STRUCTURAL /GATE LEVEL MODELING

Dr.K.Sivasankaran
Associate Professor,
VLSI Division
School of Electronics Engineering
VIT University

Structural / Gate Level Modeling

- The module is implemented in terms of logic gates and interconnections between these gates.
- All the basic gates are available as ready modules called "Primitives."
- Design at this level is similar to describing a design in terms of a gate-level logic diagram.

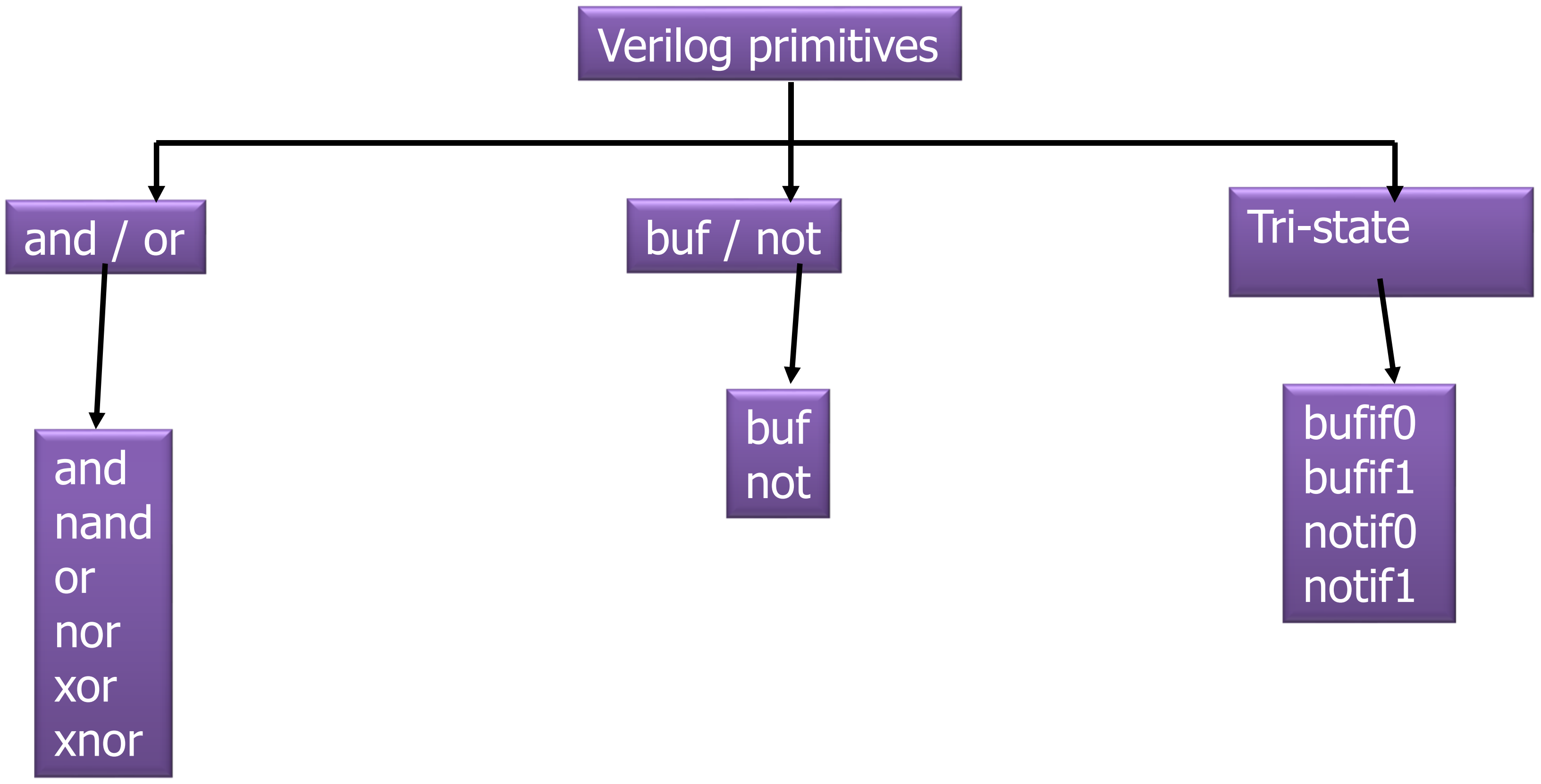
Structural / Gate Level Modeling

Features:

- Hardware design is described using instantiations of both primitives as well as modules.
- Logic independent of the ordering of instantiations of both primitives and modules.
- Concurrent execution of both primitives and modules.
- Instance name is mandatory for modules but optional for primitives.

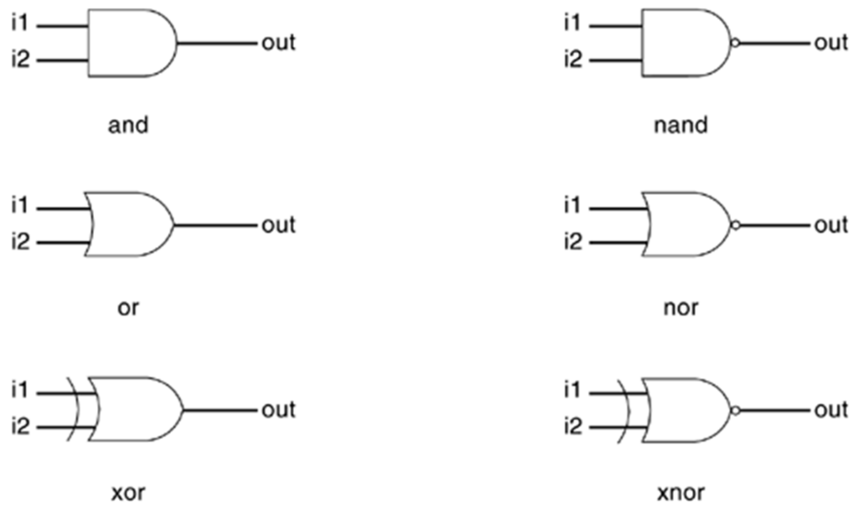
Realization of Combinational Circuits

Classification of primitives



Verilog Logic Gate Primitives

- Verilog supports basic logic gates as predefined primitives.
- These primitives are instantiated like modules except that they are predefined in Verilog.
- No module definition is needed for using the primitives.



