardware realization
- Design description:
  - Behavioral (function description in abstract way)
  - Structural (interconnection of components)

Verilog – behavioral description

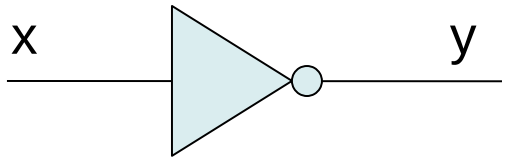



```
module label (input list_of_inputs,
              output list_of_outputs);

   wire declaration_of_internal_variables;

   assign bind_output_to_inputs_and_wires_function;
   assign bind_output_to_inputs_and_wires_function;
endmodule
```

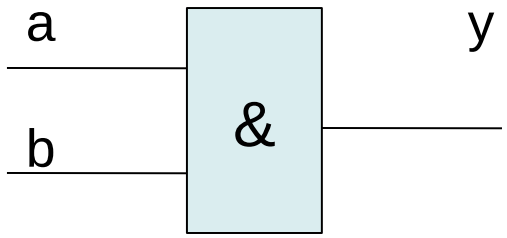
Modul – HW block with inputs and outputs

Verilog – example

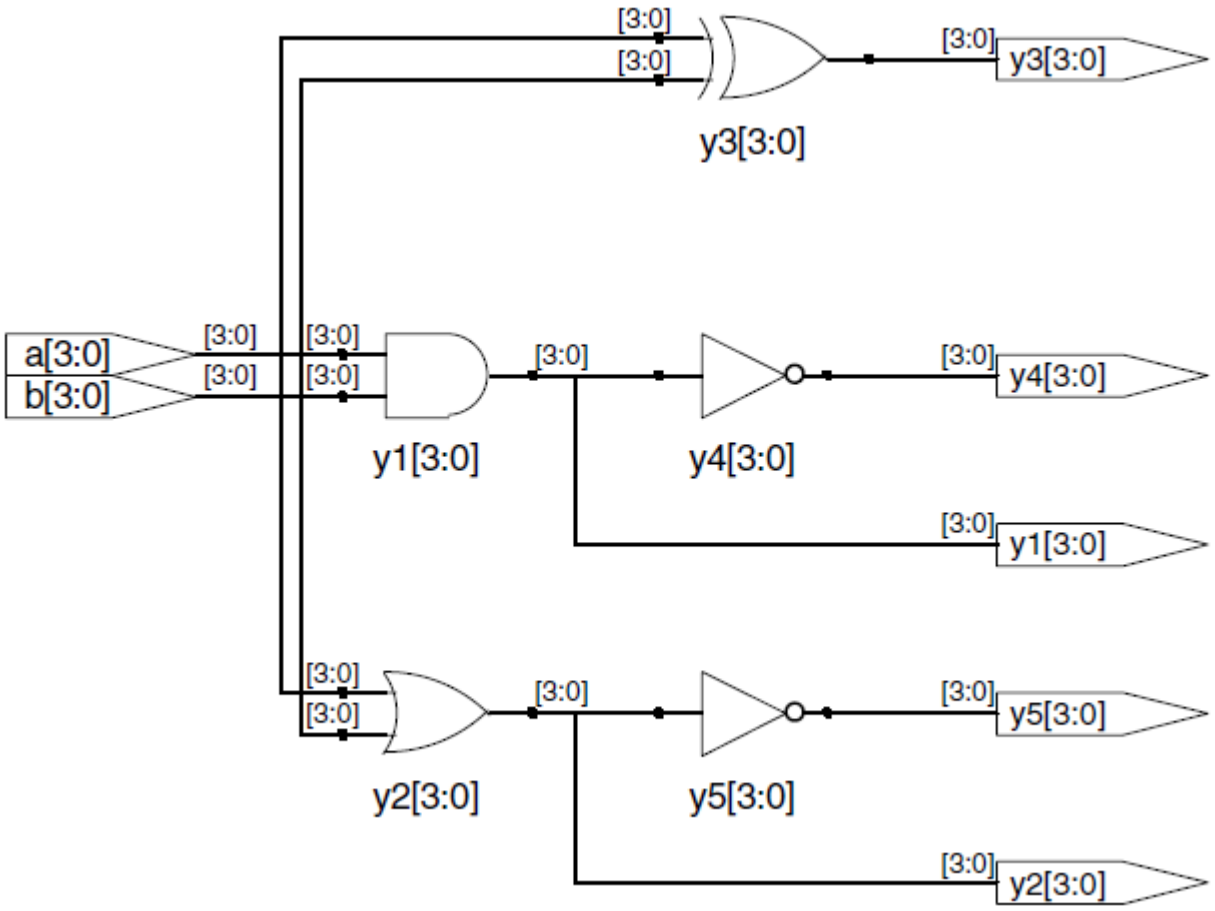
```
module inv (input x,
            output y);
   assign y = ~x;
endmodule
```




```
module and (input a, b,
            output y);
   assign y = a & b;
endmodule
```

