HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Structural Modeling

Materials partly adapted from "Digital System Designs and Practices Using Verilog HDL and FPGAs," M.B. Lin.

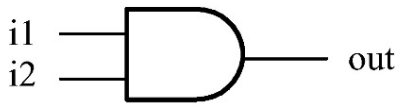
NCKU EE
LPHP Lab

OUTLINE

- Verilog Basics
- Structural Modeling
- Dataflow Modeling

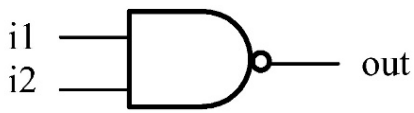
STRUCTURAL MODELING

BASICS GATES



and		i2			
		0	1	x	z
i1	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

(a) and gate

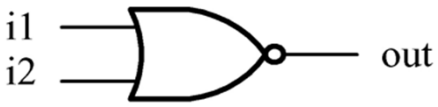


nand		i2			
		0	1	x	z
i1	0	1	1	1	1
	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x

(b) nand gate

or		i2			
		0	1	x	z
i1	0	0	1	x	x
	1	1	1	1	1
	x	x	1	x	x
	z	x	1	x	x

(c) or gate



nor		i2			
		0	1	x	z
i1	0	1	0	x	x
	1	0	0	0	0
	x	x	0	x	x
	z	x	0	x	x

(d) nor gate

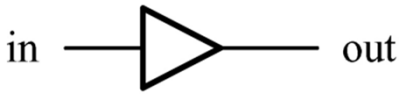
xor		i2			
		0	1	x	z
i1	0	0	1	x	x
	1	1	0	x	x
	x	x	x	x	x
	z	x	x	x	x

(e) xor gate

xnor		i2			
		0	1	x	z
i1	0	1	0	x	x
	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

(f) xnor gate

BASICS GATES

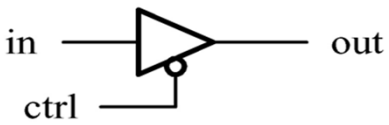


in	out
0	0
1	1
x	x
z	x

(a) buffer

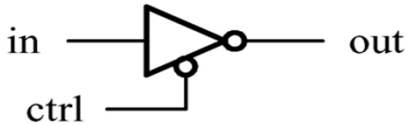
in	out
0	1
1	0
x	x
z	x

(b) not gate



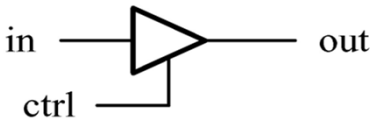
bufif0		ctrl			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	x	z	x	x

(a) bufif0



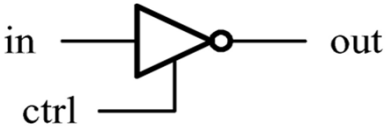
notif0		ctrl			
		0	1	x	z
in	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	x	x

(b) notif0



bufif1		ctrl			
		0	1	x	z
in	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

(c) bufif1



notif1		ctrl			
		0	1	x	z
in	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

