UNIT-2
GATE LEVEL MODELING

Syllabus: Introduction, AND Gate Primitive, Module Structure, Other Gate Primitives, Illustrative Examples, Tri-State Gates, Array of Instances of Primitives, Design of Flip – Flops with gate primitives, Delays, Strengths and Contention Resolution, Net Types , Design of Basic Circuits.

AND GATE PRIMITIVE: All the basic gates are available as "Primitives" in Verilog.

- The AND gate primitive in Verilog is instantiated with the following statement:

and g1 (O, I1, I2, . . ., In);

Here '**and**' is the keyword signifying an AND gate.

g1 is the name assigned to the specific instantiation.

O is the gate output.

I1, I2, *etc*., are the gate inputs.

Stimulus program for Testing AND GATE

- The module test_and has no port. It instantiates the AND module once.
- The test input sequence is specified within the **initial** block – the sequence of statements between the **begin** and **end** statements together form this block.
- The keyword "**initial**" signifies the settings done initially — that is, only once for the whole routine.
- The first set of statements within the **initial** block

a1 = 0;
a2 = 0;
make
a1 = a2 = 0
at zero simulation time

```
module test_and;
reg a1, a2;
wire b;
Initial
Begin
        a1 = 0;
        a2 = 0;
 #3     a1 = 1;
 #1     a1 = 0;
 #2     a2 = 1;
 #4     a1 = 1;
 #3     a2 = 0;
 #1     a2 = 1;
end
and g1(b, a1, a2);
initial $monitor ( $time, "a1 = %b, a2 = %b, b = %b"' a1, a2, b);
initial #100 $finish;
endmodule
```

After 3 time steps, a1 is set to one but a2 remains at 0. The expression "#3" means "after 3 time steps". Subsequent changes in a1 and a2 also can be explained in the same manner.

MODULE STRUCTURE

- The first statement of a module starts with the keyword **module**; it may be followed by the name of the module and the port list if any
- All the variables in the ports-list are to be identified as **inputs**, **outputs,** or **inouts**.
- The corresponding declarations have the form shown below:

  Input a1, a2;
  Output b1, b2;
  Inout c1, c2;
- The ports and the other variables used within the body of the module are to be identified as nets or registers with specific types in each case.
- The respective declaration statements follow the port-type declaration statements. Examples:

  wire a1, a2, c;
  reg b1, b2;
- The executable body of the module follows the declaration indicated above.
- The last statement in any module definition is the keyword "**endmodule**".
- Comments can appear anywhere in the module definition.

Other GATE Primitives

All the basic gates are available as "Primitives" in Verilog.

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

<table>
<tr><th>Type of gate</th><th>0 output state</th><th>1 output state</th><th>x output state</th></tr>
<tr><td>AND</td><td>Any one of the inputs is zero</td><td>All the inputs are at one</td><td rowspan="4">All other cases</td></tr>
<tr><td>NAND</td><td>All the inputs are at one</td><td>Any one of the inputs is zero</td></tr>
<tr><td>OR</td><td>All the inputs are at zero</td><td>Any one of the inputs is one</td></tr>
<tr><td>NOR</td><td>Any one of the inputs is one</td><td>All the inputs are at zero</td></tr>
<tr><td>XOR
XNOR</td><td colspan="2">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</td><td>If any one of the inputs is at x or z state, the output is at x state</td></tr>
<tr><td>BUF</td><td>If the only input is at 0 state</td><td>If the only input is at 1 state</td><td rowspan="2">All other cases of inputs</td></tr>
<tr><td>NOT</td><td>If the only input is at 1 state</td><td>If the only input is at 0 state</td></tr>
</table>

Table B.1 Truth table of AND gate

		Input 1			
		0	1	x	z
Input 2	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

Table B.2 Truth table of OR gate

		Input 1			
		0	1	x	z
Input 2	0	0	1	x	x
	1	1	1	1	1
	x	x	1	x	x
	z	x	1	x	x

Table B.3 Truth table of NAND gate

		Input 1			
		0	1	x	z
Input 2	0	1	1	1	1
	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x