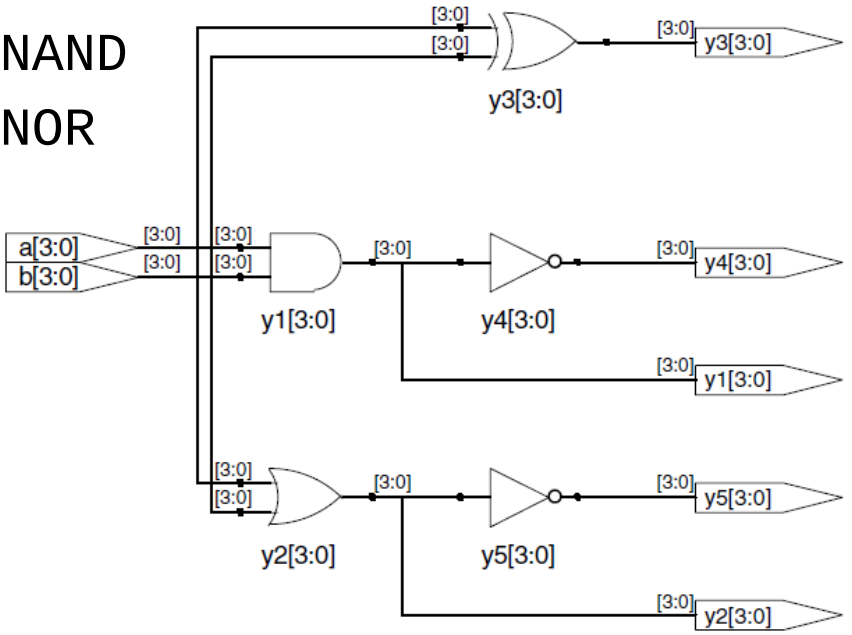


Verilog – example no. 2



Verilog – example – behavioral description

```
module gates (input [3:0] a, b,
              output [3:0] y1, y2,y3, y4, y5);

  assign y1 = a & b;      // AND
  assign y2 = a | b;      // OR
  assign y3 = a ^ b;      // XOR
  assign y4 = ~(a & b);   // NAND
  assign y5 = ~(a | b);   // NOR

endmodule
```



Verilog – simulation – in general

1. “encapsulate” simulated circuit into module without inputs and outputs
2. Declare internal variables of the module (reg, wire) for input presets (reg) and observe of the outputs (wire) of the simulated circuit
3. Assign internal variables to the simulated circuit inputs and outputs
4. specification of time sequence of circuit stimuli

Verilog – simulation – in general

```
`timescale 1ns / 1ps
module testbench_a();
    reg  a, b;   // inputs
    wire  c;     // outputs

    simulated_circui_name  instance_name(
        .a(a),
        .b(b),
        .c(c),
    );

    initial begin
        a = 0;
        b = 0;
    end

    always   #40 a = ~a;
    always   #80 b = ~b;
endmodule
```

no inputs or outputs

create internal, stimuli/monitor

create simulated circuit instance and assignment of testbench internal variables to inputs and outputs

signal names starting by dot are names of inputs and outputs in module declaration, bracketed are names of stimuli and monitor signals

Inputs initialization

always blocks (multiple statements for single allways are enclosed by begin end pair) are executed concurrently…

Invert each 80 ns

Verilog – behavioral description – continue language syntax

Verilog operator precedence

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, –	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	Conditional

