UCI University of California, Irvine

EECS 31L: INTRODUCTION TO DIGITAL DESIGN LAB LECTURE 5

Salma Elmalaki
salma.elmalaki@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Some slides are compiled from JL Gaudiot and P. Yaghini

LECTURE 5: LOGISTICS

- **All questions should be posted on canvas**
- Lab 3 is assigned —> You need to start early
- Lab 2 is currently being graded

LECTURE 5: LAST TIME

- Behavioral Description
- Sequential statements
  - if, case, for, while, …
  - Storage elements
  - DFF
  - JK FF
  - Binary counters

LECTURE 5: OVERVIEW

- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm

LECTURE 5: OVERVIEW

- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm

BLOCKING VS. NON-BLOCKING ASSIGNMENT

Inside sequential block (like always or initial)

- **blocking**
  - Use **=** operator
  - The result of this assignment can be seen in the subsequent statements
  - Statements are executed sequentially instead of concurrently
- **non-blocking**
  - Use **<=** operator
  - The result of this assignment cannot be seen in subsequent statements
  - Statements are executed concurrently instead of sequentially
  - The order of non-blocking assignments is not important as long as the destinations of the assignments are different signals

BLOCKING VS NON-BLOCKING

Blocking assignment: evaluation and assignment are immediate

```
always @ (a or b or c)
begin
  x = a | b;          1. Evaluate a | b, assign result to x
  y = a ^ b ^ c;      2. Evaluate a^b^c, assign result to y
  z = b & ~c;         3. Evaluate b&(~c), assign result to z
end
```

Non-Blocking assignment: all assignments are deferred until all right-hand sides have been evaluated (end of simulation timestep)

```
always @ (a or b or c)
begin
  x <= a | b;         1. Evaluate a | b but defer assignment of x
  y <= a ^ b ^ c;     2. Evaluate a^b^c  but defer assignment of y
  z <= b & ~c;        3. Evaluate b&(~c) but defer assignment of z
end                   4. Assign x, y, and z with their new values
```

- Sometimes, as above, both produce the same result. Sometimes, not!

EXAMPLE; BLOCKING VS. NON-BLOCKING ASSIGNMENT

```
module block_nonblock();
reg a, b, c, d , e, f ;

 // Blocking assignments
initial begin
 #10 a = 1'b1;
 #20 b = 1'b0;
 #40 c = 1'b1;
end

/* The simulator assigns
1 to a at time 10, 0 to
b at 30, and 1 to c at
70
*/
```

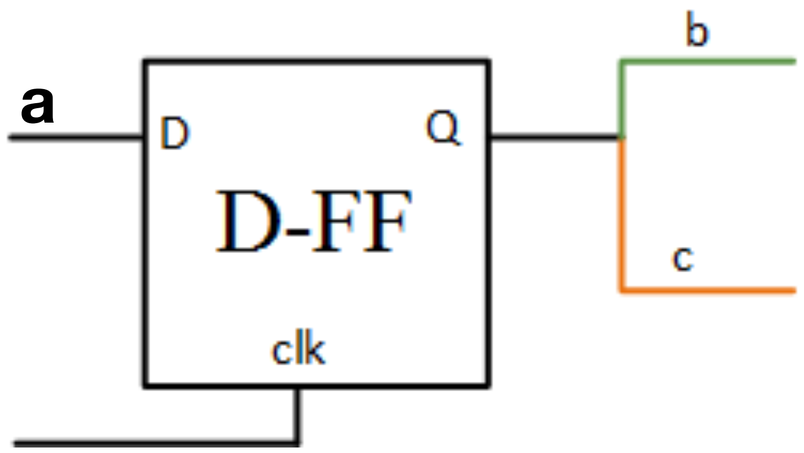
```
// Non-blocking
assignments
initial begin
 #10  d <=  1'b1;
 #20  e <=  1'b0;
 #40  f <=  1'b1;
end
endmodule

/* The simulator
assigns 1 to d at time
10, 0 to e at 20, and 1
to f at 40 */
```

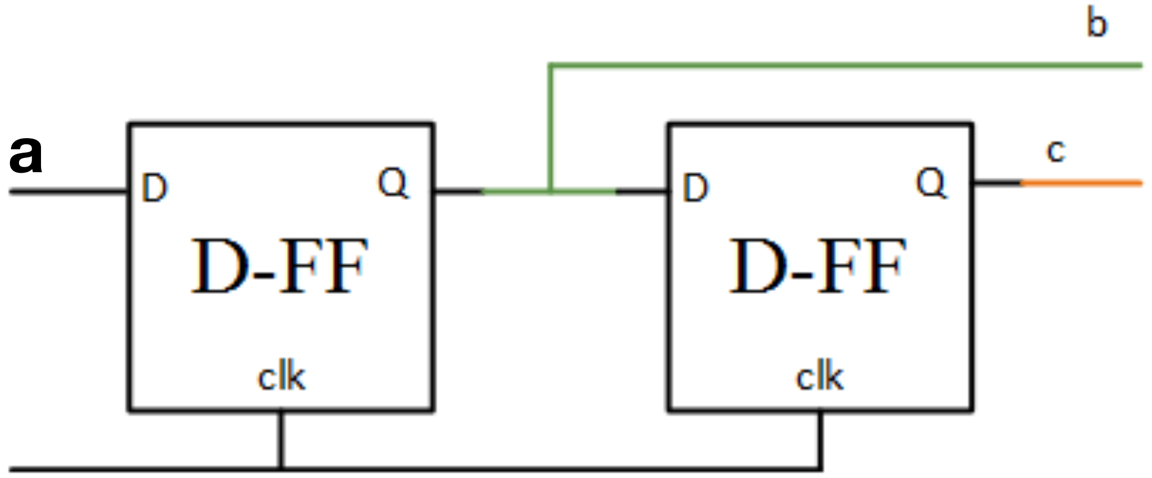
UCI University of California, Irvine

EXAMPLE; BLOCKING VS. NON-BLOCKING ASSIGNMENT

```
module blocking (clk,a,c);
  input clk;
  input a;
  output c;
  wire clk,a;
  reg c,b;
always @ (posedge clk )
 begin
   b = a;
   c = b;
 end
endmodule
```

```
module non-blocking (clk,a,c);
  input clk;
  input a;
  output c;
  wire clk,a;
  reg c,b;
always @ (posedge clk )
 begin
   b <= a;
   c <= b;
 end
endmodule
```





LECTURE 5: OVERVIEW

- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm

STRUCTURAL DESCRIPTION

- Best option when
  - Hardware details of the design are known
  - Schematic solution is available
- Structural description simulates the system by describing its logical components.
  - Components can be gate level(such as AND gates, OR gates, or NOT gates)
  - Components can be in a higher logical level, such as Register- Transfer Level (RTL) or processor level.
- Verilog recognizes all the primitive gates, such as AND, OR, XOR, NOT, and XNOR gates.
- Basic VHDL packages do not recognize any gates unless the package is linked to one or more libraries.