Table B.4 Truth table of NOR gate

		Input 1			
		0	1	x	z
Input 2	0	1	0	x	x
	1	0	0	0	0
	x	x	0	x	x
	z	x	0	x	x

Table B.5 Truth table of XOR gate

		Input 1			
		0	1	x	z
Input 2	0	0	1	x	x
	1	1	0	x	x
	x	x	x	x	x
	z	x	x	x	x

Table B.6 Truth table of XNOR gate

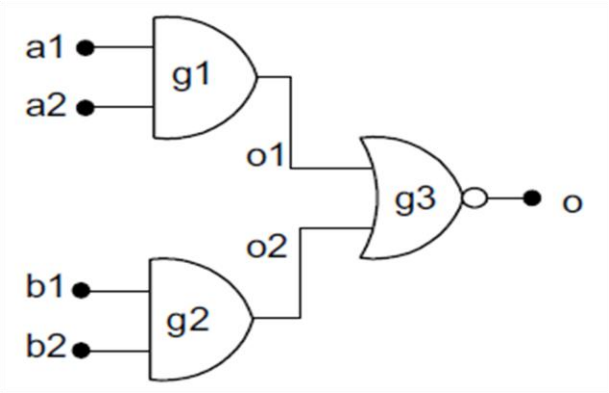
		Input 1			
		0	1	x	z
Input 2	0	1	0	x	x
	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

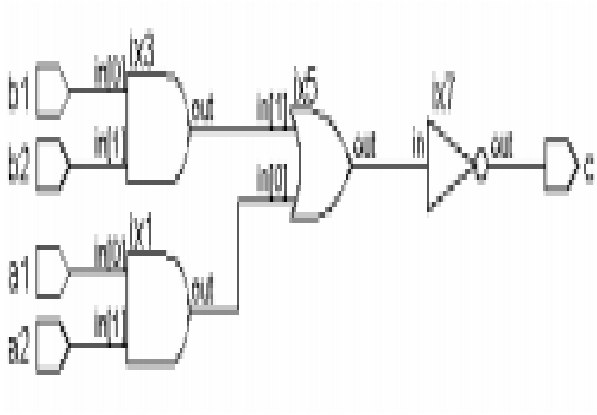
ILLUSTRATIVE EXAMPLES:

Verilog module for AOI logic

```
module aoi_gate(o,a1,a2,b1,b2);
input a1,a2,b1,b2;
output o;  wire o1,o2;
and g1(o1,a1,a2);
and g2(o2,b1,b2);
nor g3(o,o1,o2);
endmodule

module aoi_st;
reg a1,a2,b1,b2;  wire o;
initial  begin
a1 = 0; #3 a1 = 1;
a2 = 0; b1 = 0; b2 = 0;
a2 = 1; b1 = 1; b2 = 0;
```

```
end
initial #100 $stop;
initial $monitor($time , " o = %b , a1 = %b , a2 = %b , b1 = %b ,b2 = %b  ",o,a1,a2,b1,b2);
aoi_gate gg(o,a1,a2,b1,b2);
endmodule
```





Vector version of AOI GATE

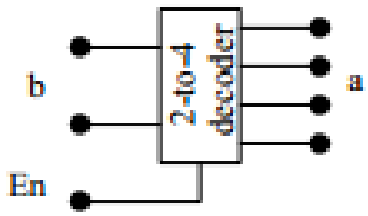
```
module aoi gate2(o,a);
input [3:0]a;//A is a vector of 4 bits width
output o;// output o is a scalar
wire o1,o2;//these are intermediate signals
and (o1,a[0],a[1]),(o2,a[2],a[3]);
nor (o,o1,o2);/*The nor gate has one instantiation
with assigned name g3.*/
endmodule

module aoi st2;
reg[3:0] aa;
aoi_gate2 gg(o,aa);
initial
       begin
       aa = 4'b000;//a being a vector, all its
  #3   aa = 4'b0001;//bit components are
  #3   aa = 4'b0010;//assigned values at one go.
  #3   aa = 4'b0100;//Similarly their changes are
  #3   aa = 4'b1000;//combined in the assignments
  #3   aa = 4'b1100;
  #3   aa = 4'b0110;
  #3   aa = 4'b0011;
       end
initial
$monitor(  $time , " aa = %b , o = %b " , aa,o);
initial #24 $stop;
endmodule
```