Verilog – behavioral description

- Ternar operator „?“

```
module mux2 (input [3:0] d0, d1,
             input s,
             output [3:0] y);
    assign y = s ? d1 : d0;
endmodule
```

- reduce

```
module and5(input [4:0] a, output y);
    assign y = &a;
```

Instead of

```
    assign y = a[4] & a[3] & a[2] & a[1] & a[0];
```

similar for operators: |, ^, ~&, a ~| (or, xor, nand, a nor)

Verilog – behavioral description

- numbers

```
N'BaseVale  // N: bit width; Base: b,o,d,h
b – binry, o – octal, d – decimal, h - hexa
3'b101 // 101
'b11 // when assigned to 6-bit variable 000011
3'd5 // 101
```

- third state value „z“

```
module tristateBuffer (input [3:0] a,
                       input en,
                       output [3:0] y);
    assign y = en ? a : 4'bz;
endmodule
```

Verilog – behavioral description

- invalid value „x“

  „x“ is result of simultaneous assignment of 0 and 1, uninitialized output, when gate cannot determine output value („z“ on input; 1 and „z“ = „x“ but 0 and „z“ = 0)

- Concatenation { }

  y = $c_2c_1d_0d_0d_0c_0101$ is build by :

```
assign y = {c[2:1],{3{d[0]}},c[0],3'b101};
```

  na = ~a; nb = ~b; nc = ~c; can be rewritten as:

```
{na, nb, nb} = ~{a, b, c};
```

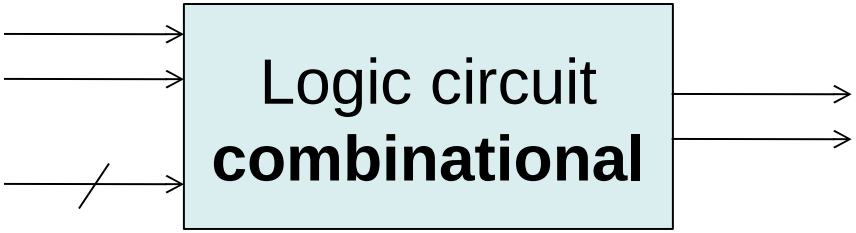
Verilog – behavioral description

- delay

```
‘timescale 1ns/1ps              // unit/resolution

module example (input a, b,
                output y);
    assign #3 y = a & b;        // 3 units = 3 ns
endmodule
```