STRUCTURAL DESCRIPTION

Verilog Structural Description

```
module system(a,b,sum,cout);

    input a,b;

   output sum, cout;

   xor X1(sum, a, b);  // (outputs, inputs)   /* X1 is an optional identifier; it can be omitted.*/

   and a1(cout, a, b);   /* a1 is optional identifier; it can be omitted.*/

endmodule
```

What if you want to build your primitive cell?

BINDING AND INSTANTIATION

```
module two (s1, s2, a1, b1);
    input a1;
    input  b1;
    output  s1, s2;
    xor (s1, a1, b1);
    and (s2, a1, b1);
endmodule
```

BINDING AND INSTANTIATION

Binding between two Modules in Verilog

```
module one( O1,O2,a,b);
    input [1:0] a;
    input [1:0] b;
    output [1:0] O1, O2;

two M0(O1[0], O2[0], a[0], b[0]);
two M1(O1[1], O2[1], a[1], b[1]);
endmodule

module two (s1, s2, a1, b1);
    input a1;
    input  b1;
    output  s1, s2;
    xor (s1, a1, b1);
    and (s2, a1, b1);
endmodule
```

Binds module two to module one

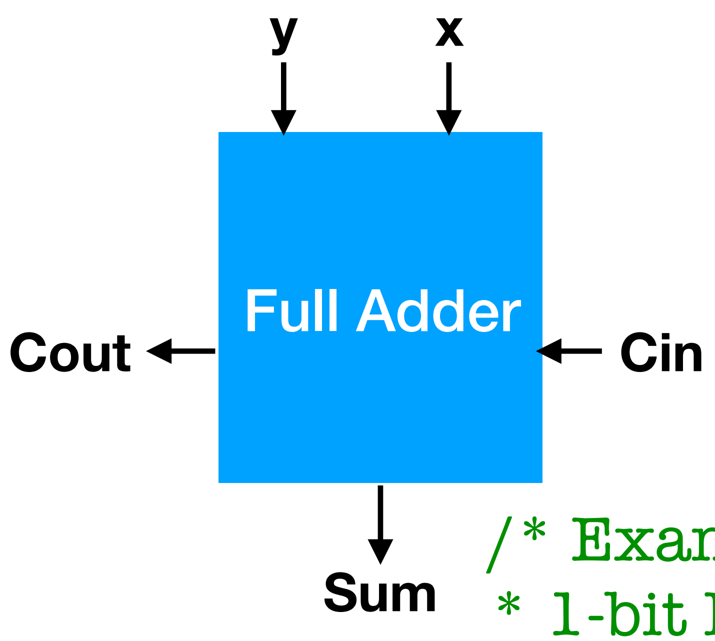
The relationship as follows:

- `O1[1]` is the output of a two-input XOR gate with `a[0]` and `b[0]` as the inputs.
- `O2[1]` is the output of a two-input AND gate with `a[1]` and `b[1]` as the inputs.

Binds module two to module one

- Port Association
  - Implicit by order (Position Mapping)

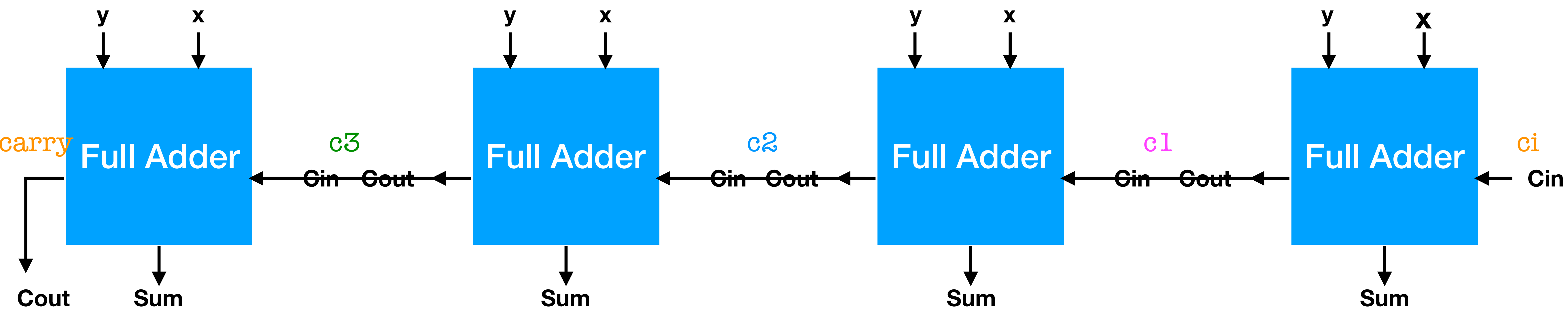
MODULE INSTANTIATION



x	y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	$\boxed{1}$	0
0	1	0	$\boxed{1}$	0
0	1	1	0	1
1	0	0	$\boxed{1}$	0
1	0	1	0	1
1	1	0	0	1
1	1	1	$\boxed{1}$	1

```
/* Example
 * 1-bit FA module code
 */
module fulladder (input x, input y, input cin,
                  output sum, output cout);
wire m1, m2, m4, m7;

assign m1 = (~x & ~y & cin);
assign m2 = (~x & y & ~cin);
assign m4 = (x & ~y & ~cin);
assign m7 = (x & y & cin);
assign sum = m1 | m2 | m4 | m7;
assign cout = (x & y) | (cin & (x | y));
endmodule;
```