Verilog Program for 2 to 4 Decoder

```
Module decoder24 (D0,D1,D2,D3, A0,A1);
input A0,A1;
output D0,D1,D2,D3;
and g1 (D0, (~A1),(~A0));
and g2 (D1, (~A1),A0);
and g3 (D2, A1,(~A0));
and g4 (D3, A1,A0);
endmodule
```

Verilog Program for 2 to 4 Decoder in Vector version

```
module dec2 4 (a,b,en);
output [3:0] a;
input [1:0]b; input en;
wire [1:0]bb;
not(bb[1],b[1]),(bb[0],b[0]);
and(a[0],en, bb[1],bb[0]),(a[1],en, bb[1],b[0]),
(a[2],en, b[1],bb[0]),(a[3],en, b[1],b[0]);
endmodule
```



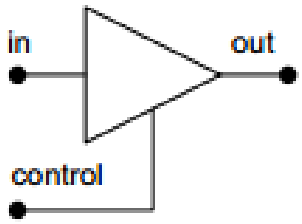
Testbench Program for 2 to 4 Decoder

```
module tst dec2 4();
wire [3:0]a;
reg[1:0] b; reg en;
dec2_4 dec(a,b,en);
initial
begin
  {b,en} =3'b000;
#2{b,en} =3'b001;
#2{b,en} =3'b011;
#2{b,en} =3'b101;
#2{b,en} =3'b111;
end
initial
$monitor ($time  ,  "output a =  %b, input b  = %b ",
a, b);
endmodule
```

TRI-STATE GATES

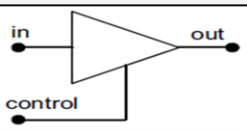
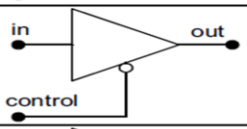
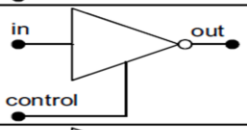
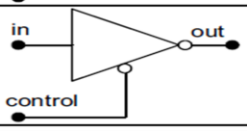
- Four types of tri-state buffers are available in Verilog as primitives.
- Their outputs can be turned ON or OFF by a control signal. The direct buffer is instantiated as **Bufif1 nn (out, in, control);**

  out as the single output variable
  in as the single input variable and **control** as the single control signal variable.



- When control = 1, out = in.
- When control = 0, out is cut off from the input and tri-stated.

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

ARRAY OF INSTANCES OF PRIMITIVES

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

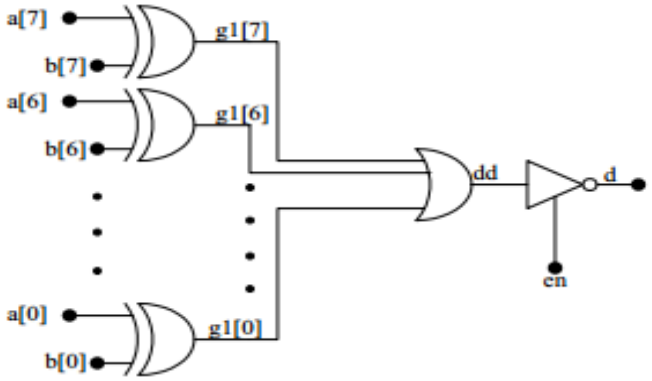
- and gate [7] (a[3], b[3], c[3]),
  gate [6] (a[2], b[2], c[2]),
  gate [5] (a[1], b[1], c[1]),
  gate [4] (a[0], b[0], c[0]);

Syntax: **and gate [mm : nn](a, b, c);**

A Byte Comparator

- A circuit to compare two variables each of one byte.
- The circuit outputs a flag d; d is 1 if the two bytes are equal; else it is 0.
- The output is activated only if the enable signal en = 1. If en = 0, the circuit output is tri-stated