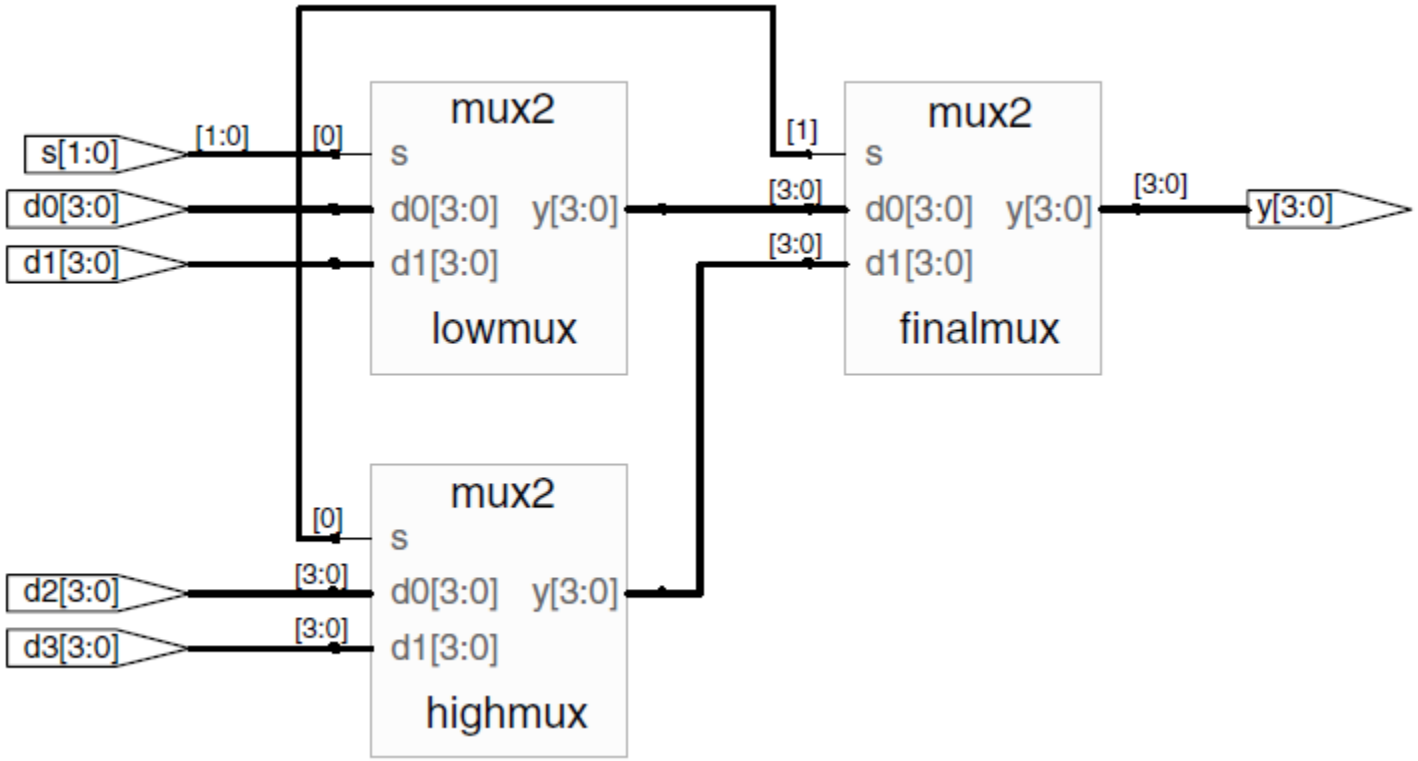
Delays are used only during simulation, they are ignored during synthesis.

Verilog – structural description



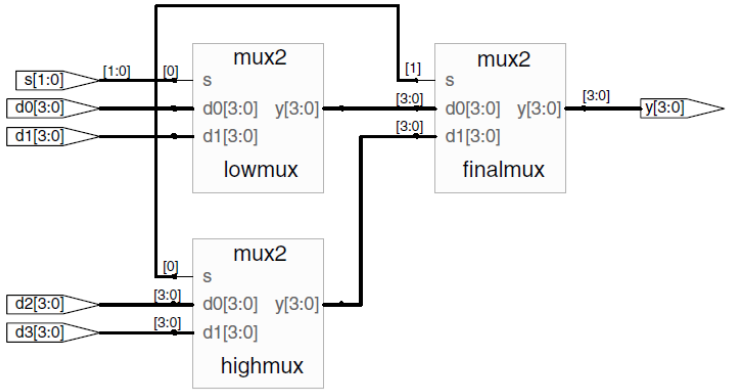

```
module name (input input_list,
             output output_list);

  wire internal_variables_declaration;

  module_name instance_name(port_mapping);
  module_name instance_name(port_mapping);
endmodule
```

Verilog – structural description – example



Verilog – structural description – example

```
module mux4 (input [3:0] d0, d1, d2, d3,
             input [1:0] s,
                 output [3:0] y);

   wire [3:0] low, high;

   mux2 lowmux (d0, d1, s[0], low);
   mux2 highmux (d2, d3, s[0], high);
   mux2 finalmux (low, high, s[1], y);

endmodule
```

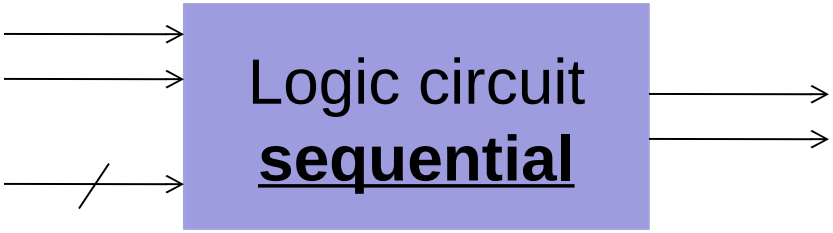


Verilog – structural description

```
module mux2 (input [3:0] d0, d1,
             input s,
             output [3:0] y);

  tristateBuffer t0 (d0, ~s, y); // ~s expression allowed
  tristateBuffer t1 (d1, s, y);

endmodule
```

```
module tristateBuffer (input [3:0] a,
                input en,
                output [3:0] y);
    assign y = en ? a : 4'bz;
endmodule
```