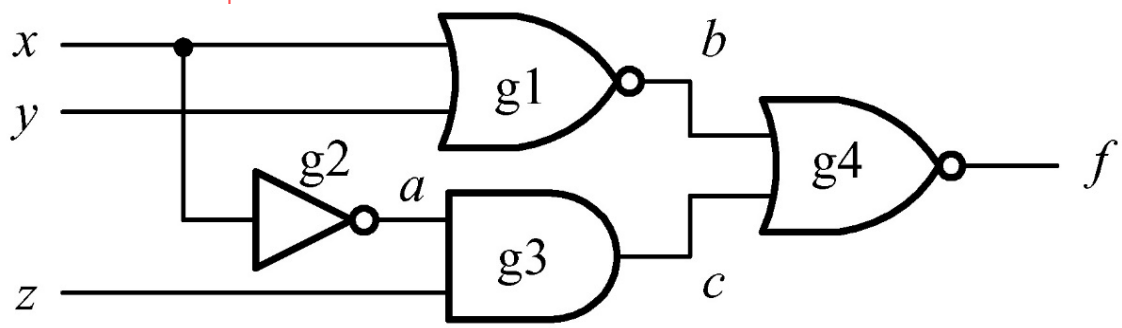
(d) notif1

INSTANTIATION OF BASIC GATES

- To instantiate and/or gates

  gatename [instance_name](output, input1, input2, ..., input*n*);
  - instance_name is optional

```
module basic_gates (x, y, z, f) ;
input   x, y, z;
output f ;
wire a, b, c;
// Structural modeling
    nor  g1  (b, x, y);
    not  g2  (a, x);
    and g3  (c, a, z);
    nor g4  (f, b, c);
endmodule
```



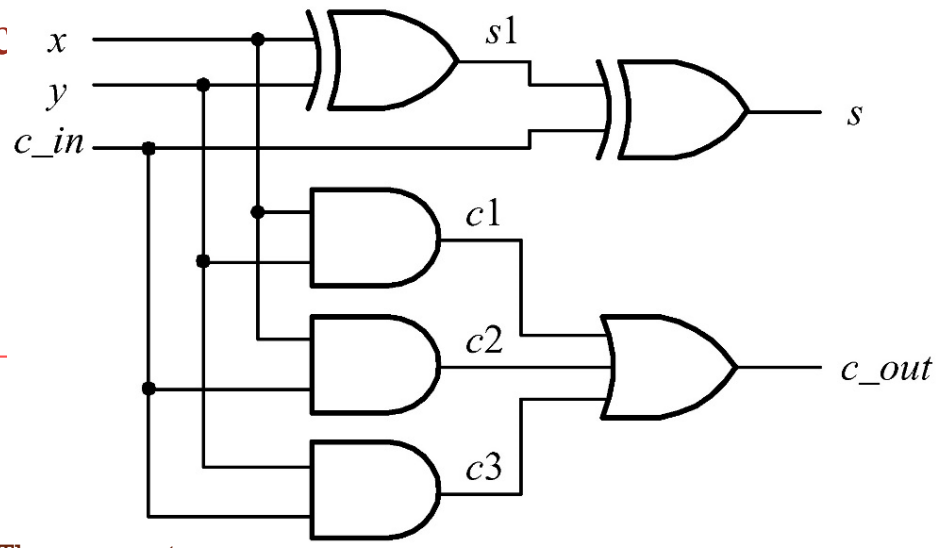
ARRAY OF INSTANCES

- Array instantiations may be a synthesizer dependent!
  - Suggestion: check this feature before using the synthesizer

```
wire [3:0] out, in1, in2;
// basic array instantiations of nand gate.
nand n_gate[3:0] (out, in1, in2);

// this is equivalent to the following:
nand n_gate0 (out[0], in1[0], in2[0]);
nand n_gate1 (out[1], in1[1], in2[1]);
nand n_gate2 (out[2], in1[2], in2[2]);
nand n_gate3 (out[3], in1[3], in2[3]);
```