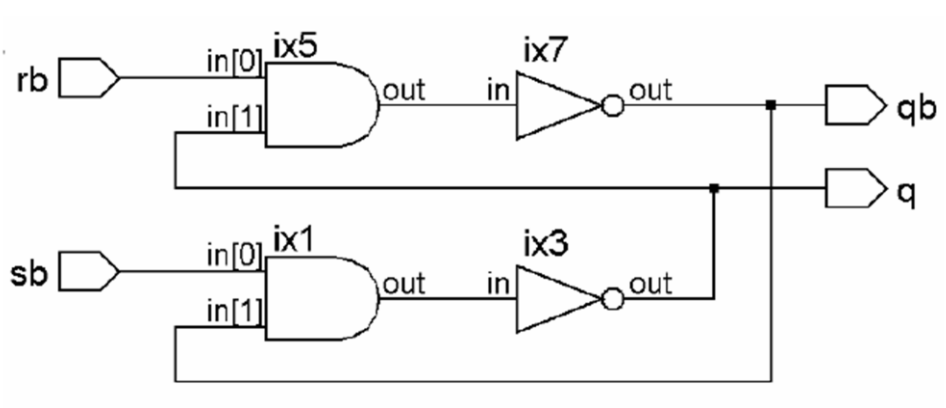
```
module comp(d,a,b,en);
input en;
input[7:0]a,b;
output d;
wire [7:0]c;
wire dd;
xor g1[7:0](c,b,a);
or(dd,c);
notif1(d,dd,en);
endmodule
```



DESIGN OF FLIP-FLOPS WITH GATE PRIMITIVES

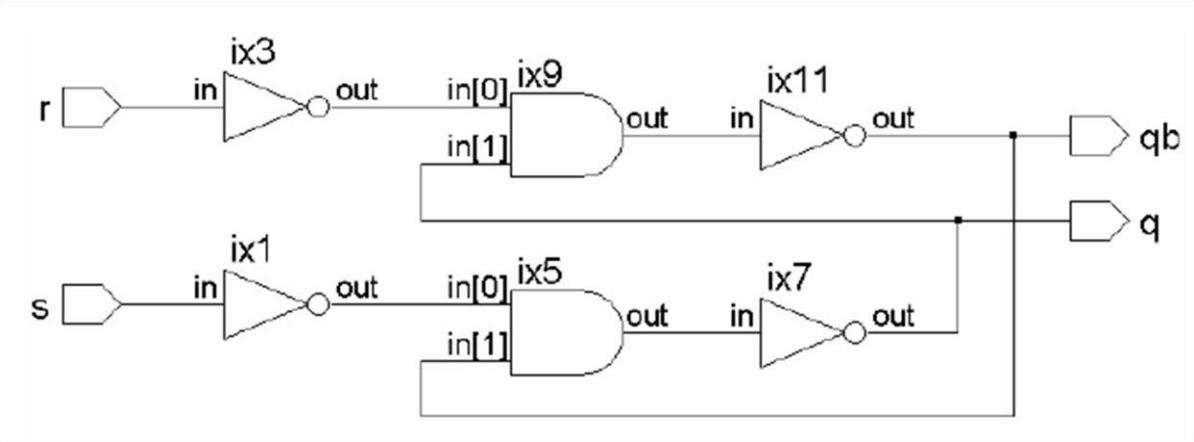
Simple Latch

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



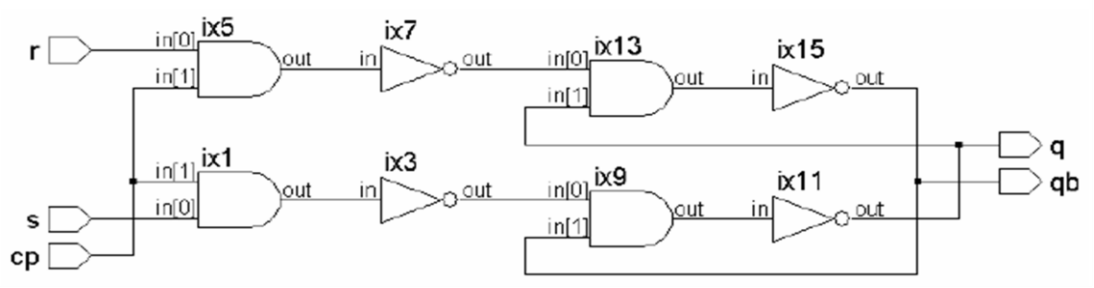
RS Flip-Flop module

```
module srff(s,r,q,qb);
input s,r;  output q,qb;  wire ss,rr;
not(ss,s),(rr,r);
nand(q,ss,qb);
nand(qb,rr,q);
endmodule
```