PORT INSTANTIATION



- Port Association
  - Implicit by order (Position Mapping)
  - Explicit by name (Nominal Mapping)

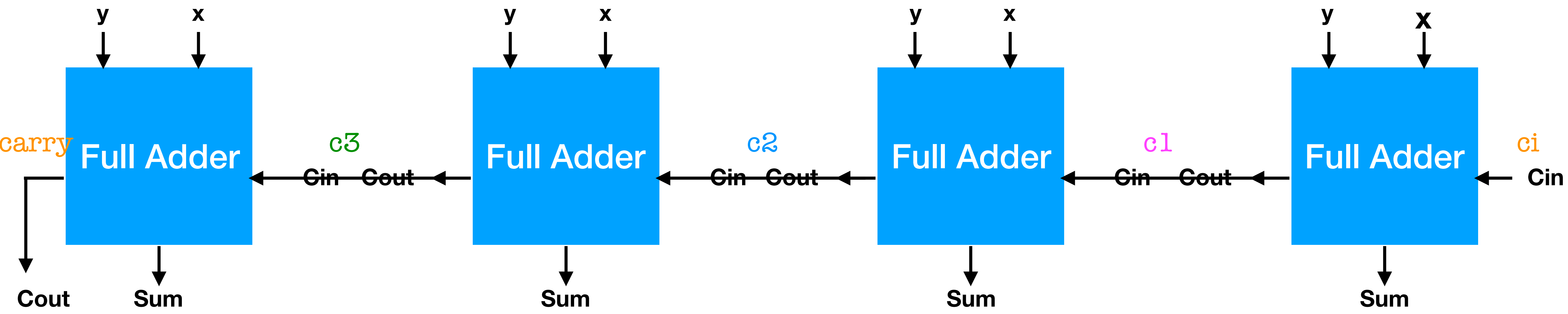
```
/* 1-bit FA module code */
module fulladder (input x, input y, input cin,
                        output sum, output cout);
wire m1, m2, m4, m7;

assign m1 = (~x & ~y & cin);
assign m2 = (~x & y & ~cin);
assign m4 = (x & ~y & ~cin);
assign m7 = (x & y & cin);
assign sum = m1 | m2 | m4 | m7;
assign cout = (x & y) | (cin & (x | y));
endmodule;
```

```
/* Implicit by Order */
module fa_4bit (result, carry, r1, r2, cin);
  // input port declarations
   input [3:0] r1 ;
   input [3:0] r2 ;
   input ci ;
  // Output Port Declarations
   output [3:0] result ;
   output carry ;
   wire c1, c2, c3;
   fulladder u0 ( r1[0], r2[0], ci, result[0], c1 );
   fulladder u1 ( r1[1], r2[1], c1, result[1], c2);
   fulladder u2 ( r1[2], r2[2], c2, result[2], c3 );
   fulladder u3 ( r1[3], r2[3], c3, result[3], carry );
endmodule;
```

The order of the ports match the instantiated module

UCI University of California, Irvine

PORT INSTANTIATION



- Port Association
  - Implicit by order (Position Mapping)
  - Explicit by name (Nominal Mapping)

```
/* 1-bit FA module code */
module fulladder (input x, input y, input cin,
                              output sum, output cout);
wire m1, m2, m4, m7;

assign m1 = (~x & ~y & cin);
assign m2 = (~x & y & ~cin);
assign m4 = (x & ~y & ~cin);
assign m7 = (x & y & cin);
assign sum = m1 | m2 | m4 | m7;
assign cout = (x & y) | (cin & (x | y));
endmodule;
```

```
/* Explicit by name */
fulladder u0 (
  .x (r1[0]) ,
  .y (r2[0]) ,
  .cin   (ci),
  .sum  (result[0]) ,
  .cout  (c1) );
fulladder u1 (
  .x (r1[1]) ,
  .y (r2[1]) ,
  .cin  (c1) ,
  .sum (result[1]) ,
  .cout  (c2) );
fulladder u2 (
  .x (r1[2]) ,
  .y (r2[2]) ,
  .cin (c2) ,
  .sum (result[2]),
  .cout  (c3) );
fulladder u3 (
  .x (r1[3]) ,
  .y (r2[3]) ,
  .cin (c3),
  .sum (result[3]),
  .cout  (carry) );
```

INSTANTIATION EXAMPLE OF GENERIC AND-GATE

Suppose you want to write a module to do this operation
c = &a

`Verilog Generic and-gate`

```
module and_gate (a,c);

input [4:0] a; output c;

wire [5:0] tmp;
assign tmp[0]=1’b1;
assign tmp[1] = tmp[0] & a[0];
assign tmp[2] = tmp[1] & a[1];
assign tmp[3] = tmp[2] & a[2];
assign tmp[4] = tmp[3] & a[3];
assign tmp[5] = tmp[4] & a[4];
assign c = (tmp[5]==1’b1)?1’b1:1'b0;
endmodule
```

Do you see a pattern?

What if you want to design a generic module with arbitrary size of input a?

INSTANTIATION EXAMPLE OF GENERIC AND-GATE

Verilog Generic and-gate

```
module and_gate (a,c);
parameter N=2;
input [N-1:0] a; output c;

wire [N:0] tmp;
assign tmp[0]=1'b1;

genvar i;
generate
for (i=0;i<N;i=i+1)
 begin
  assign tmp[i+1]=tmp[i]&a[i];
 end
endgenerate
assign c = (tmp[N]==1'b1)?1'b1:1'b0;
endmodule
```

Verilog and-gate instantiation

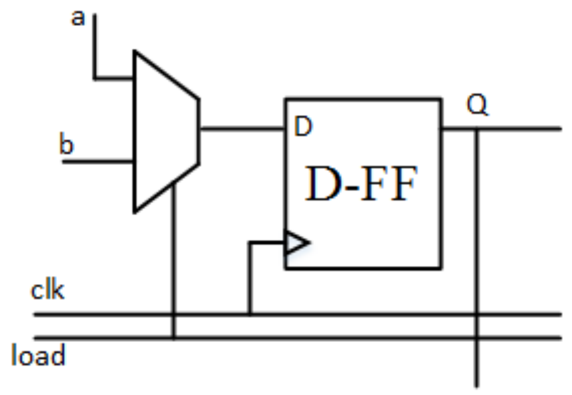
```
module testbench();
parameter N=4;
reg [N-1:0] in_tb;
wire c_tb;

and_gate #(N) instant (.a(in_tb),
                      .c(c_tb)
                      );


initial
begin
  in_tb=&1'b1;
end

always
begin
//rest of test bench
```