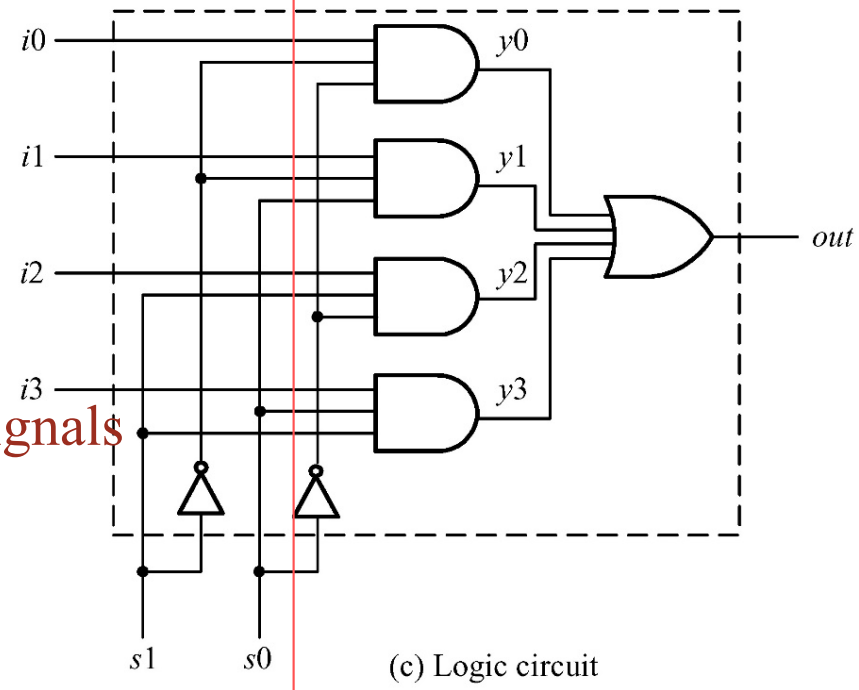
AN EXAMPLE --- A 1-BIT FULL ADDER

```
module full_adder_structural(x, y, c_in, s, c_out);
// I/O port declarations
input    x, y, c_in;
output   s, c_out;
wire     s1, c1, c2, c3;
// Structural modeling of the 1-bit full adder.
  xor   xor_s1(s1, x, y);      // compute sum.
  xor   xor_s2(s, s1, c_in);
  and   and_c1(c1, x, y);      // compute c
  and   and_c2(c2, x, c_in);
  and   and_c3(c3, y, c_in);
  or    or_cout(c_out, c1, c2, c3);
endmodule
```



