DIGITAL DESIGN

LAB3　GATES, BITWISE OPERATION & PRIMITIVE IN VERILOG, STRUCTURED DESIGN

2020 FALL TERM

LAB3

- Gates and Primitives in verilog
- bitwise and logic operations in verilog
- Two ways to do the design
  - 1. data flow
  - 2. structured design

BITWISE AND LOGICAL OPERATIONS IN VERILOG

Four-valued logic (The IEEE 1364 standard): 0, 1, Z (high impedance), and X (unknown logic value).

Operator : ~ & ^ ~^ ^~ | ! && ||

Priority:
~ ! > & > ^ ~^ ^~ > | > && > ||

```
assign  q = a & b & c ;
assign  z = x | y;
assign  t = a & b | ~c;
```

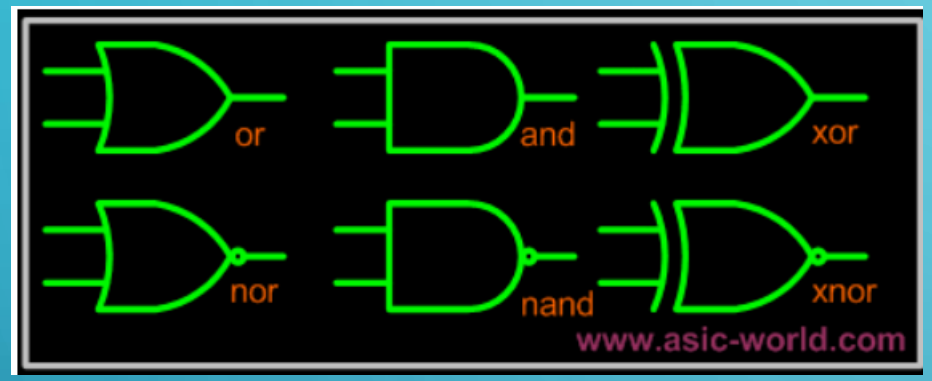
Operator type	Operator symbols	Operation performed
Bitwise	~	Bitwise NOT (1's complement)
	&	Bitwise AND
	|	Bitwise OR
	^	Bitwise XOR
	~^ or ^~	Bitwise XNOR
Logical	!	NOT
	&&	AND
	||	OR

PRIMITIVE

- Verilog has built in primitives like gates, transmission gates, and switches. These are rarely used in design (RTL Coding), but are used in post synthesis world for modeling the ASIC/FPGA cells; these cells are then used for gate level simulation, or what is called as SDF simulation. Also the output netlist format from the synthesis tool, which is imported into the place and route tool, is also in Verilog gate level primitives.
- Note : RTL engineers still may use gate level primitives or ASIC library cells in RTL when using IO CELLS, Cross domain synch cells.

http://www.asic-world.com/verilog/index.html

PRIMITIVE GATE



Gate	Description
and	N-input AND gate
nand	N-input NAND gate
or	N-input OR gate
nor	N-input NOR gate
xor	N-input XOR gate
xnor	N-input XNOR gate

```
module gates();

wire out0;
wire out1;
wire out2;
reg  in1,in2,in3,in4;

not U1(out0,in1);
and U2(out1,in1,in2,in3,in4);
xor U3(out2,in1,in2,in3);

initial begin
  $monitor(
  "in1=%b in2=%b in3=%b in4=%b out0=%b out1=%b out2=%b",
   in1,in2,in3,in4,out0,out1,out2);
  in1 = 0;
  in2 = 0;
  in3 = 0;
  in4 = 0;
   #1  in1 = 1;
   #1  in2 = 1;
   #1  in3 = 1;
   #1  in4 = 1;
   #1  $finish;
end

endmodule
```

http://www.asic-world.com/verilog/index.html

TWO WAYS TO DO THE DESIGN

- Data flow design: using "assign " as *continuous assignment* , to transfer the data from input ports through variables to the output ports .