LECTURE 5: OVERVIEW

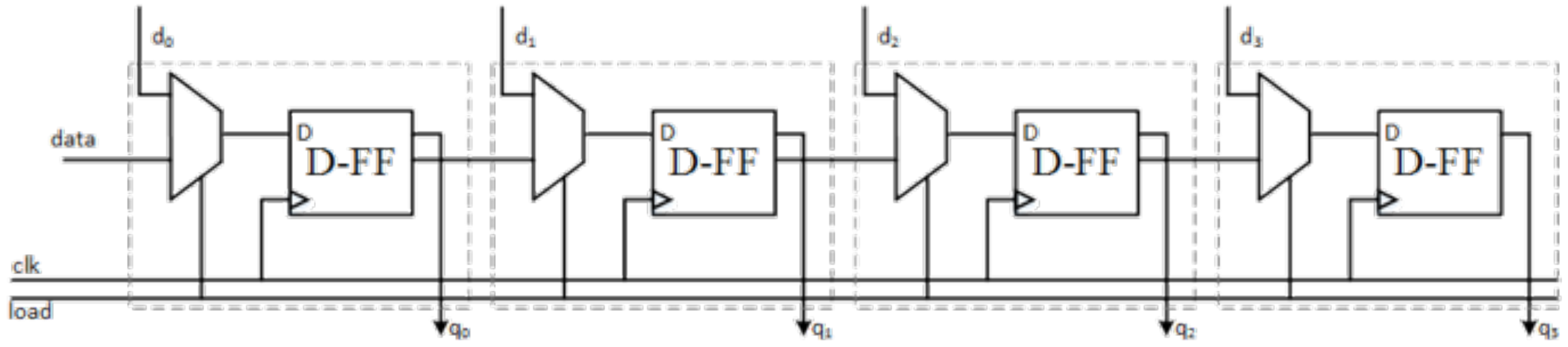
- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm

UCI University of California, Irvine

INSTANTIATION EXAMPLE OF SHIFT REGISTER WITH LOAD



Four bit shift register

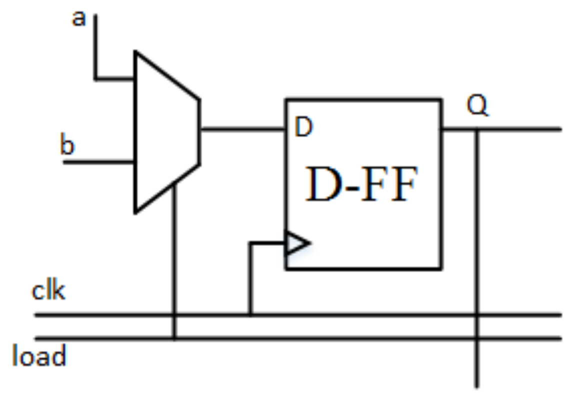


UCI University of California, Irvine

INSTANTIATION EXAMPLE OF SHIFT REGISTER WITH LOAD

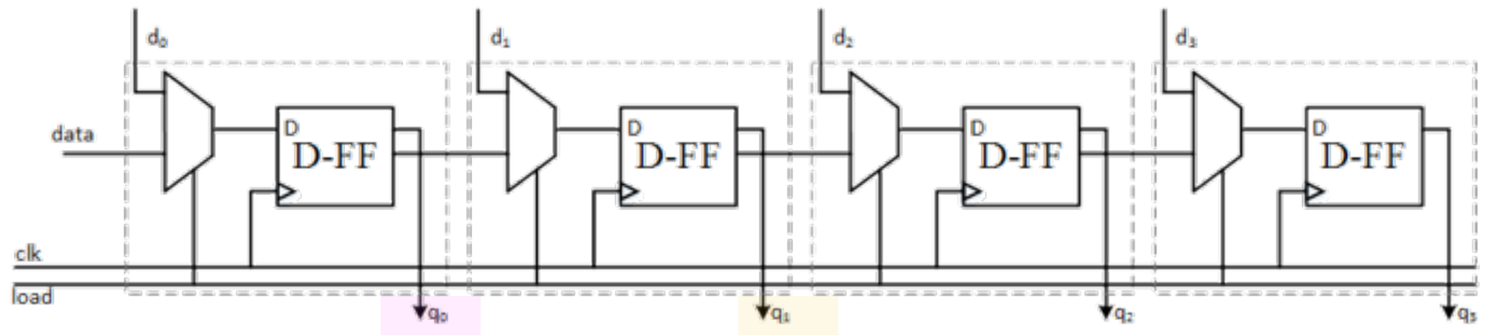
Verilog standard cell

```
module shift_reg_cell (a,b,load,clk,q);

input  a,b,load,clk;
output reg q;

always @ (posedge clk)
begin
if (load==1'b1)
  q=a;
else
  q=b;
end
endmodule
```



Verilog shift register

```
module shift_reg (data,clk,load,d,q);
input  data,clk,load;
input [0:3] d;
output [0:3] q;
```



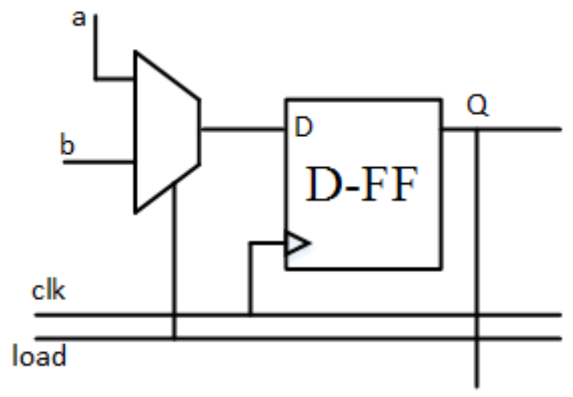
```
shift_reg_cell DFF0(.a(d[0]),.b(data),.load(load),.clk(clk),.q(q[0]));
shift_reg_cell DFF1(.a(d[1]),.b(q[0]),.load(load),.clk(clk),.q(q[1]));
shift_reg_cell DFF2(.a(d[2]),.b(q[1]),.load(load),.clk(clk),.q(q[2]));
shift_reg_cell DFF3(.a(d[3]),.b(q[2]),.load(load),.clk(clk),.q(q[3]));
endmodule
```