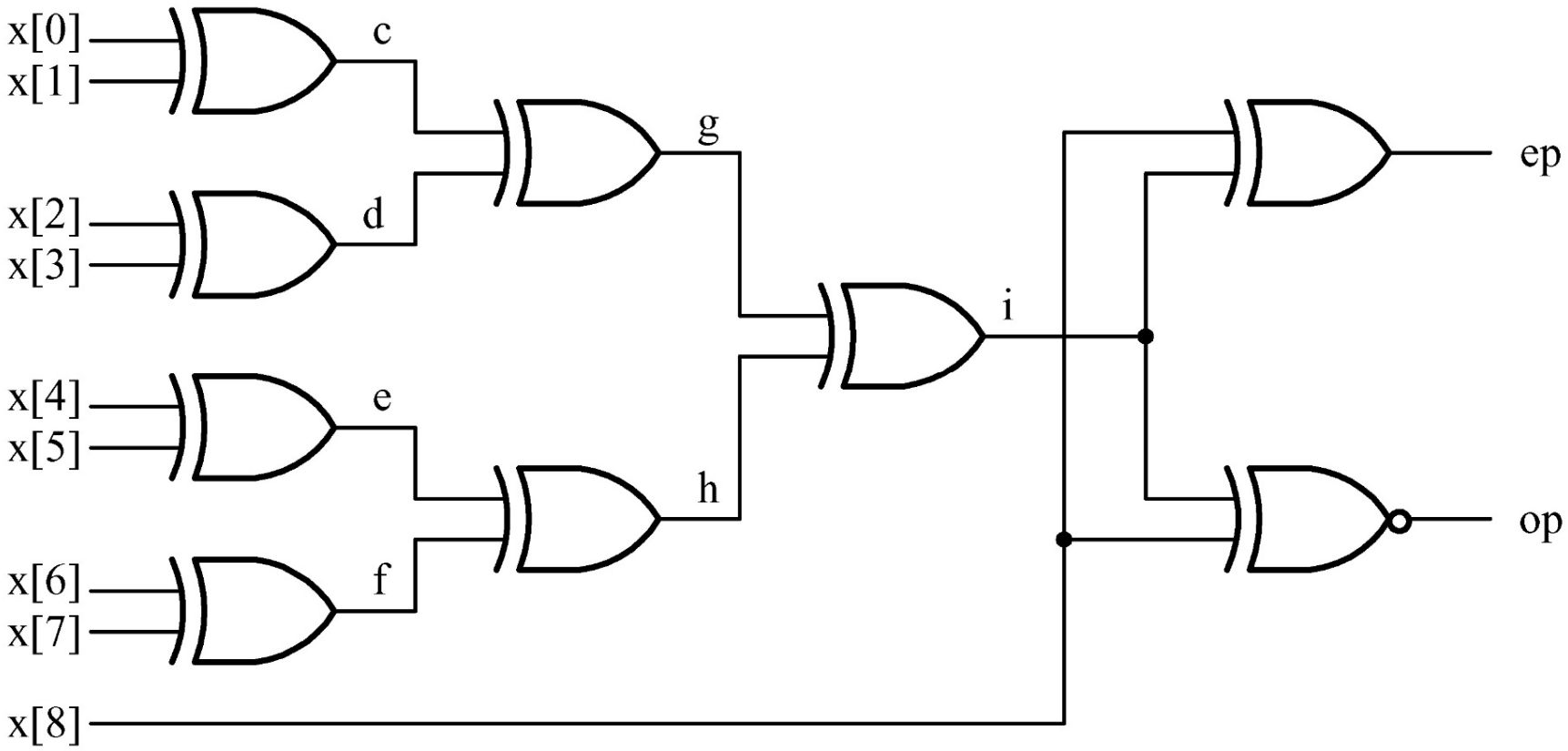
AN EXAMPLE --- A 4-TO-1 MULTIPLEXER

```
module mux4_to_1_structural (i0, i1, i2, i3, s1, s0,
out);
input   i0, i1, i2, i3, s1, s0;
output  out;
wire    s1n, s0n;   // Internal wire
wire    y0, y1, y2, y3;
// Gate instantiations
   not (s1n, s1);    // Create s1n and s0n signals
   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
```



(c) Logic circuit

AN EXAMPLE --- A 9-BIT PARITY GENERATOR



AN EXAMPLE --- A 9-BIT PARITY GENERATOR

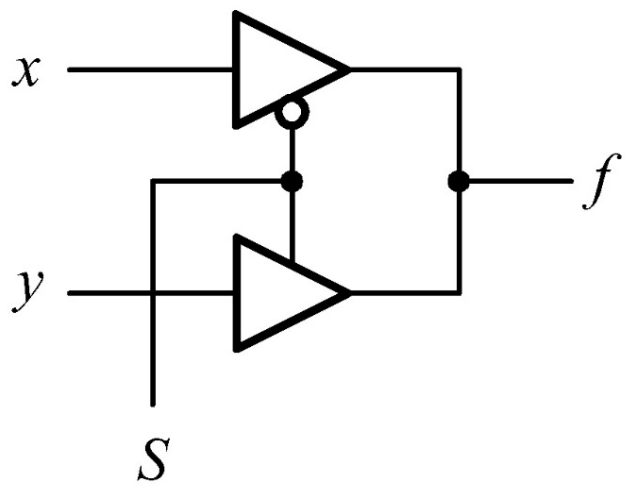
```
module parity_gen_9b_structural(x, ep, op);
// I/O port declarations
input     [8:0]  x;
output    ep, op;
wire      c, d, e, f, g, h, j;
   xor xor_11(c, x[0], x[1]);  // first level
   xor xor_12(d, x[2], x[3]);
   xor xor_13(e, x[4], x[5]);
   xor xor_14(f, x[6], x[7]);
   xor xor_21(g, c, d);          // second level
   xor xor_22(h, e, f);
   xor xor_31(i, g, h);          // third level
   xor xor_ep(ep, i, x[8]);      // fourth level
   xnor xnor_op(op, i, x[8]);
endmodule
```