Primitive gates

Features:

1-output, multiple inputs.

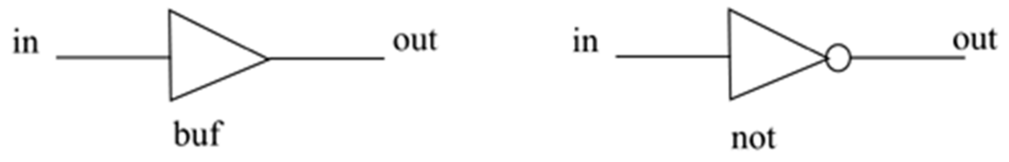
Output transitions (0, 1, x).

```
and i1 (output, input_1, input_2, …, input_n);
nand i2 (output, input_1, input_2, …, input_n);
or  i3 (output, input_1, input_2, …, input_n);
nor  i4 (output, input_1, input_2, …, input_n);
xor i5 (output, input_1, input_2, …, input_n);
xnor i6 (output, input_1, input_2, …, input_n);
```

buf / not primitives

Features:

- 1-input, multiple outputs.
- Output transitions (0, 1, x).



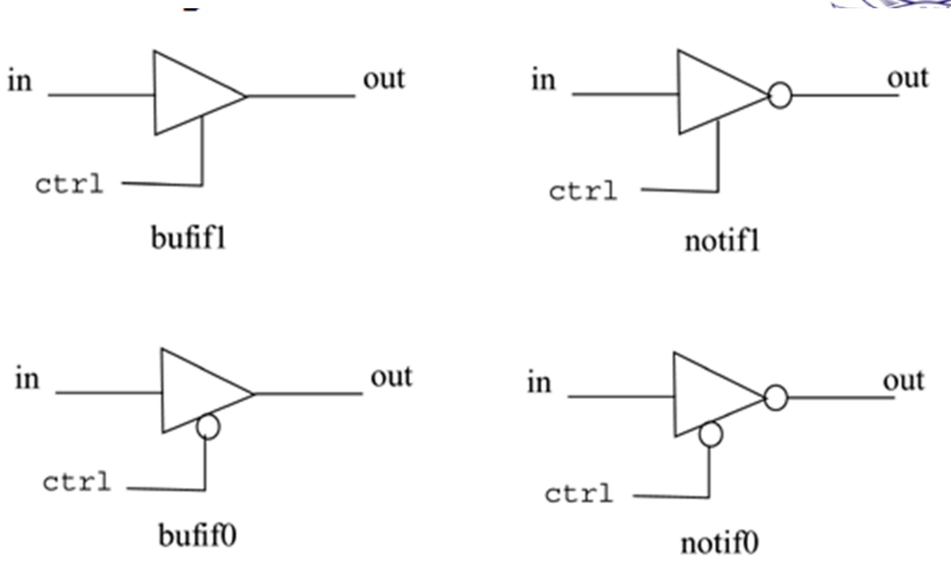
buf i1 (output_1, output_2, …, output_n, input);

not i2 (output_1, output_2, …, output_n, input);

Tri-state Primitives

Features:

- Has only 3 terminals.
- Output transitions (0, 1, x,



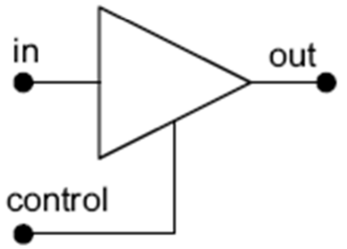
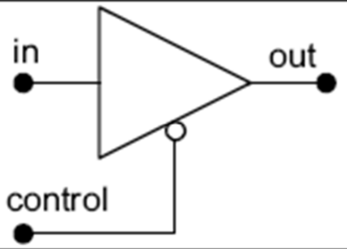
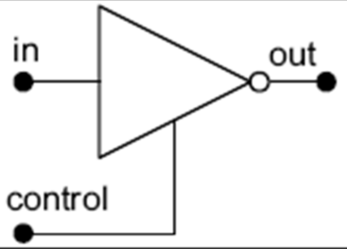
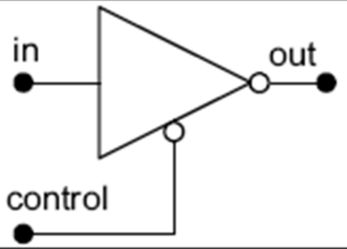
bufif0 i1 (output, data input, control input);

bufif1 i2 (output, data input, control input);

notif0 i3 (output, data input, control input);

notif1 i4 (output, data input, control input);

Instantiation and functional details of tri-state buffer primitives

Typical instantiation	Functional representation	Functional description
`bufif1` (out, in, control);		Out = in if control = 1; else out = z
`bufif0` (out, in, control);		Out = in if control = 0; else out = z
`notif1` (out, in, control);		Out = complement of in if control = 1; else out = z
`notif0` (out, in, control);		Out = complement of in if control = 0; else out = z