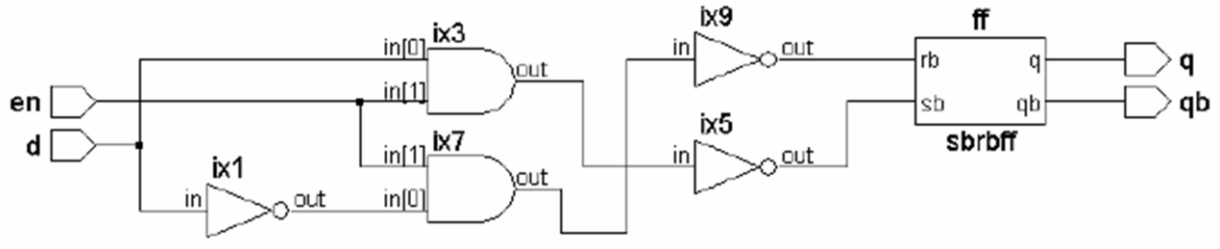
A Clocked RS Flip-Flop module

```
module srffcplev(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
```



D-Latch module

```
module dlatch(en,d,q,qb);
input d,en;  output q,qb;  wire dd;  wire s,r;
not n1(dd,d);
nand (sb,d,en);  nand g2(rb,dd,en);
sbrbff ff(sb,rb,q,qb);
endmodule
```



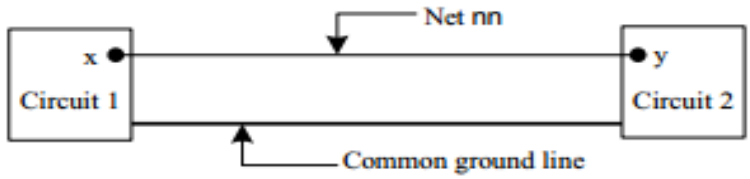
DELAYS

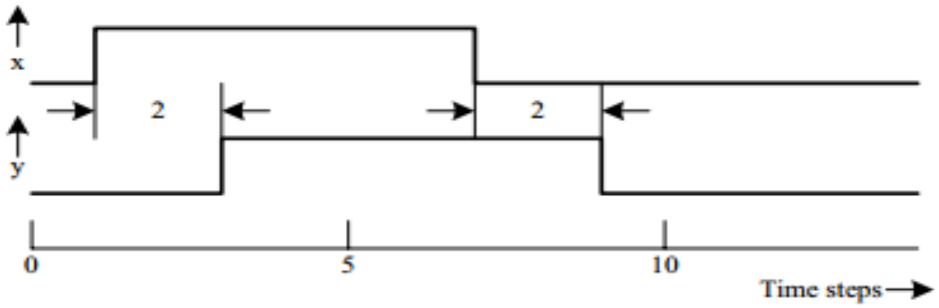
- Verilog has the facility to account for different types of propagation delays of circuit elements. Any connection can cause a delay due to the distributed nature of its resistance and capacitance.
- These manifest as propagation delays in the 0 to 1 transitions and 1 to 0 transitions from input to the output. Such propagation delays can differ for the two types of transitions

Net Delay

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

```
module netdelay(x,y);
input x;
output y;
wire #2 nn;
not (nn,x); //circuit1 in Figure 5.21
buf y = x; //circuit2 in Figure 5.21
endmodule
```