Verilog



- Output value binding by `assign` statement is evaluated for each change of expression input on the right side
- Otput value controlled by `always @()` is evaluated only for specified conditions (left side of the statement/expression holds its value even when inputs of the right side change value)
- Blocking assignment are evaluated „=“ sequentionally as they are introduced in the source
- Nonblocking assignments „<=“ are evaluated simultaneously; left sides of all assignments are updated after evaluation of right sides

Verilog – always

- Suggestions:
  Use blocking assignemnts for combinational circuits, non-blocking for sequential

- Output assignment by `always` directives:

```
always @ (sensitivity_list)
   statement;
```

  - `statement` is evalueted only occurs event which is enumerated in `sensitivity_list`
  - If `always @(*)` is specified then it is trigerred by any change on inputs of all enclosed statements
  - All signals on the left side of assignemnt (=, <=) has to be declared as `reg`

Verilog – blocking vs. non blocking assignments

```
module circuit (input a, b,
                output reg y);
reg x;
always @(*)
  begin
    x = a ^ b; //blocking
    y = x & a; //blocking
  end
endmodule
```

```
module circuit (input a, b,
                output reg y);
reg x;
always @(*)
  begin
    x <= a ^ b; //non-blocking
    y <= x & a; //non-blocking
  end
endmodule
```

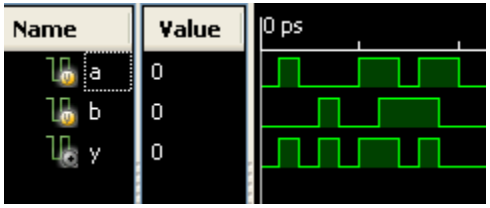
Assume that a=b=0. Then y=0. a is assigned to 1 (a=1) at some time instant, which triggers always block evaluation. Blocking statements are executed in order of their appearance, that is x is set to new value before y is evaluated. When non-blocking assignments are used then all right sides are evaluated simultaneously and then right sides are assigned as separated step. The consequence is that y stays equal to 0. But change of x from 0 to 1 results in repeated triggering of always block, and following expected evaluation of y. But if sensitivity list is changed to: always @(a,b) instead of @(*) then the behavior of the two circuits differs

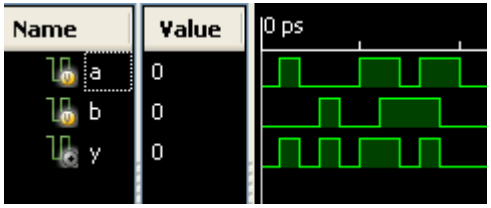
Verilog

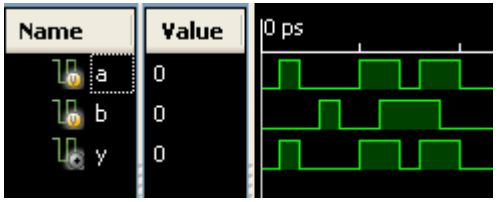
```
module abc(
    input a,
    input b,
    output reg y
    );

  always @(*)
    y = a;

  always @(*)
    y = b;

endmodule
```

```
module abc(
    input a,
    input b,
    output reg y
    );

  always @(*)
    y <= a;

  always @(*)
    y <= b;

endmodule
```

```
module abc(
    input a,
    input b,
    output reg y
    );

  always @(*)
    assign y = a;

  always @(*)
    y = b;

endmodule
```







Verilog

Variables declarations as:

- `reg x;` means, that variable is a single bit signal which is used ob the left side in `always`
- `reg [7:0] x;` declares 8-bit variable…
- `reg x[7:0];` declares `x` as vector of eight one bit variables which can be accessed by index (x[0], x[1],…)
- `reg [7:0] x[15:0];` declares 16 elements vector of 8-bit variables …