AND / OR PRIMITIVES TRUTH TABLE

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

nand	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

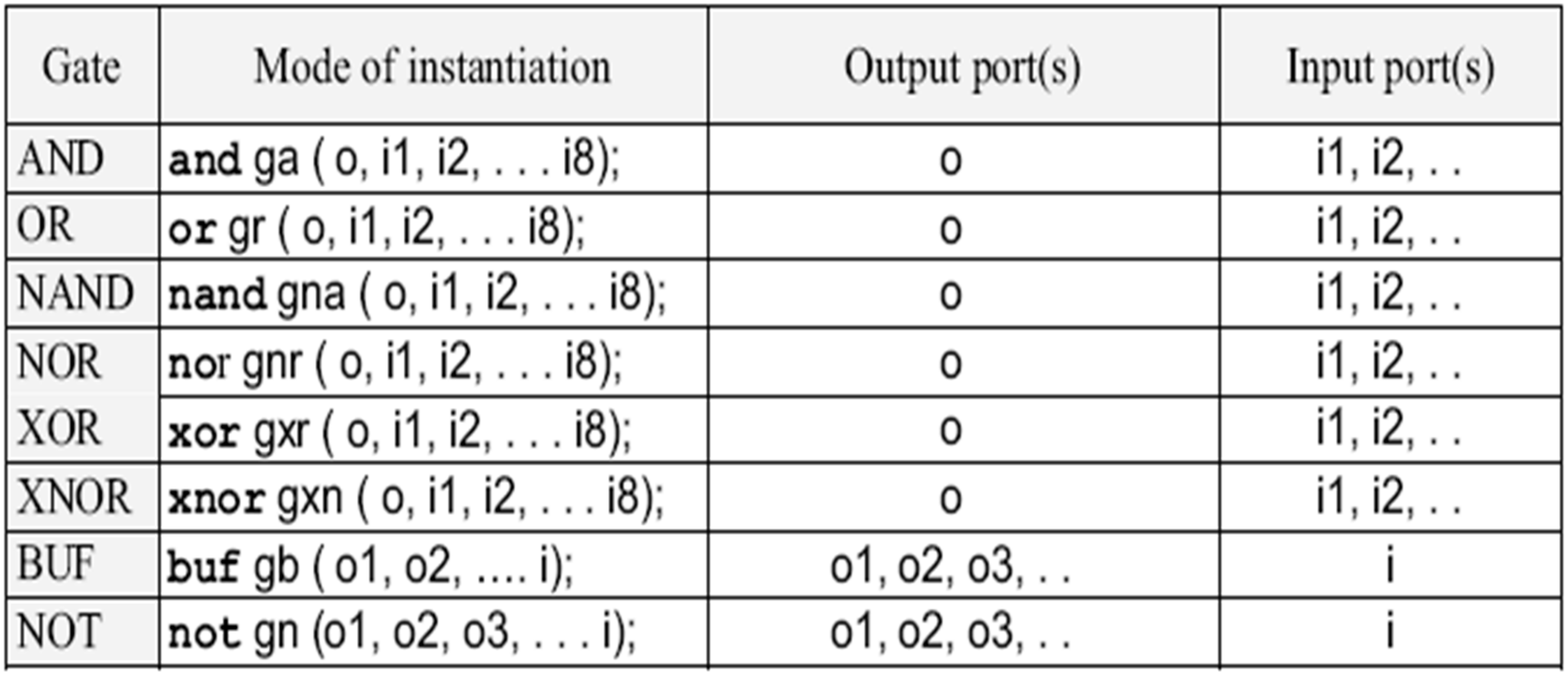
nor	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