INSTANTIATION OF TRISTATE BUFFERS

- To instantiate tristate buffers

  buf_name[*instance_name*](output, input, control);

  - The instance_name is optional




```
// 2-to-1 mux
module two_to_one_mux_tristate (x, y, s, f);
input   x, y, s;
output f;
tri   f; // internal declaration
// data selector body
    bufif0 b1 (f, x, s);
    bufif1 b2 (f, y, s);
endmodule
```

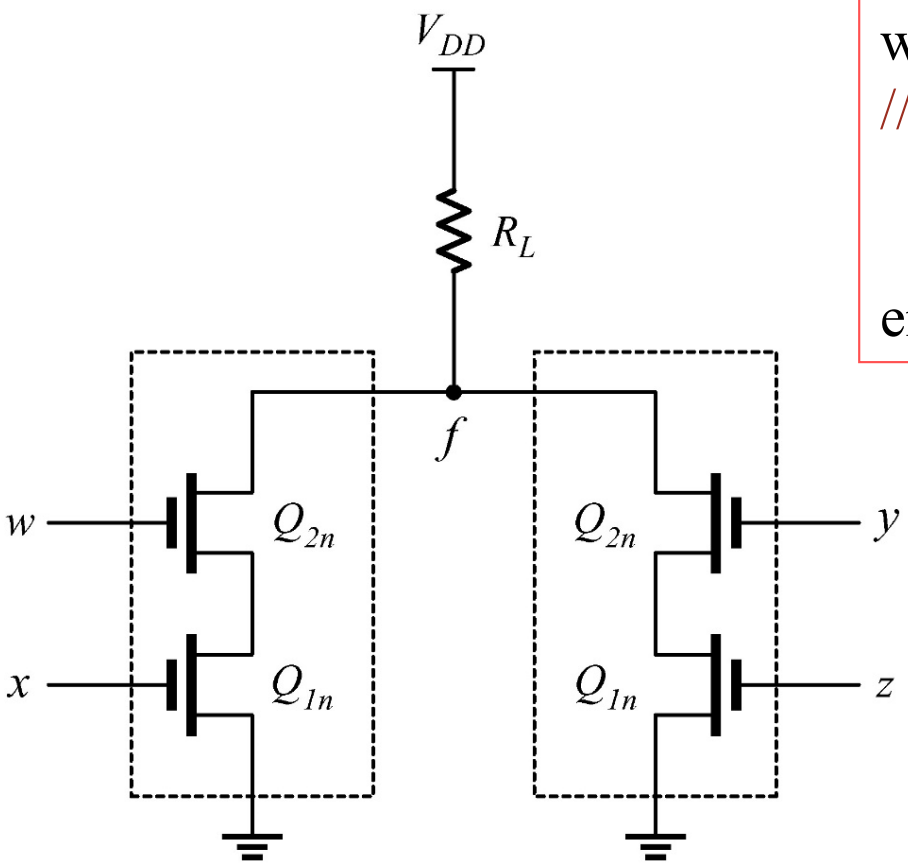
WAND/TRIAND AND WOR/TRIOR

triand/ wand	0	1	x	z
0	0	0	0	0
1	0	1	x	1
x	0	x	x	x
z	0	1	x	z

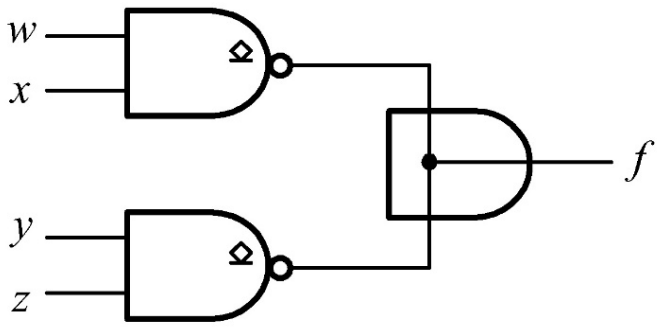
trior/ wor	0	1	x	z
0	0	1	x	0
1	1	1	1	1
x	x	1	x	x
z	0	1	x	z