Verilog – always

D flip-flop triggered by rising edge of clk (positive edge):

```
module D_flip_flop (input clk,
                    input [3:0] d,
                    output reg [3:0] q);
   always @ (posedge clk)
      q <= d;
endmodule
```

D flip-flop with asynchronous reset:

```
module D_flip_flop_r (input clk, reset,
                      input [3:0] d,
                      output reg [3:0] q);
   always @ (posedge clk, posedge reset)
      if (reset) q <= 4'b0;
      else q <= d;
endmodule
```

Verilog – always

When `sensitivity_list` reacts/triggers on change of all inputs and output value is assigned for all combinations on inputs then `always @ ()` is can be used to describe combinational circuit description. example:

```
module comb (input a, b
             output reg y);
   always @ (a,b)
      y  = ~a & b;
endmodule
```

It is better to use “*”:

```
module comb (input a, b
             output reg y);
   always @ (*)          // <--
      y  = ~a & b;
endmodule
```

Verilog – if-else

`if, if-else, case, casez` has to be used inside `always`. Examples:

```
module D_flip_flop_en_r (input clk, reset, enable,
                         input [3:0] d,
                         output reg [3:0] q);
   always @ (posedge clk, posedge reset)
      if (reset) q <= 4'b0;
      else if (en) q <= d;
endmodule
```

Verilog – case

```
module seven_segment_display_decoder (input [3:0] data,
                      output reg [6:0] segments);
   always @ (*)
       case (data)
         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_1011;
         default: segments  7'b000_0000;
       endcase
endmodule
```

Verilog – casez

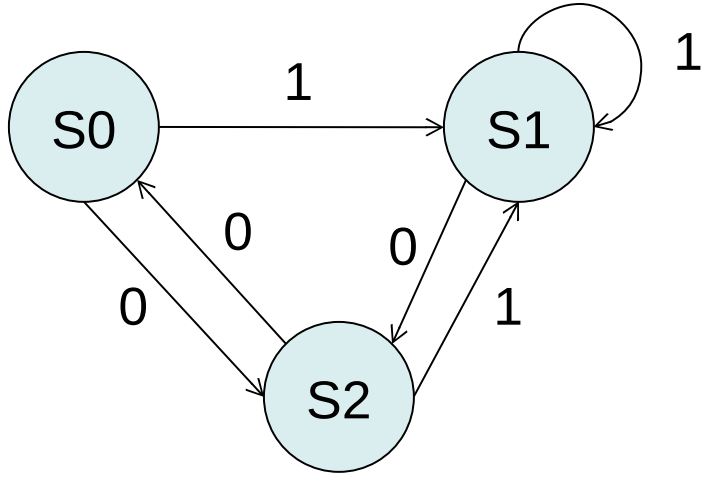
casez: the “?” placeholder matches 0 or 1, bit/place value is ignored.

```
module priority_circuit(input [3:0] a,
                output reg [3:0] y);
      always @ (*)
         casez (a)
           4’b1???: y = 4’b1000;
           4’b01??: y = 4’b0100;
           4’b001?: y = 4’b0010;
           4’b0001: y = 4’b0001;
           default: y = 4’b0000;
         endcase
endmodule
```

Verilog – Mealy/Moore finite state machine

```
module FSM (input x, clk, reset,
            output y);
   reg [1:0] state, nextstate;
   parameter S0 = 2'b00;    // parameter – constant definition
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;

   always @ (posedge clk, posedge reset)
       if (reset) state <= S0;
       else state <= nextstate;

   always @ (*)
       case (state)           // next state logic
           S0: if (x) nextstate = S1;
               else   nextstate = S0;
           S1: nextstate = S2;
           S2: nextstate = S0;
           default: nextstate = S0;
       endcase
   assign y = (x & state == S0 ) ? 1'b1 : 1'b0;   // output logic
endmodule
```