BUF / NOT PRIMITIVES TRUTH TABLE

buf	
input	output
0	0
1	1
x	x
z	x

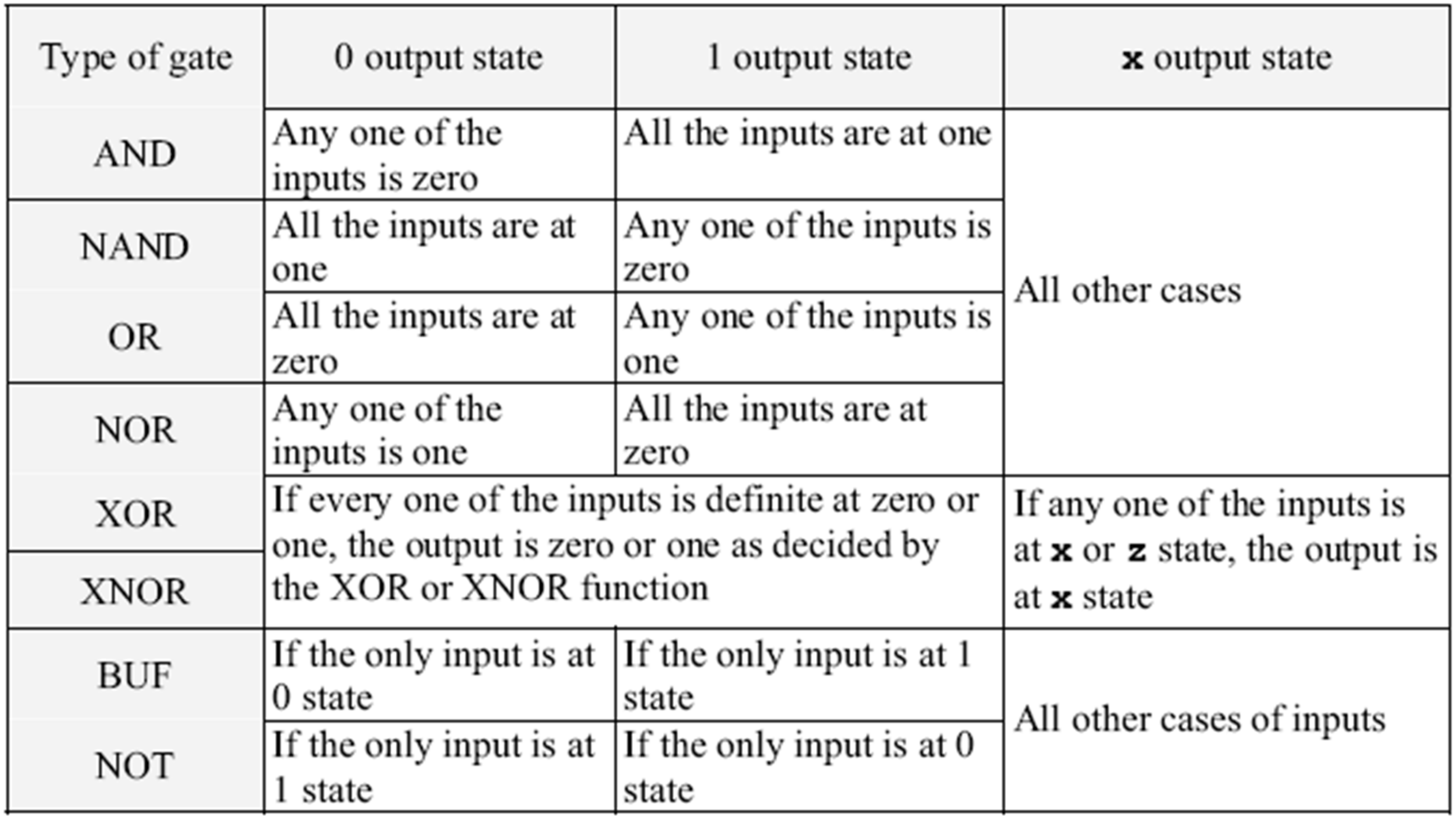
not	
input	output
0	1
1	0
x	x
z	x

Basic gate primitives in Verilog with details



Gate	Mode of instantiation	Output port(s)	Input port(s)
AND	**and** ga (o, i1, i2, . . . i8);	o	i1, i2, . .
OR	**or** gr (o, i1, i2, . . . i8);	o	i1, i2, . .
NAND	**nand** gna (o, i1, i2, . . . i8);	o	i1, i2, . .
NOR	**nor** gnr (o, i1, i2, . . . i8);	o	i1, i2, . .
XOR	**xor** gxr (o, i1, i2, . . . i8);	o	i1, i2, . .
XNOR	**xnor** gxn (o, i1, i2, . . . i8);	o	i1, i2, . .
BUF	**buf** gb (o1, o2, ... i);	o1, o2, o3, . .	i
NOT	**not** gn (o1, o2, o3, . . . i);	o1, o2, o3, . .	i

Rules for deciding the output values of gate primitives for different input combinations



Type of gate	0 output state	1 output state	**x** output state
AND	Any one of the inputs is zero	All the inputs are at one	All other cases
NAND	All the inputs are at one	Any one of the inputs is zero	
OR	All the inputs are at zero	Any one of the inputs is one	
NOR	Any one of the inputs is one	All the inputs are at zero	
XOR	If every one of the inputs is definite at zero or one, the output is zero or one as decided by the XOR or XNOR function		If any one of the inputs is at **x** or **z** state, the output is at **x** state
XNOR			
BUF	If the only input is at 0 state	If the only input is at 1 state	All other cases of inputs
NOT	If the only input is at 1 state	If the only input is at 0 state	

Array of Instances of primitives

The primitives available in Verilog can also be instantiated as arrays.

A judicious use of such array instantiations often leads to compact design descriptions.

A typical array instantiation has the form

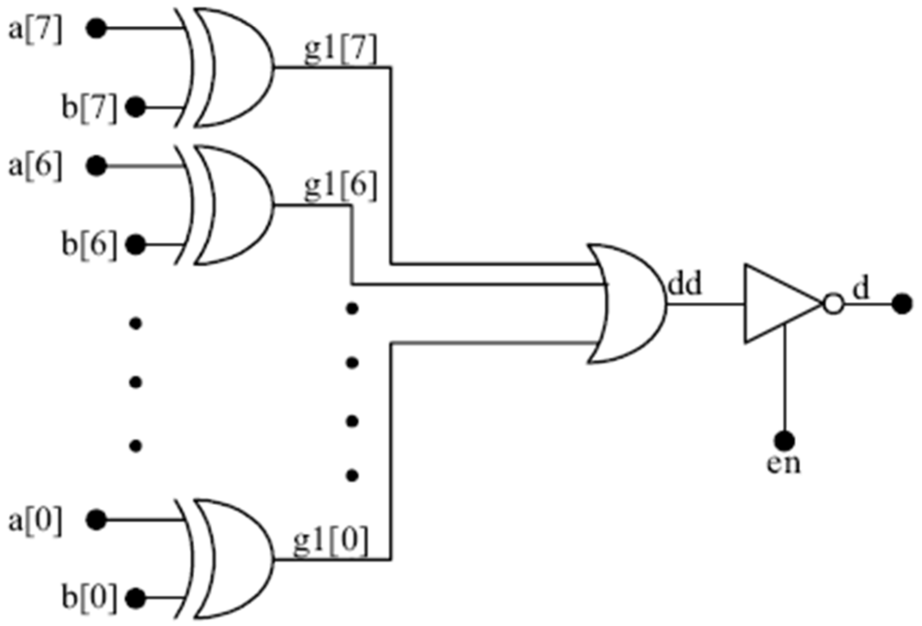
```
and gate [7 : 4 ] (a, b, c);
```

where a, b, and c are to be 4 bit vectors.

The above instantiation is equivalent to combining the following 4 instantiations:

```
and gate [7] (a[3], b[3], c[3]), gate [6] (a[2], b[2], c[2]), gate [5] (a[1], b[1],
c[1]), gate [4] (a[0], b[0], c[0]);
```

Example – Array of Instance

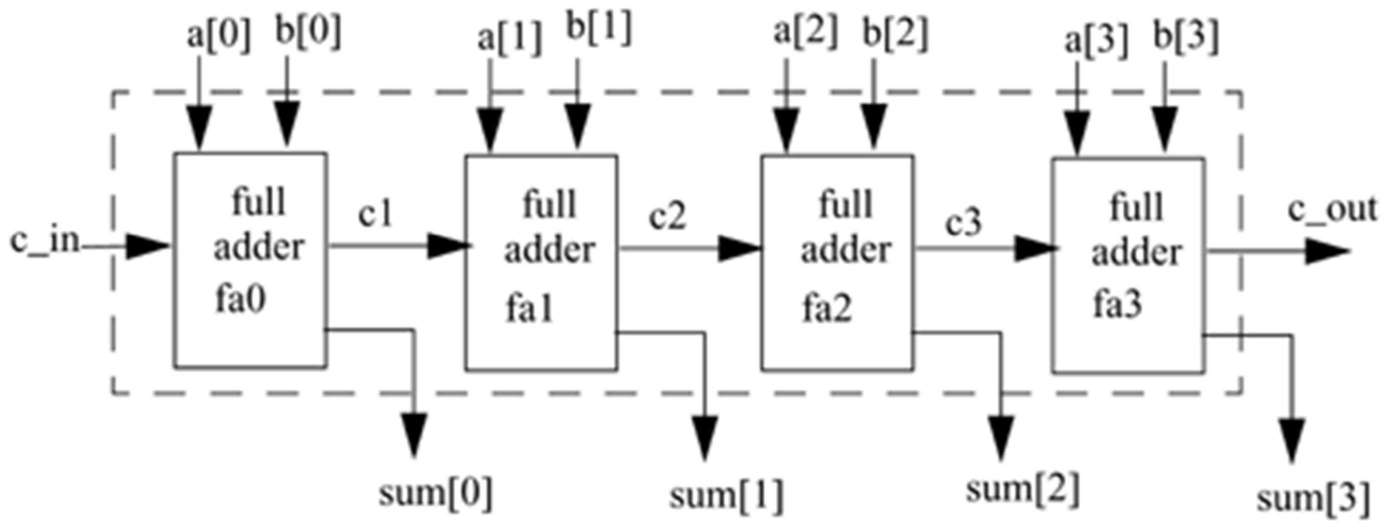


```
module comp(d,a,b,en);
input en;
input[7:0]a,b;
output d;
wire [7:0]c;
wire dd;
xor g1[7:0](c,b,a);
or(dd,c);
notif1(d,dd,en);
endmodule
```