WIRED AND GATES

```
module open_drain (w, x, y, z, f);
input   w, x, y, z;
output f;
wand   f;   // internal declaration
// wired AND logic gate
      nand n1 (f, w, x);
      nand n2 (f, y, z);
endmodule
```



(a) Circuit



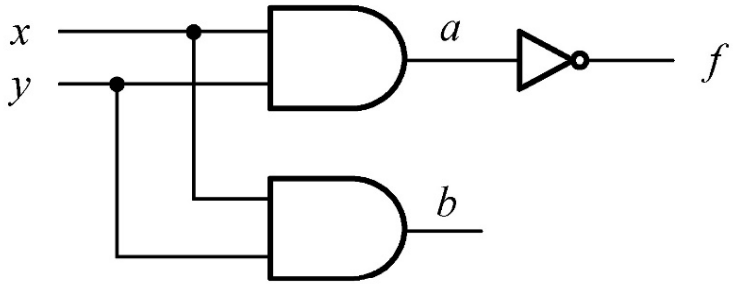
$f = (wx)'(yz)'$

(b) Logic symbol

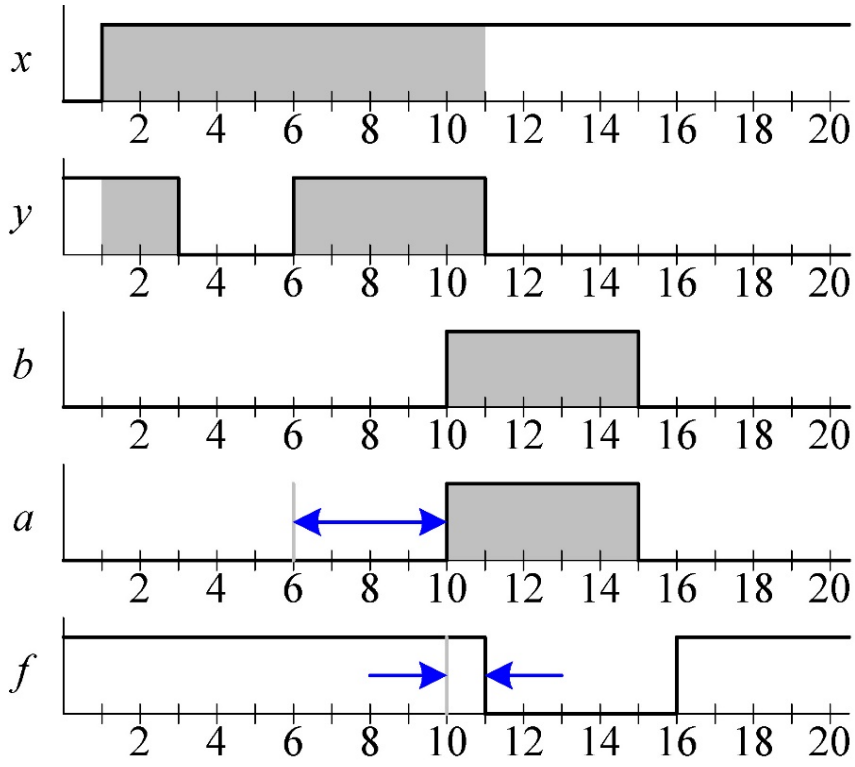
DELAY MODELS

- Inertial delay model
  - To model gate delays
  - The default gate delay is 0
  - The default delay model for HDL (Verilog HDL and VHDL)
- Transport delay model
  - To model net (i.e. wires) delays
  - The default delay of a net is zero