UCI University of California, Irvine

INSTANTIATION EXAMPLE OF SHIFT REGISTER WITH LOAD (GENERATE)

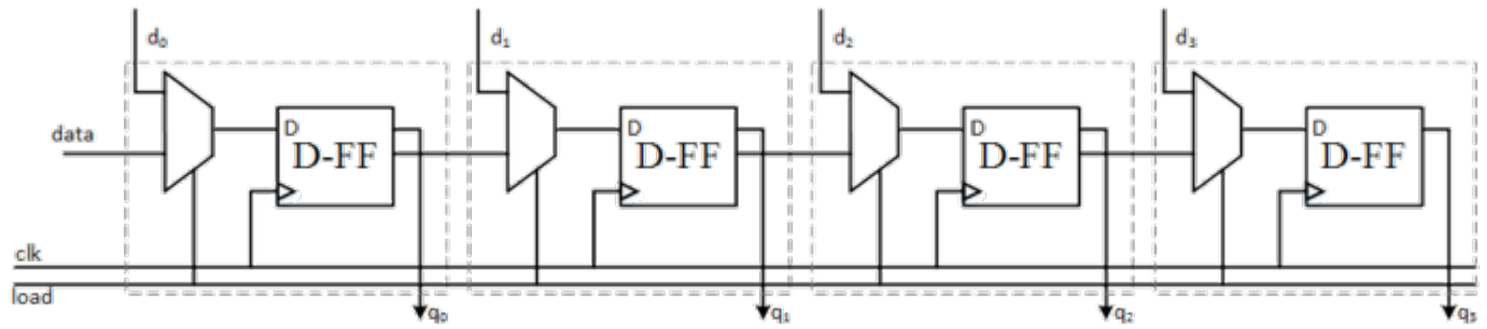
Verilog-standard cell

```
module shift_reg_cell (a,b,load,clk,q);

input  a,b,load,clk;
output reg c;

always @ (posedge clk)
begin
if (load==1'b1)
  q=a;
else
  q=b;
end
endmodule
```



Verilog-shift register

```
module shift_reg (data,clk,load,d,q);
input  data,clk,load;
input [0:3] d;
output [0:3] q;

shift_reg_cell DFF0(.a(d[0]),.b(data),.load(load),.clk(clk),.q(q[0]));
genvar i
generate
for (i=1; i<4;i=i+1)
begin
  shift_reg_cell OTHERS (.a(d[i]),.b(q[i-1]),.load(load),.clk(clk),.q(q[i]));
end
endgenerate
endmodule
```



UCI University of California, Irvine

INSTANTIATION EXAMPLE OF SHIFT REGISTER WITH LOAD (GENERATE)

N-bit Shift Register

```
module shift_reg (data,clk,load,d,q);
input  data,clk,load;
input [0: N] d;
output [0: N] q;
parameter N = 4
shift_reg_cell DFF0(.a(d[0]),.b(data),.load(load),.clk(clk),.q(q[0]));
genvar i
generate
for (i=1; i<N;i=i+1)
begin
  shift_reg_cell OTHERS (.a(d[i]),.b(q[i-1]),.load(load),.clk(clk),.q(q[i]));
end
endgenerate
endmodule
```

The change you need to make a generic shift register

LECTURE 5: OVERVIEW

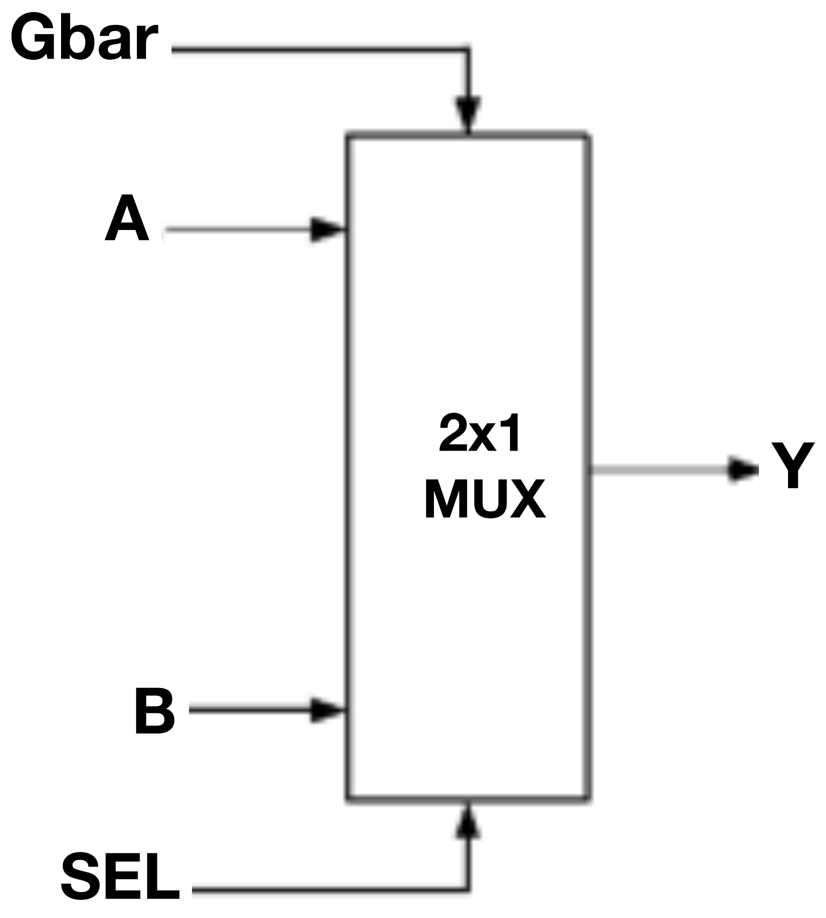
- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm

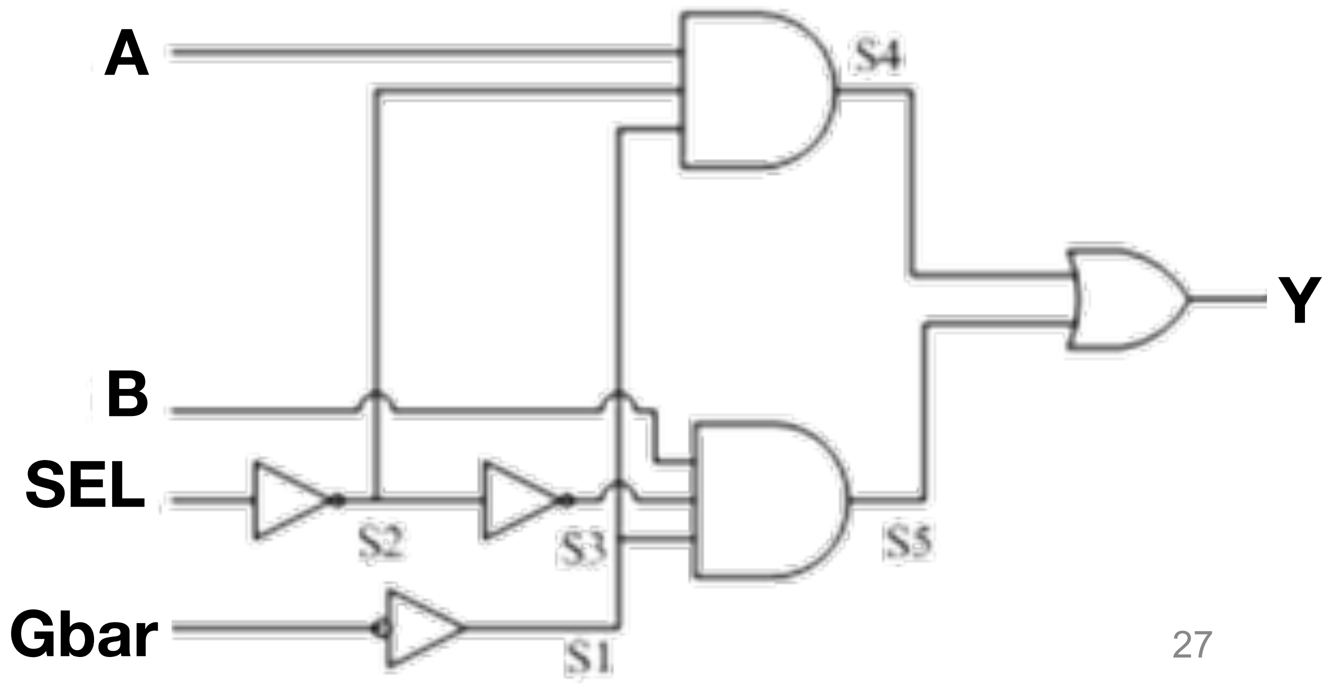
Break 10 minutes

EXAMPLE OF 1-BIT MUX 2_1 (WITH DELAY) FROM LECTURE 3

- 2x1 multiplexer
  - with active low enable
  - with constant delay
    - All gate delays are 7 ns

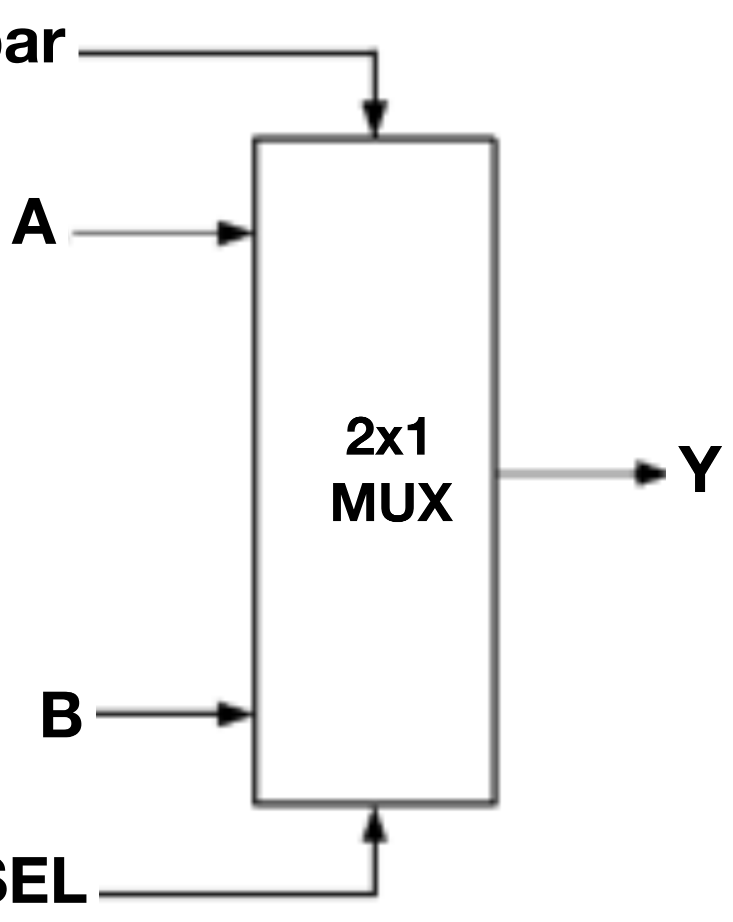
Input		Output
SEL	Gbar	Y
X	1	0
0	0	A
1	0	B