In the general case the array size is specified in terms of two constant expressions. These can involve constants, previously defined parameters and algebraic operators:

Such an instantiation can have a form as
and gate [offset*2+size-1: offset*2] (a, b, c);

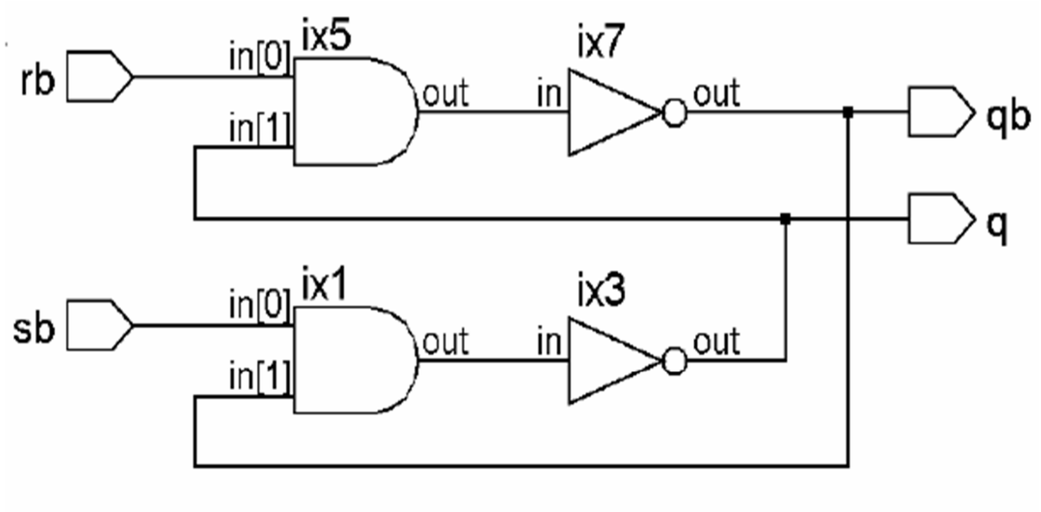
Realize 4-bit Ripple Carry Full Adder



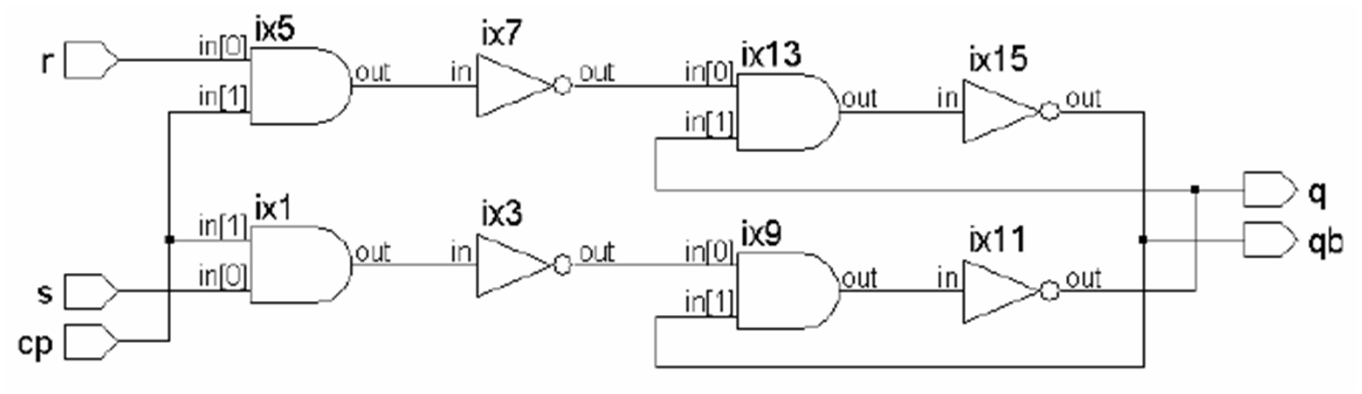
Realization of Sequential Circuits

RS Latch

```
module sbrbff(sb,rb,q,qb);
input sb,rb;
output q,qb;
nand(q,sb,qb);
nand(qb,rb,q);
endmodule
```