THE EFFECTS OF INERTIAL DELAYS

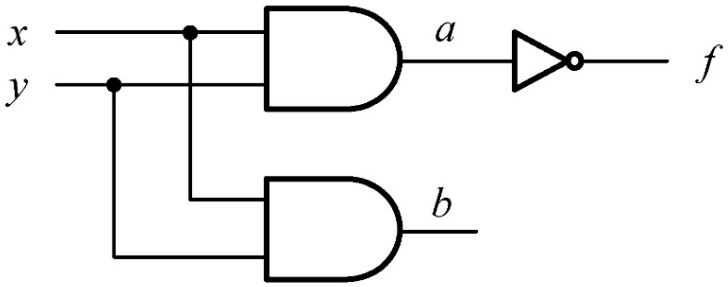



```
wire a;
and  #4 (b, x, y); // inertial delay
and  #4 (a, x, y);
not  #1 (f, a);
```

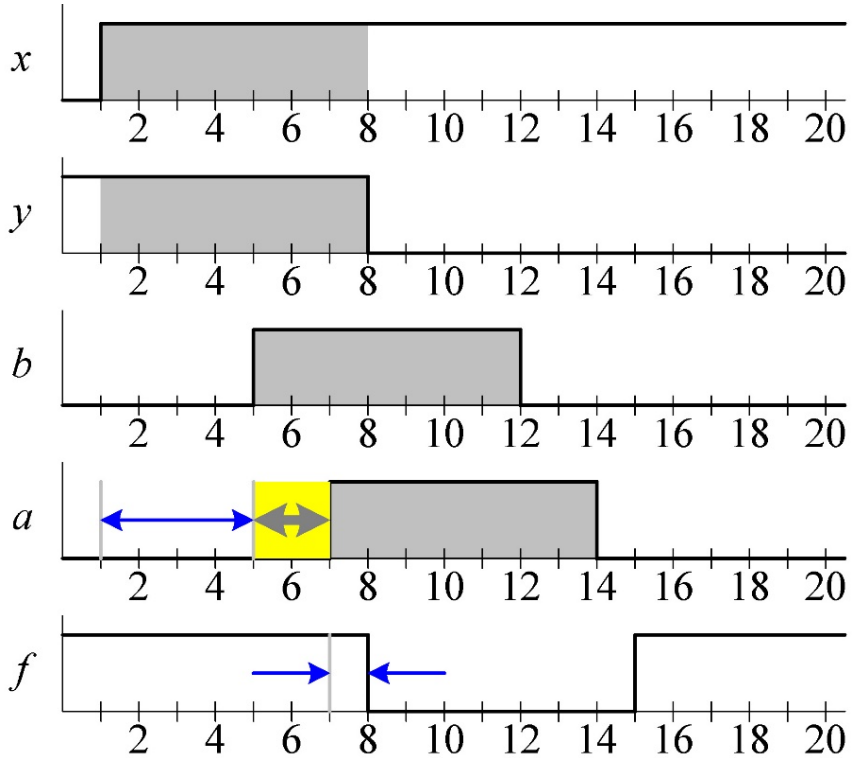


Inertial delay

THE EFFECTS OF TRANSPORT AND INERTIAL DELAYS




```
wire #2 a;          // transport delay
and  #4 (b, x, y);// inertial delay
and  #4 (a, x, y);
not  #1 (f, a);
```