Verilog – **another** Moore finite state machine



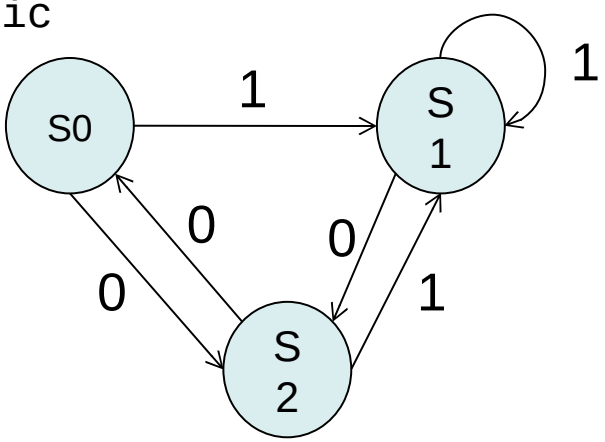
Outputs table

State	Output y
S0	0010
S1	0110
S2	1010

```
module FSM2 (input x, clk, reset,
             output reg [3:0] y);

   reg [1:0] state, nextstate;
   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;

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

Verilog – Moore finite state machine

```
always @(*)
    case (state)            // next state logic
        S0: nextstate = x ? S1 : S2;
        S1: nextstate = x ? S1 : S2;
        S2: nextstate = x ? S1 : S0;
        default: nextstate = S0; //!!
    endcase
```




```
always @(*)
    case (state)            // output logic
        S0: y = 4'b0010;
        S1: y = 4'b0110;
        S2: y = 4'b1010;
        default: y = 4'b0010;  //!!
    endcase

endmodule
```

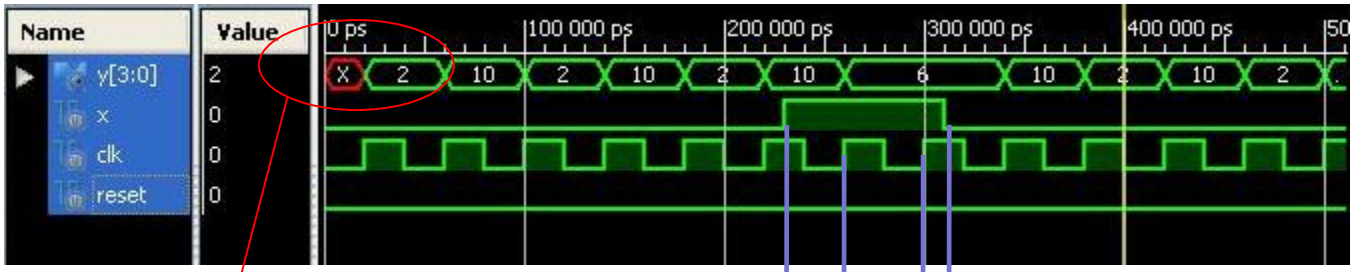
State	Output y
S0	0010
S1	0110
S2	1010

Verilog – Moore finite state machine - Simulation/testbed

```
`timescale 1ns / 1ps

module test_moore;
   reg x;          // Inputs
   reg clk;
   reg reset;
   wire [3:0] y;  // Output

   FSM2 uut (
       .x(x),
       .clk(clk),
       .reset(reset),
       .y(y)
   );
```

```
   initial begin
       x = 0;
       clk = 0;
       reset = 0;

       #230 x = 1;
       #80 x = 0;
       #100;
     end

     always #20 clk = ~clk;

endmodule
```

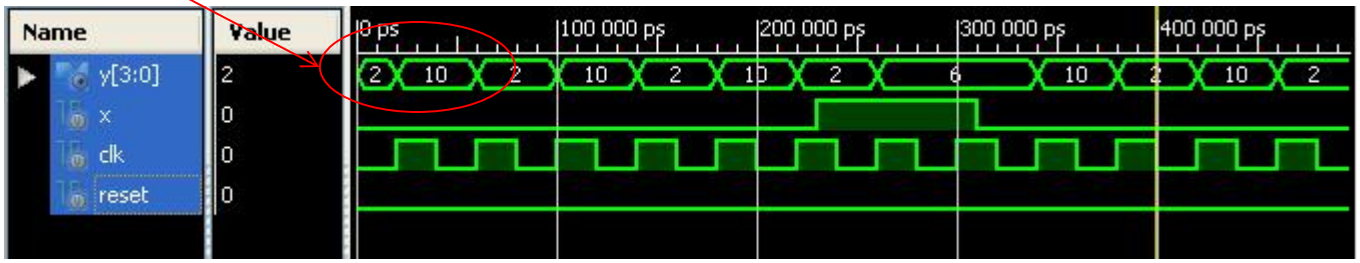
Verilog – Moore finite state machine