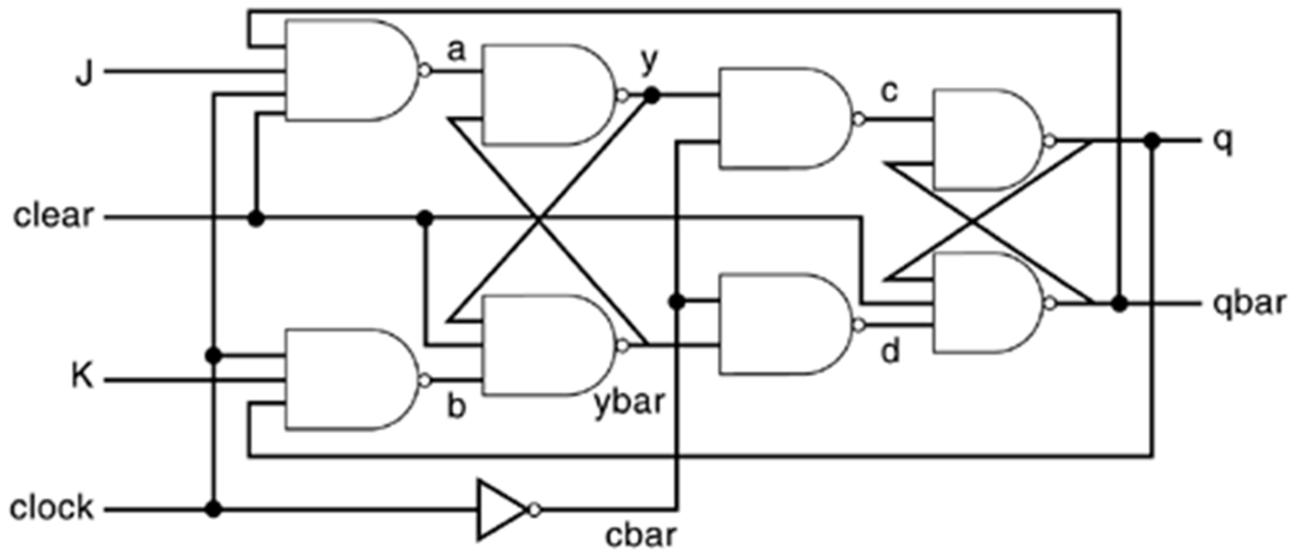


RS Flip Flop

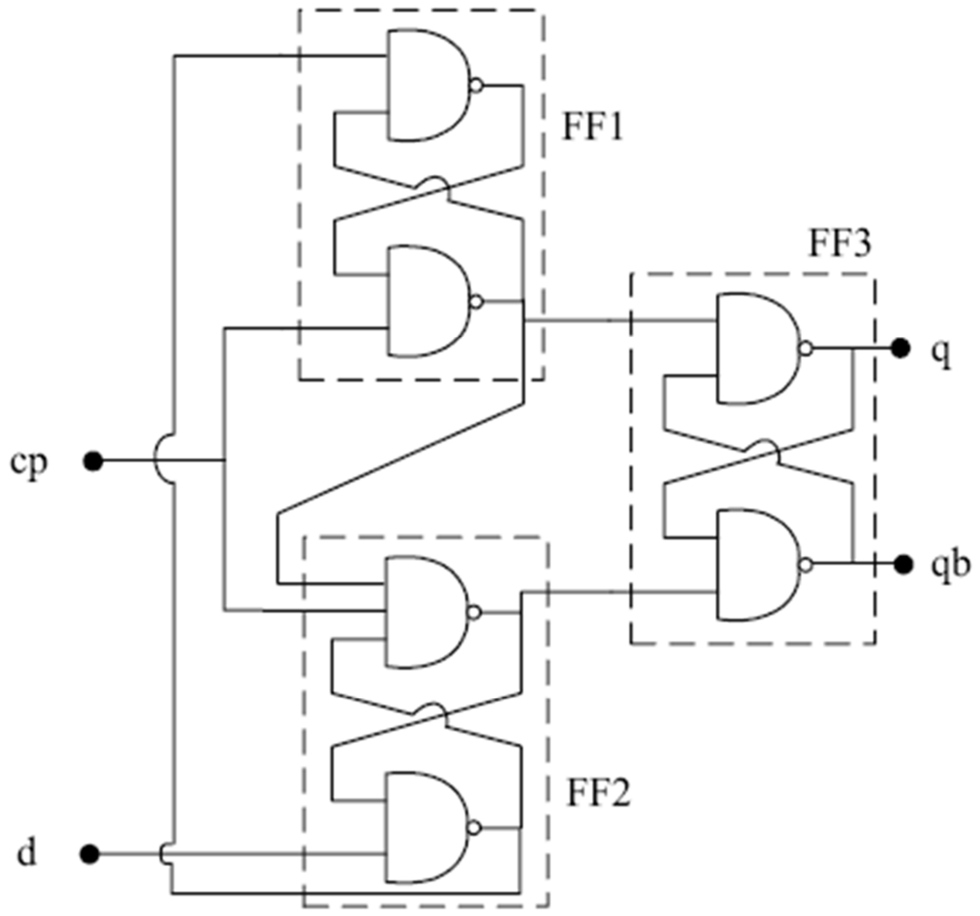


```
module srff(cp,s,r,q,qb);
input cp,s,r;
output q,qb;
wire ss,rr;
nand (ss,s,cp),(rr,r,cp),(q,ss,qb),(qb,rr,q);
endmodule
```