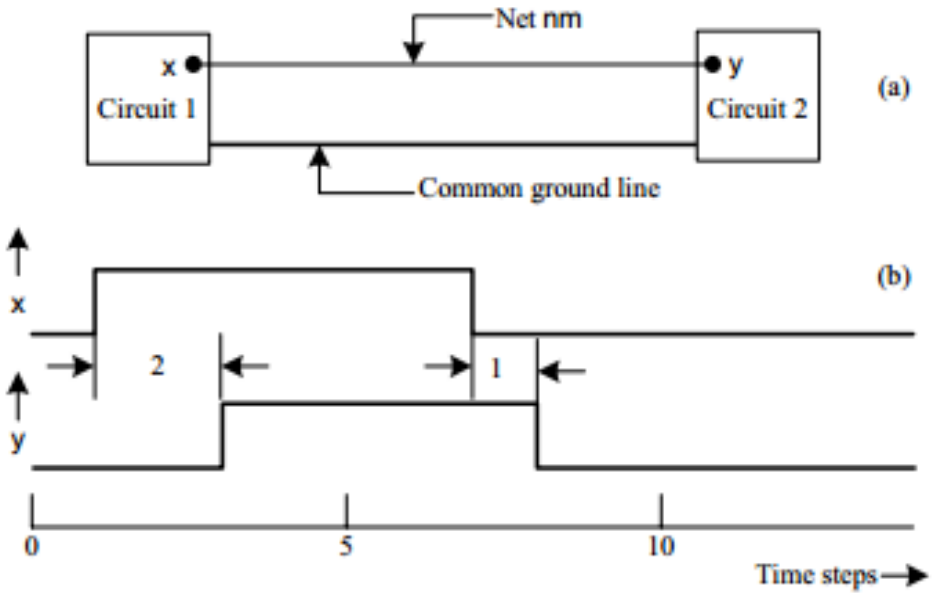




wire # (2, 1) nm;

//the positive (0 to 1) transition has a delay of 2 time steps
//The negative (1 to 0) transition has a delay of 1 time step

```
module netdelay1(x,y);
input x;
output y;
wire #(2,1) nn;
not (nn,x);
y=nn;
endmodule
```



Gate Delay

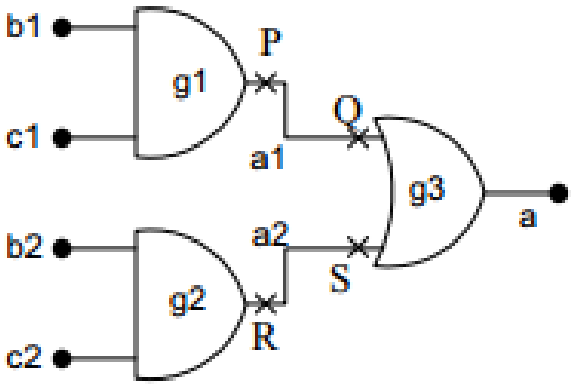
and #3 g(a, b, c);

The above represents an AND gate description with a uniform delay of 3 ns for all transitions from input to output.

and #(2, 1) g(a, b, c);

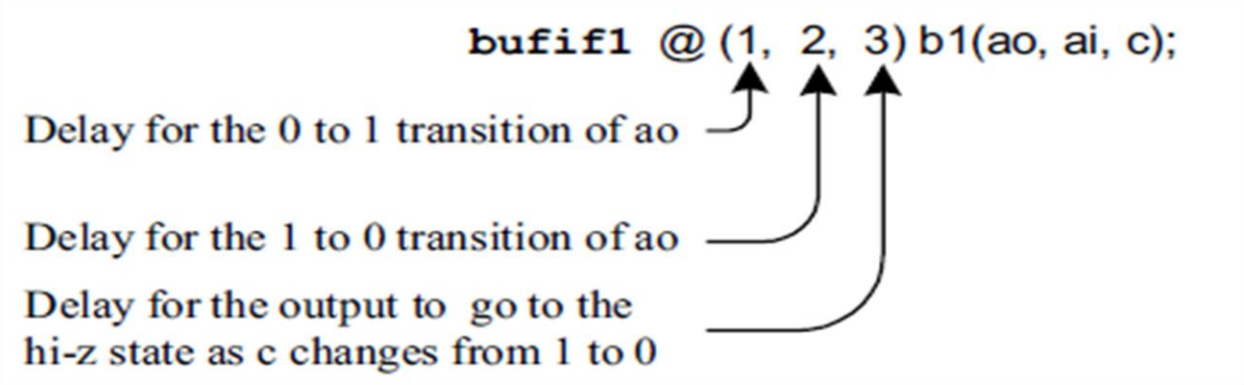
With the above statement the positive (0 to 1) transition at the output has a delay of 2 time steps while the negative (1 to 0) transition has a delay of 1 time step.

```
module gates(b1,b2,c1,c2,a);
input b1,b2,c1,c2;
wire #(2,1)a1,a2;
output a;
and #(3,4)g1(a1,b1,c1);
and #(5,6)g2(a2,b2,c2);
or  #(8,7)g3(a,a1,a2);
endmodule
```



Delays with Tri-state Gates: The instantiation inclusive for a tri-state buffer of the bufif1 type. Three time delay values are specified:

- The first number represents the delay associated with the positive (0 to 1) transition of the output.
- The second number represents the delay associated with the negative (1 to 0) transition of the output.
- The third number represents the delay for the output to go to the hi-Z state as the control signal changes from 1 to 0 (*i.e.*, ON to OFF command).