```
module FSM3 (input x, clk, reset,
             output reg [3:0] y);

   reg [1:0] state, nextstate;
   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;

   initial  state <= S0;
```

Setup and hold !!

Only difference between modules FSM3 and FSM2 is then next line...



Verilog – parameterized modules

(parameter …) exactly after module name

```
module mux2
        # (parameter width = 8)
        (input [width-1:0] d0, d1,
         input s,
         output [width-1:0] y);
  assign y = s ? d1 : d0;
endmodule
```

Verilog – parameterized modules

It is possible to write:

```
module mux4_12(input [11:0] d0, d1, d2, d3,
               input [1:0] s,
               output [11:0] y);

  wire [7:0] low, hi;

  mux2 #(12) lowmux(d0,d1,s[0],low); //change 8 to 12
  mux2 himux (d2,d3,s[1],hi); //original width=8, error
  mux2 #(12) outmux (low, hi, s[1], y);
endmodule
```

It is better to create parameterized module…

Warning: #(…) specifies parameter value; #... is delay.

Verilog – final notes

- Verilog is a hardware description. When designing the hardware, the “top-down” principle is used, in which individual subsystems are gradually refined, and the functionality of the system as a whole can be simulated at the design stage without specific specification of sub-subsystems at the lowest level. The same HDL code describing the behavioral behavior of a circuit can be implemented differently depending on the synthesizer tool used, its setup and the degree of optimization. For example:

```
module adder (input [7:0] a, b,
              output [7:0] y);
    assign y = a + b;  // how is addition implementd?
endmodule
```

Quiz question no. 1 – What is wrong???

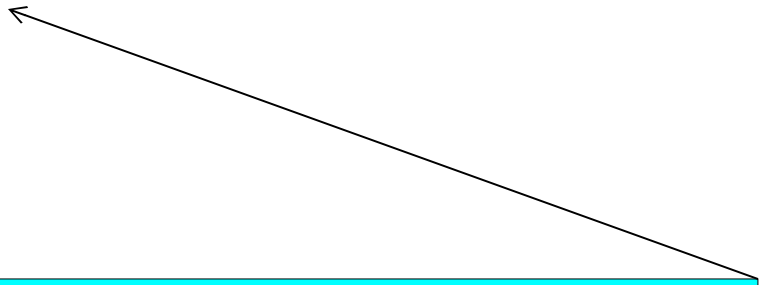
D flip-flop triggered by rising edge of clk (positive edge):

```
module D_flip_flop (input clk,
                    input d,
                    output q);
      always @ (posedge clk)
         q <= d;
   endmodule
```

Quiz question no. 1 – What is wrong???

D flip-flop triggered by rising edge of clk (positive edge):

```
module D_flip_flop (input clk,
                    input d,
                    output q);
      always @ (posedge clk)
          q <= d;
   endmodule
```



All signals (used in always clause) on the left side of assignment (=, <=) has to be declared as `reg`

Quiz question no. 2 – What is wrong???

Multiplexor

```
module mux2 (input select, d0, d1,
             output reg y);
      always @ (select)
        if(select) y <= d1;
        else y <= d0;
endmodule
```

Quiz question no. 2 – What is wrong???

Multiplexor

```
module mux2 (input select, d0, d1,
             output reg y);
      always @ (select)
        if(select) y <= d1;
        else y <= d0;
endmodule
```

y values is not changed when only data inputs are changed. Solution:
always @(*)
or ternar operator: `assign y = select ? d1 : d0;` and do not use always

Verilog

Literatura:

- Harris D.M. – Harris S.L.: Digital design and computer architecture, ed. Morgan–Kaufmann, 2007, chapter 4: Hardware description languages

Links:

- http://www.aldec.com/Products/Product.aspx?productid=0380ca74-7c15-4a01-b727-2f7caab53730
- http://vol.verilog.com/
- http://www.verilogtutorial.info/
- http://courses.cit.cornell.edu/ece576/Verilog/LatticeTestbenchPrimer.pdf