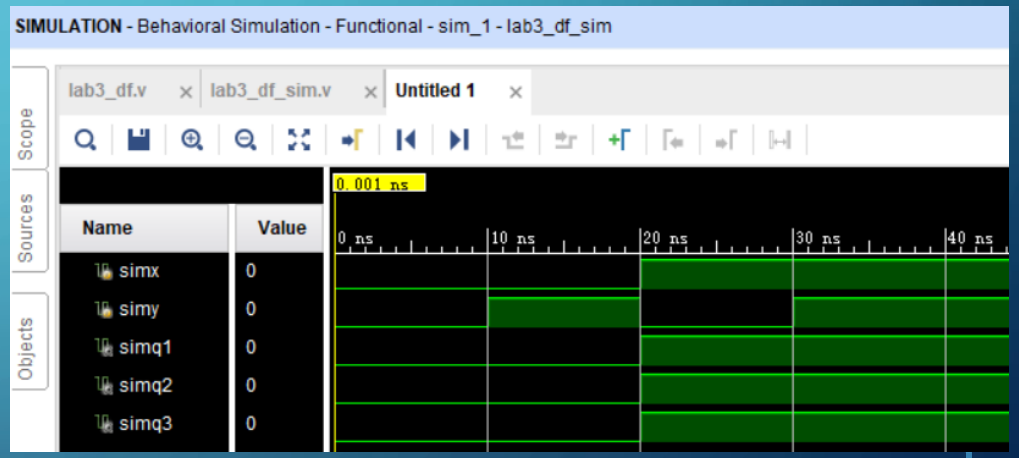
- Structured design: using verilog to define the relationship between modules. It is based on the module/IP.

DATA FLOW DESIGN

Demo: a) x | (x&y) b) x & (x | y)

```
module lab3_df(
    input x,
    input y,
    output q1,
    output q2,
    output q3
    );
    assign q1 = x;
    assign q2 = x | (x & y);
    assign q3 = x & (x | y);

endmodule
```

```
module lab3_df_sim( );
    reg simx,simy;
    wire simq1,simq2,simq3;
    lab3_df u_df(
    .x(simx), .y(simy), .q1(simq1), .q2(simq2), .q3(simq3) );

    initial
    begin
        simx=0;
        simy=0;
     #10
        simx=0;
        simy=1;
     #10
        simx=1;
        simy=0;
     #10
        simx=1;
        simy=1;
    end
endmodule
```

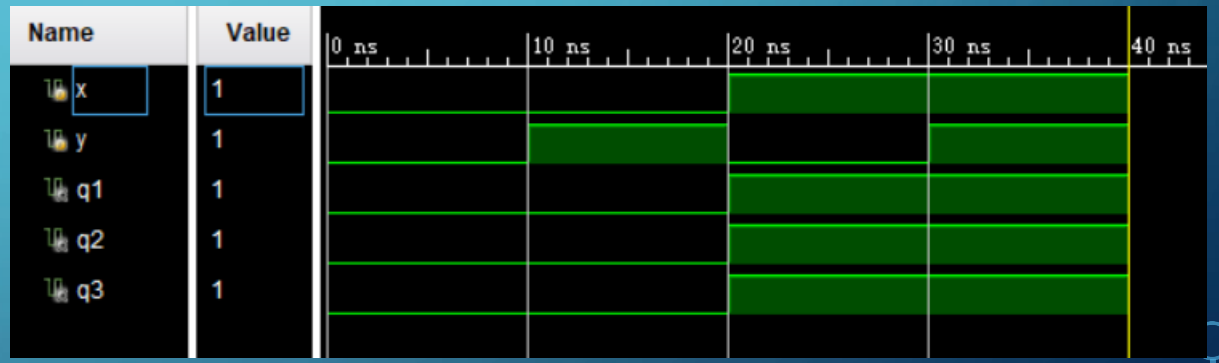


STRUCTURE DESIGN

Demo: a) x | (x&y) b) x & (x | y)

```
module lab3_demo_sd(x,y,q1,q2,q3);
input x,y;
output q1,q2,q3;
wire o_xandy,o_xory;

assign q1=x;

and and1(o_xandy,x,y);
or or1(o_xory,x,y);

or or2(q2,o_xandy,x);
and and2(q3,o_xory,x);

endmodule
```

```
module lab3_demo_sd_tb( );
reg x,y;
wire q1,q2,q3;
lab3_demo_sd usd(x,y,q1,q2,q3);

initial
begin
          {x,y}=2'b00;
    #10   {x,y}=2'b01;
    #10   {x,y}=2'b10;
    #10   {x,y}=2'b11;
    #10   $finish(0);
end
endmodule
```



PRACTICES

- 1. Do the design (both x and y are 1 bit width) using two ways respectively:

  1) structure design with primitive gates 2)data flow

  a) ~(x | y) b) ~x & ~y c) ~(x & y) d) ~x | ~y

- 2. create testbench ,do simulation to verify function of the design
- 3. generate bitstream file, test on the board
- 4. think about : if the bitwise of x and y is more than 1 bit
  - 1) can the primitive gates be used in question 1?
  - 2) will the result of bit-width calculation be same as which on 1 bit width input