Master-Slave JK-flipflop



Edge Triggered Flip-Flop

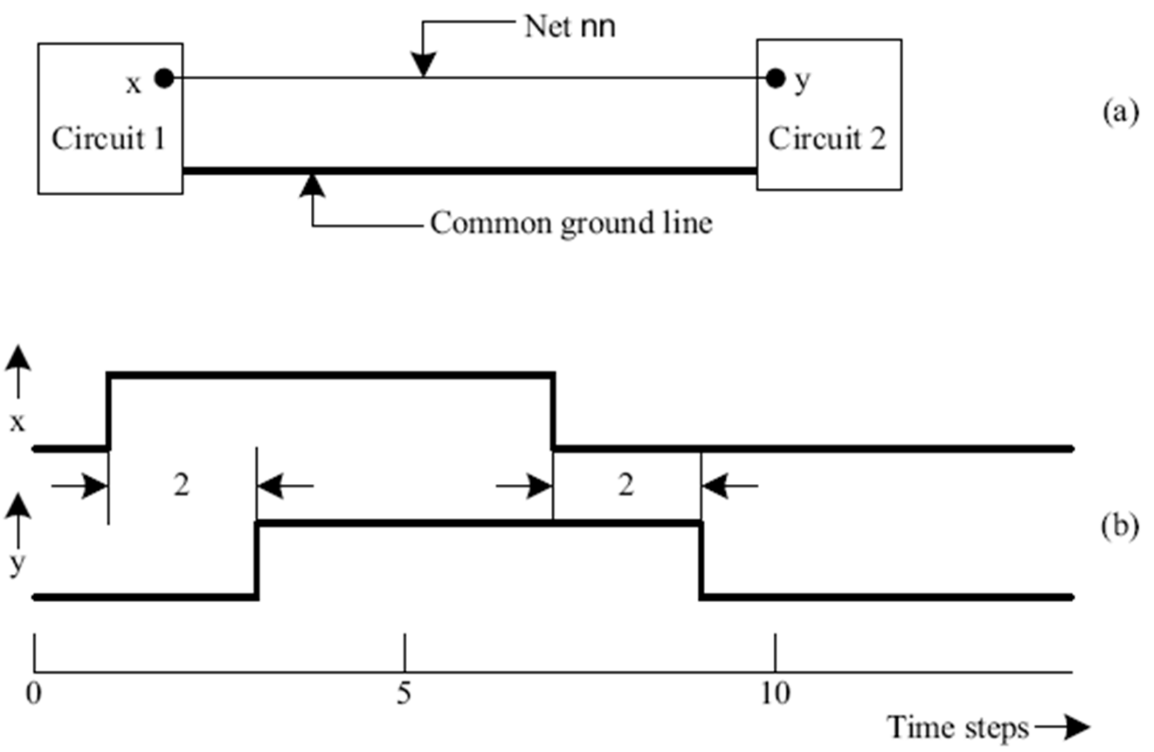


Net Delay

One of the simplest delays is that of a direct connection – a net. It can be part of the declaration statement

wire #2 nn; // nn is declared as a net with a propagation delay of 2 time steps

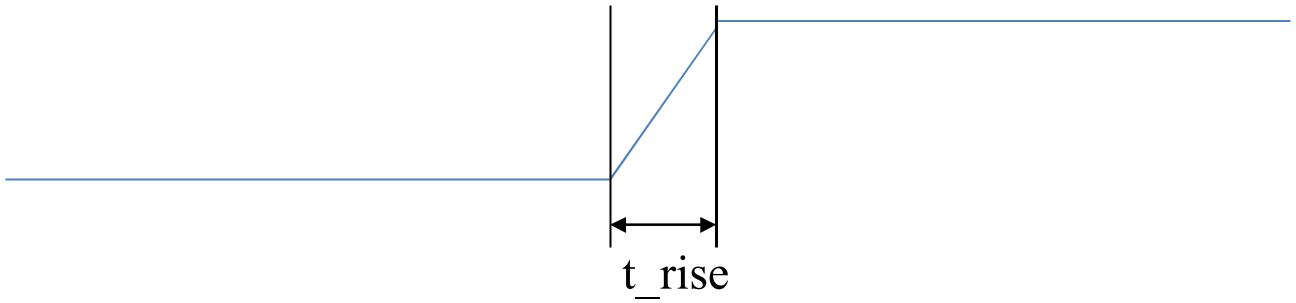
wire # (2, 1) nm; //nm is declared as net with rise delay of 2t.u and fall delay of 1t.u



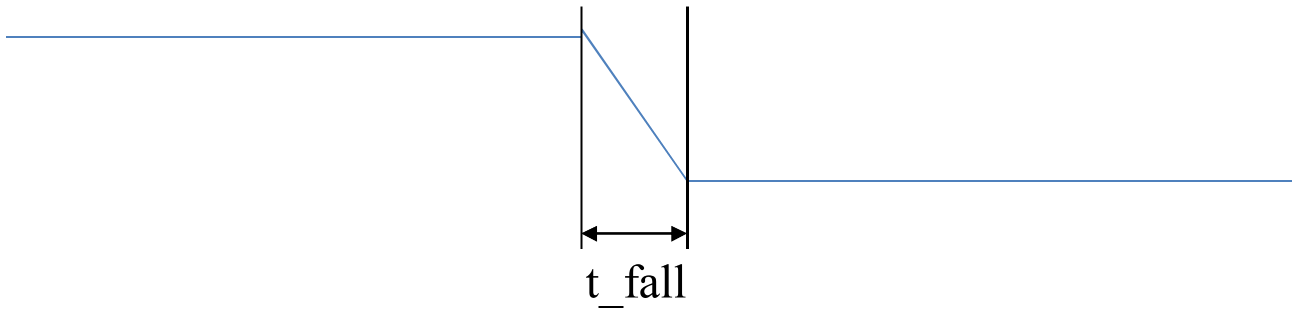
```
module
netdelay(x,y);
input x;
output y;
wire #2 nn;
not (nn,x);
buf y = x; //
endmodule
```

Gate Delays

- Rise delay is associated with a gate output transition to 1 from another value



- Fall delay is associated with a gate output transition to 0 from another value



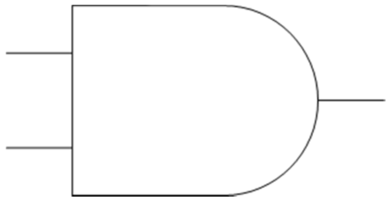
- Turn-off delay is associated with a gate output transition to the high impedance value (z) from another value

Gate delay specifications

- One delay specification: If specified, it is used for all transitions.

  and #(delay time) a1 (out, i1, i2);

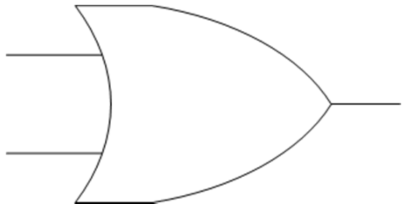
  and #(4) a1 (out, i1, i2);



- Two delay specification: If specified, they refer to rise and fall times.

  or #(rise_del, fall_del) o1 (out, i1, i2);

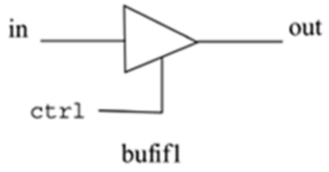
  or #(5, 6) o1 (out, i1, i2);



- Three delay specification: If specified, they refer to rise, fall and turn-off times.

  bufif1 #(rise_del, fall_del, turn_off_del) b1 (out, in, cnt);

  bufif1 #(2, 3, 5) b1 (out, in, crtl);



MIN : TYP : MAX VALUES

- Primitive gate delays allow three values each for the rise, fall and turn-off delays.
- The three values are minimum, typical and maximum, and the three are separated by colons.
- Either of the three values can be selected at the start of the simulation (run time). If no value is selected, typical value is the default.