- Delays for the other tri-state buffers – namely **bufif0**, **notif1** and **notif0** – may be specified in a similar manner.

min, typical, max delays

- It is customary for manufacturers to specify delays and their range in the following manner:

- Each of the delays in a gate primitive or for a net can be specified in terms of these three values.

For example: **and #(2:3:4) g1(a0, a1, a2);** can instantiate an AND gate with the following time delay specifications:

- The 0 to 1 rise time and the 1 to 0 fall time are equal.
- The minimum value of either is 2 time steps. Typical value is 3 time steps and the maximum value is 4 time steps.
- Note that the colon that separates the numbers signifies that the timings specified are the minimum, typical, and maximum values.
- At the time of simulation, one can specify the simulation to be carried out with any of these three delay values. If the same is not specified, the simulation is carried out with the typical delay value.
- The group of minimum, typical, and maximum delay values for the propagation delays can be specified separately for any gate primitive.

Thus an AND gate primitive can be specified as **and #(1:2:3, 2:4:6) g2(b0, b1, b2);**

- Here for the 0 to 1 transition of the output (rise time) the gate has a minimum delay value of 1 ns, a typical value of 2 ns, and a maximum value of 3 ns.
- Similarly, for the 1 to 0 transition (fall time) the gate has a minimum delay value of 2 ns, a typical delay value of 4 ns, and a maximum delay value of 6 ns.
- Such delay specifications can be associated with nets as well as tri-state type gates also.

wire #(1:2:3) a;

/* The net a has a propagation delay whose minimum, typical and maximum values are 1 ns, 2 ns, and 3 ns, respectively*/

bufif1 #(1:2:3, 2:4:6, 3:6:9) g3 (a0, b0, c0);

The different delay values for the buffer are as follows:

- The output rise time (0 to 1 transition) has a minimum value of 1 ns, a typical value of 2 ns and a maximum value of 3 ns.
- The output fall time (1 to 0 transition) has a minimum value of 2 ns, a typical value of 4 ns and a maximum value of 6 ns.
- The output turn-off time (1 to 0) has a minimum value of 3 ns, a typical value of 6 ns, and a maximum value of 9 ns. */

STRENGTHS AND CONTENTION RESOLUTION

- When the outputs of two gates are joined together, the signal level is decided by the relative magnitudes of the source impedances.
- Sometimes a disparity between the impedances is intentionally introduced to minimize circuit hardware.
- Effects of such differences in the impedances are indirectly introduced in design descriptions by assigning "strengths" to specific signals.

Name	supply	strong	pull	weak	High impedance
Abbreviations	su1 su0	st1 st0	pu1 pu0	we1 we0	HiZ1 HiZ0
Strength	Strongest			Weakest	

```
buf  (supply1,  pull0)  (o, i);
```

Strength of 1 state in the output — Strength of 0 state in the output

STRENGTH CONTENTION in GATE PRIMITIVES

- When two signals of opposite polarity and differing strengths drive a line, the output status is decided by the stronger signal.
- However, if the signals are of equal strength, the output is indeterminate.

Logic value of input i1	Logic value of input i2	Logic value of output o	Remarks
0	0	0	No contention
0	1	1	Contention; the stronger signal – i2 – prevails
1	0	1	Contention; the stronger signal – i1 – prevails
1	1	1	No contention

```
module contres(o,i1,i2);
input i1,i2;
output o;
buf(supply1,pull0)g1(o,i1), g2(o,i2);//note that the
endmodule// same net is driven by both the gates.
```

Net Charges

- Whenever a net is driven by a signal, it takes the logic value of the signal.
- When the signal source is tri-stated, the net too gets tri-stated.
- In practice the net can have a capacitor associated with it, which can store the signal level even after the signal source dries up (*i.e.*, tri-stated).
- Such nets are declared with the keyword **trireg.**

Name	large	medium	small
Strength	Strongest		Weakest

- A **trireg** net can be driven with possibilities of contention from two or more sources
- A **trireg** net can be in one of two possible states only:

***Driven state*:** When driven by a source or multiple sources, the net assumes the strength of the source.

***Capacitive state*:** When the driven source (sources) is (are) tri-stated, the net retains the last value it was in – by virtue of the capacitance associated with it. The value can be 0, 1 or **x** (but not the high impedance value).