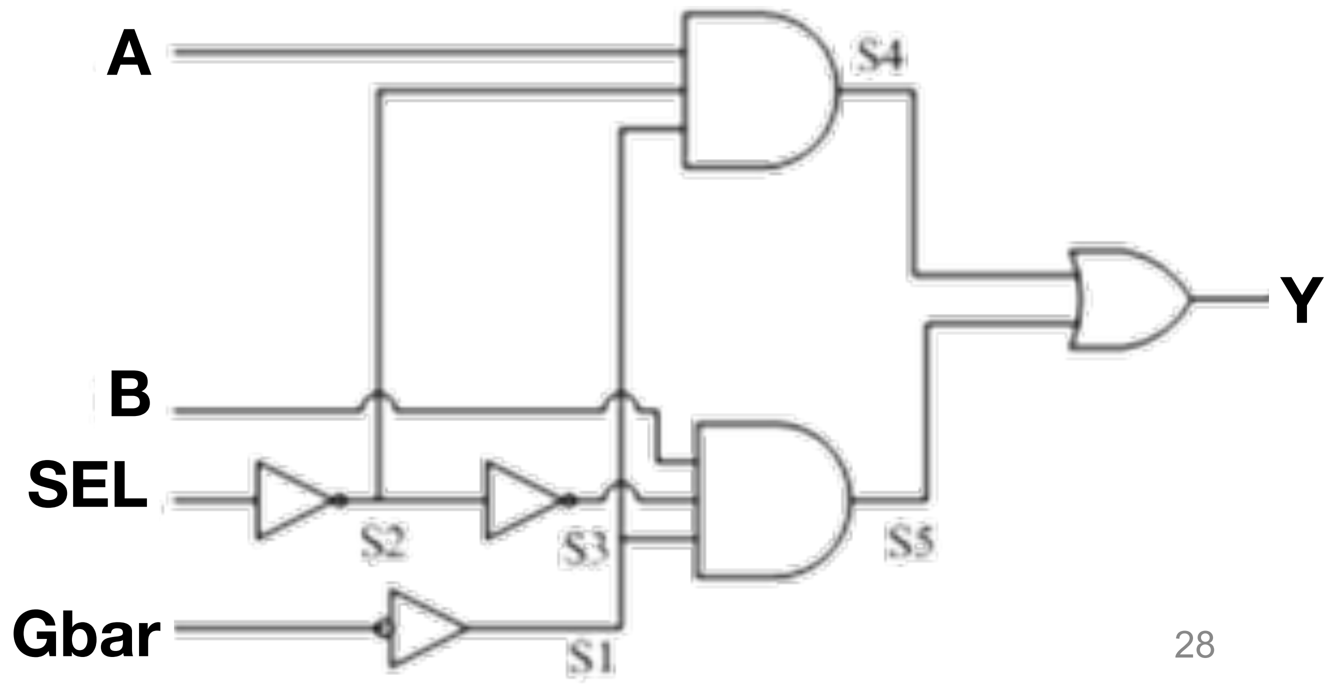




EXAMPLE OF 1-BIT MUX 2_1 (WITH DELAY) FROM LECTURE 3

```
module mux_enable(a,b,sel,gbar,y)
   input a,b,sel,gbar;
   output y;
 wire s1,s2,s3,s4,s5;
 time  dly = 7;//simulation screen units
 assign #dly  y=s4|s5;
 assign #dly  s4=a & s2 & s1;
 assign #dly  s5=b & s3 & s1;
 assign #dly  s2= ~sel;
 assign #dly  s3= ~s2;
 assign #dly  s1= ~gbar;
 endmodule
```

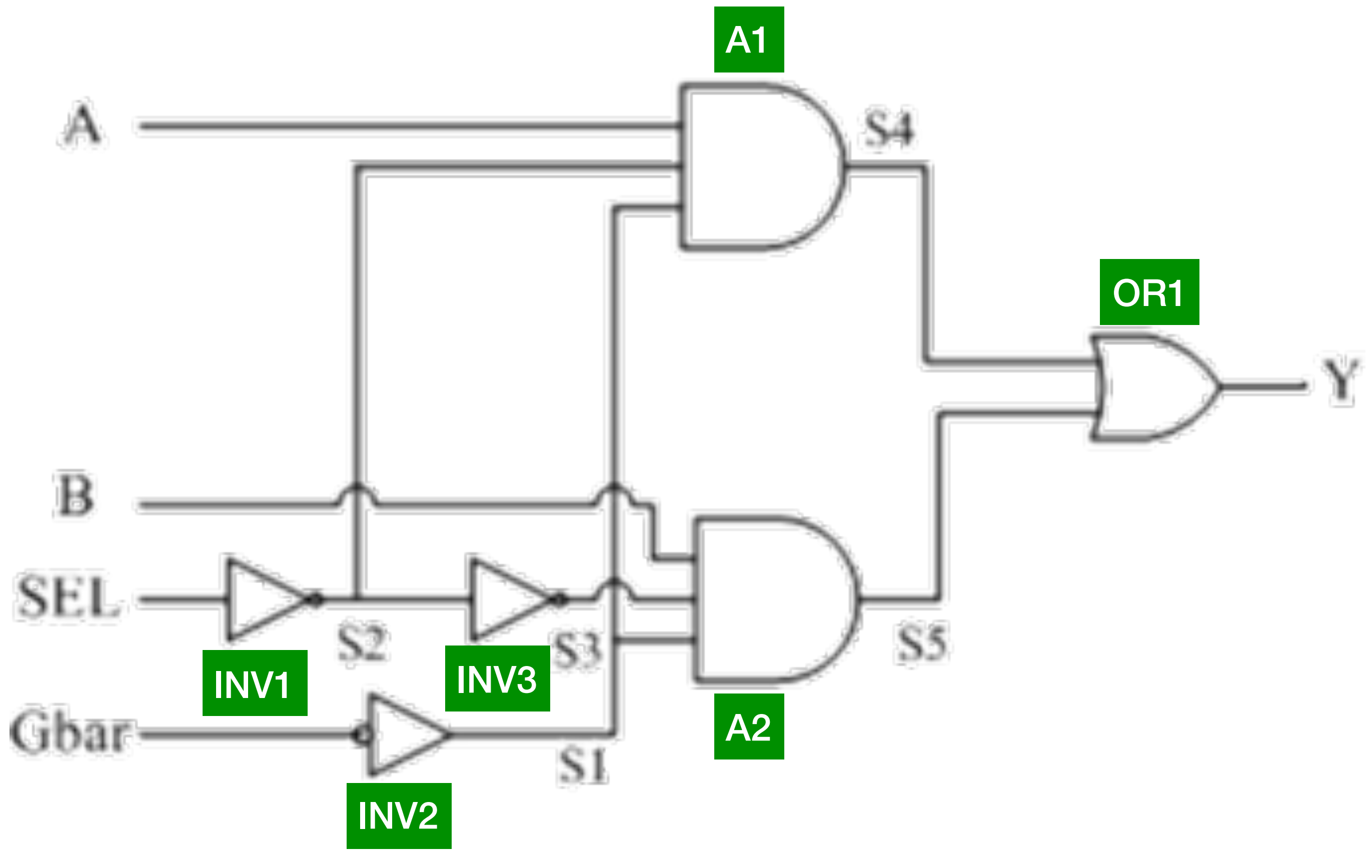




UCI University of California, Irvine

STRUCTURAL DESCRIPTION OF 1-BIT MUX 2_1 (WITH DELAY)