EXAMPLES OF MIN : TYP : MAX VALUES

- One delay specification with min:typ:max values.

  and #(2:4:5) a1 (out, i1, i2);

  rise_{min},fall_{min} = 2, rise_{typ},fall_{typ} = 4, rise_{max},fall_{max} = 5.

- Two delay specification with min:typ:max values.

  or #(1:5:3, 2:6:4) o1 (out, i1, i2);

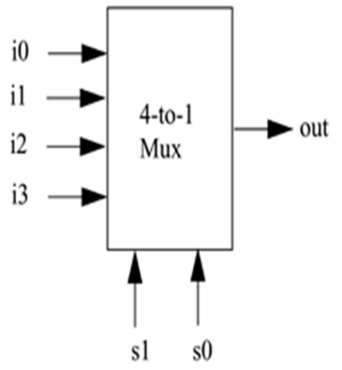
  rise_{min}=1, rise_{typ}=5, rise_{max}=3, fall_{min}=2, fall_{typ}=6, fall_{max}=4.

- Three delay specification with min:typ:max values.

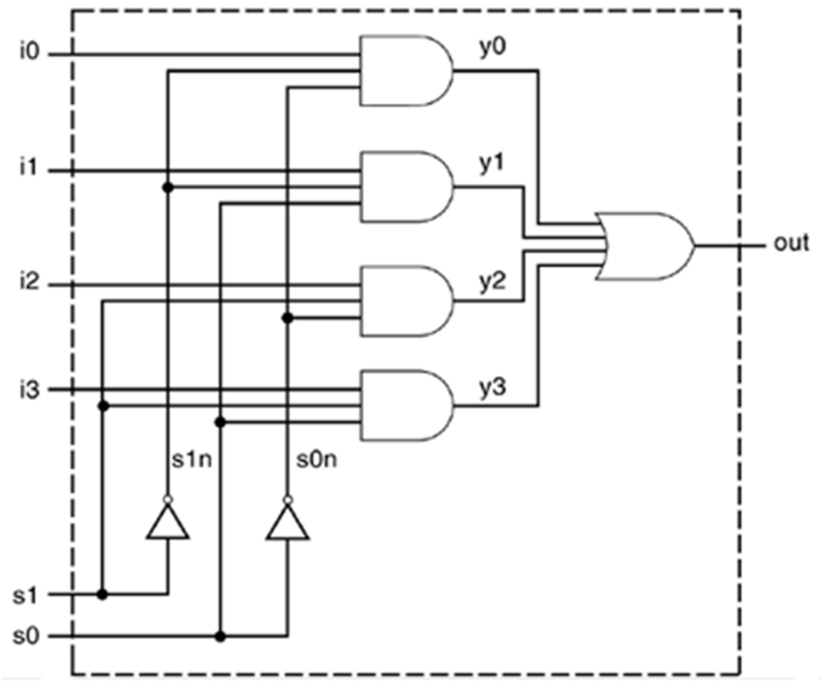
  bufif1 #(1:2:4, 1:3:5, 3:5:6) b1 (out, i1, i2);

  rise_{min}=1, rise_{typ}=2, rise_{max}=4, fall_{min}=1, fall_{typ}=3, fall_{max}=5, turn-off_{min}=3, turn-off_{typ}=5, turn-off_{max}=6.

Example - Primitive Instantiation



s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3



Primitive Instantiation - Example

// Module 4-to-1 multiplexer. Port list is taken exactly from the I/O diagram.

```
module mux4_to_1 (out, i0, i1, i2, i3,
    s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
wire s1n, s0n;
wire y0, y1, y2, y3;
not (s1n, s1);
 not (s0n, s0);
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);
or (out, y0, y1, y2, y3);
endmodule
```

Module Instantiation - Example

```
module twmux (a,b,s,y);
input a,b,s;
output y;
wire y,s1,w1,w2;
not n1(s1,s);
and a1(w1,a,s);
and a2 (w2,b,s1);
or o1(y,w1,w2);
endmodule
```

```
module  frmux (a,b,c,d,se1,se2,y);
input a,b,c,d,se1,se2;
output y;
wire y,se1,se2,w1,w2;
twmux t1(a,b,se2,w1);
twmux t2(c,d,se1,w2);
twmux t3(w1,w2,se1,y);
endmodule
```

Connecting Ports to External Signals

- There are two methods of making connections between signals specified in the module instantiation and the ports in a module definition. These two methods cannot be mixed.
  - Connecting by ordered list
  - Connecting ports by name

Connecting by ordered list

- Connecting by ordered list is the most intuitive method for most beginners.
- The signals to be connected must appear in the module instantiation in the same order as the ports in the port list in the module definition

Connecting by ordered list-Example

```
module twmux (a,b,s,y);
input a,b,s;
output y;
wire y,s1,w1,w2;
not n1(s1,s);
and a1(w1,a,s);
and a2 (w2,b,s1);
or o1(y,w1,w2);
endmodule
```

```
module  frmux (a,b,c,d,se1,se2,y);
input a,b,c,d,se1,se2;
output y;
wire y,w1,w2;
twmux t1(a,b,se1,w1);
twmux t2(c,d,se1,w2);
twmux t3(w1,w2,se2,y);
endmodule
```

Connecting Ports by name

- For large designs where modules have, say, 50 ports, remembering the order of the ports in the module definition is impractical and error-prone.

- Verilog provides the capability to connect external signals to ports by the port names, rather than by position.

Connecting Ports by name -Example

```
module twmux (a,b,s,y);
input a,b,s;
output y;
wire y,s1,w1,w2;
not n1(s1,s);
and a1(w1,a,s);
and a2 (w2,b,s1);
or o1(y,w1,w2);
endmodule
```

```
module  frmux (a,b,c,d,se1,se2, y);
input a,b,c,d,se1,se2;
output y;
wire y, w1,w2;
twmux t1(.a(a), .b(b), .s(se1), .y(w1) );
twmux t2(.a (c), .b(d), .s(se1), .y(w2) );
twmux t3(.a(w1), .b(w2), .s(se2), .y(y));
endmodule
```