- When in the capacitive state, a net can have a storage strength associated with it. Three such storage strengths are possible – namely **large, medium,** and **small**.
- When a storage strength is not specified, it is assigned the default value – **medium**.
- For a **trireg** net one cannot assign storage strength capacity separately for the 0 and the 1 states.

Net Storage

Exercise: As long as the signal control = 1, the signal out follows the signal in. When control goes to 0, out is disconnected from the input and it ''floats.'' It retains the last value due to the capacitance storage capacity. The storage strength is **medium,** signifying a medium value of capacitance.

```
module charge(out,in,control);
output out;
trireg(medium)out;
input in,control;
bufif1 g1(out,in,control);
endmodule

module tst charge; //TESTBENCH
reg in, control;
charge c1(out,in,control);
initial
     begin
     in =0;control =0;//when control=0 output is x
     #2 control =0;in =0;
     #2 control =1;in =0;
     #2 control =1;in =1;
     #2 control =0;in =0; // output is retained at
     end // the last value
initial $monitor($time ," in= %b ,control = %b , out=
%b " ,in,control,out);
initial #40$stop;
endmodule
```

Signal strength names and weights

Signal strength name	Strength level
Supply (drive)	Strongest 7
Strong (drive)	6
Pull (drive)	5
Large (capacitance)	4
Weak (drive)	3
Medium (capacitance)	2
Small (capacitance)	Weakest 1
High impedance	0

NET Types: wire is possibly the simplest type of net declaration.

wand and wor

- Strengths on nets can be decided in ways other than a direct declaration also.
- 'wand' and 'wor' are two types of net declarations for such contention resolution.

- wand is a wire declaration, which resolves to AND logic in case of contention.
- wor is a wire declaration, which resolves to OR logic in case of a contention.

- All synthesizers support wire. Triand, trior, tri0, and tri1 may not be supported by some synthesizers.

Logic value of i1	Logic value of i2	Logic value of o	Remarks
0	0	0	No contention
0	1	0	Contention resolved according to **wand** declaration
1	0	1	Contention resolved by the stronger signal
1	1	1	No contention

TRI

- The keyword tri has a function identical to that of wire.
- When a net is driven by more than one tri-state gate, it is declared as tri rather than as wire. The distinction is for better clarity. Similarly, Triand and trior are the counterparts of wand and wor, respectively.
- If the output of a tri-state buffer is to be pulled up to the 1 state when tri-stated, it is declared as net tri1. Similarly, it is declared as tri0 if it is to be pulled down to 0 state when tri-stated.
  Reset, *Chip Enable* and similar signals can be pulled up or down as required with tri0 or tri1; this signifies the normal status –that is, the chip is disabled or the reset is disabled.
- Similarly, the *reset* can be activated for a specified period to reset the chip; subsequently, the reset can be deactivated to restore normal operation of the chip.
- All synthesizers support wire. Triand, trior, tri0, and tri1 may not be supported by some synthesizers.

Supply 0 and Supply 1

- supply0 and supply1 are the keywords signifying the high- and low-side supplies.
- Nets to be connected to the Vcc supply are declared as supply1, and those to be grounded are declared as supply0.

DESIGN OF BASIC CIRCUITS: Elementary gates are the basic building blocks of all digital circuits – whether combinational, sequential, or involved versions combining both. Any digital circuit however involved it may be, can be realized in terms of gate primitives. The step-by-step procedure to be adopted may be summarized as follows:

1. Draw the circuit in terms of the gates and Name gates and signals.
2. Using the same nomenclature as above, do the design description.
3. As the functional blocks like encoder, decoder, half-adder, full-adder, *etc*., get more and more involved, treat each as a building block with corresponding inputs and outputs.
4. Make more involved circuits in terms of the building blocks – as far as possible. Each block within another block manifests as an instantiation of one module within another.