- MUX 2_1 (See lecture 3)
  - Not gates
    - 0 delay
    - 7ns delay
  - And gate
    - 2 inputs, 0 delay
    - 2 inputs, 7ns delay
    - 3 inputs, 0 delay
    - 3 inputs, 7ns delay
  - Or gate
    - 2 inputs, 0 delay
    - 2 inputs, 7ns delay
    - 3 inputs, 0 delay
    - 3 inputs, 7ns delay
  - Xor gate
    - 2 inputs, 0 delay



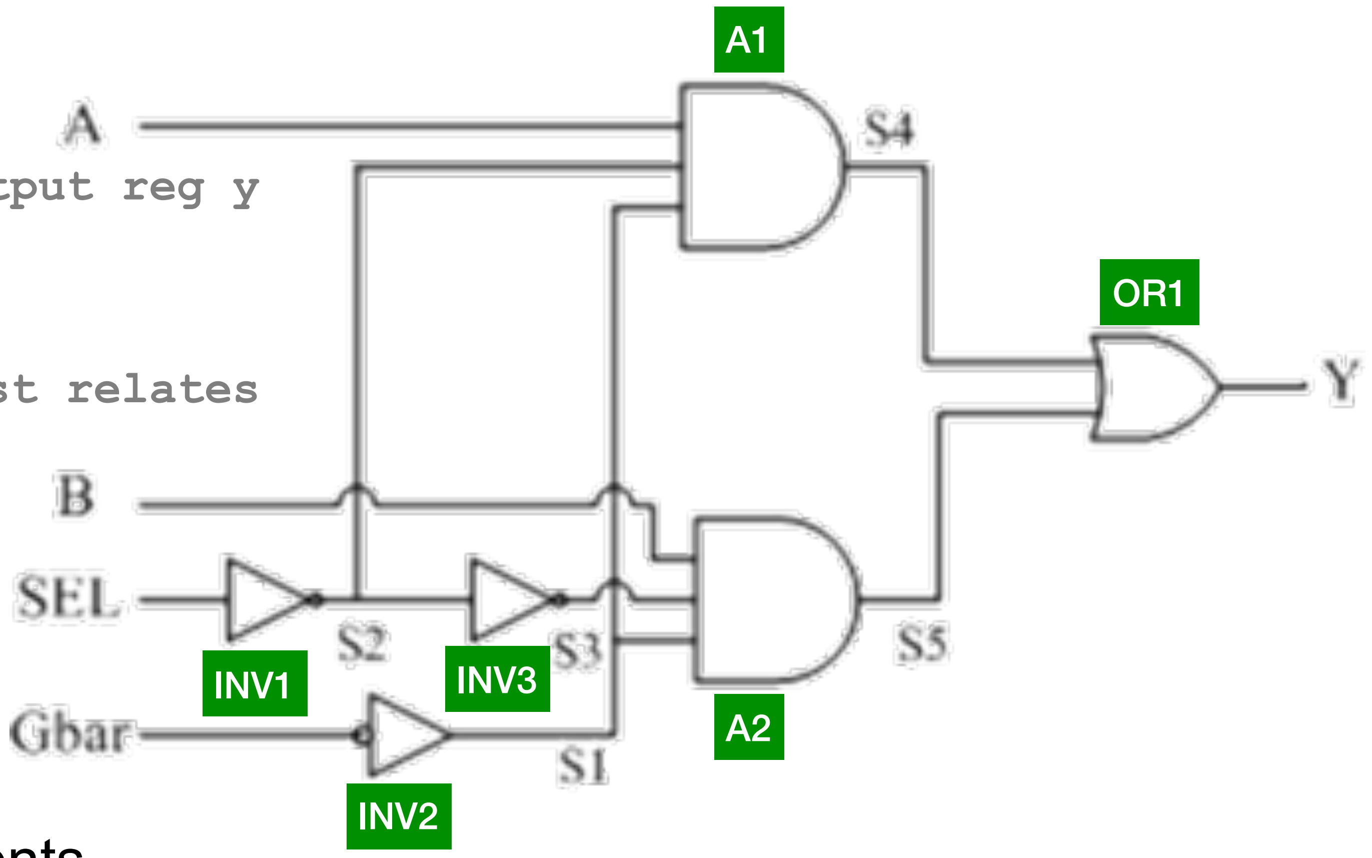
Instance	delay
INV1, INV3	7ns
INV2	0ns
A1,A2	7ns
OR1	7ns

STRUCTURAL DESCRIPTION OF 1-BIT MUX 2_1 (WITH DELAY)

```
module mux2x1 (a,b,sel,gbar,y);
input  a,b,sel,gbar;
output wire y;  // no need to be defined as output reg y

wire s1,s2,s3,s4,s5;

and #7 (s4,a,s2,s1);  /* the red ones, the first relates
to output*/
or  #7 (y,s4,s5);
and #7 (s5,b,s3,s1);
not #7 (s2,sel);
not #7 (s3,s2);
not  (s1,gbar);

endmodule
```

why?

Because all the assignments are concurrent in structural, so the y should be net type not reg type



Instance	delay
INV1, INV3	7ns
INV2	0ns
A1,A2	7ns
OR1	7ns

LECTURE 5: OVERVIEW

- Blocking vs. non blocking assignment
- Structural description
- Binding and port instantiation
- Instantiation with GENERATE
- Structural with Delay
- Prepare for the midterm - Last year midterm will be posted

PREPARE FOR THE MIDTERM

- Differences between the three types of design description:
  - Dataflow
  - Behavioral
  - Structural
- Assignment with delays in concurrent execution
- Difference between concurrent and sequential execution
- Blocking vs. non blocking
- Common mistakes in Verilog
- How to write a testbench (initial block, simulate a clock, …)

QUESTIONS