Inertial delay

Transport delay

GATE DELAY SPECIFICATIONS

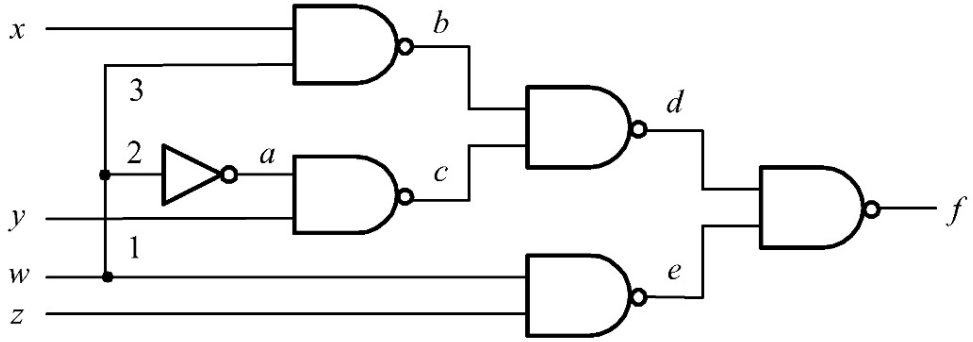
- Specify propagation delay only
  - gatename #(prop_delay) [instance_name](output, in_1, in_2,…);
- Specify both rise and fall times
  - gatename #(t_rise, t_fall) [instance_name](output, in_1, in_2,…);
- Specify rise, fall, and turn-off times (tristate buffers)
  - gatename #(t_rise, t_fall, t_off) [instance_name](output, in_1, in_2,…);

Delay specifier: min:typ:max

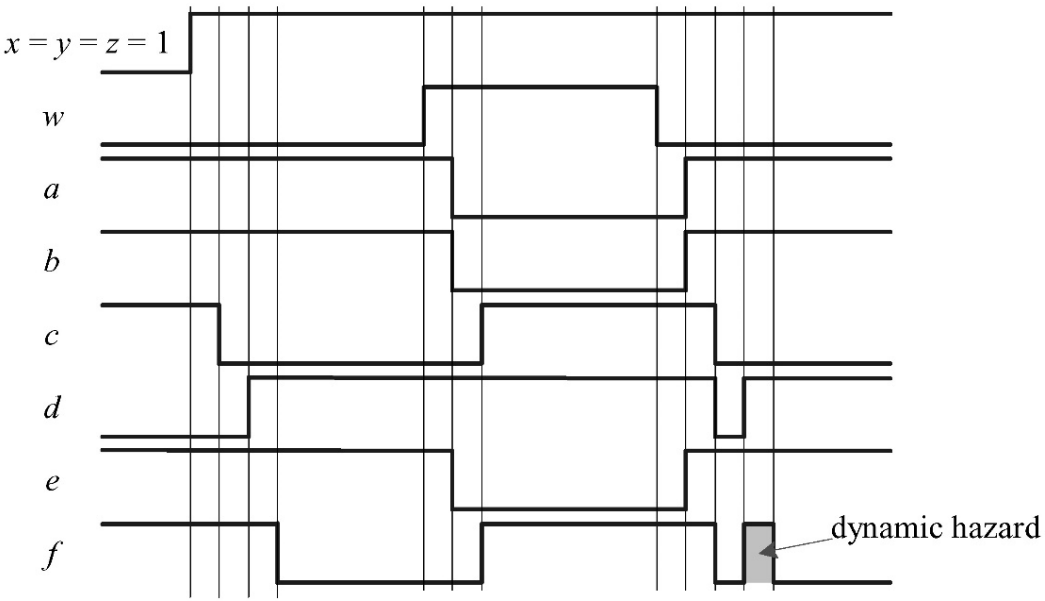
GATE DELAYS SPECIFICATIONS

```
// Only specify one delay
and  #(5)  a1 (b, x, y);

// Only specify one delay using min:typ:max
not  #(10:12:15) n1 (a, x);

// Specify two delays using min:typ:max
and  #(10:12:15, 12:15:20) a2 (c, a, z);

// Specify three delays using min:typ:max
bufif0   #(10:12:15, 12:15:20, 12:13:16) buf1 (f, b, c);
```

A DYNAMIC HAZARD EXAMPLE



(a) Logic circuit



(b) Timing

A DYNAMIC HAZARD EXAMPLE

```
// dynamic hazard example
module hazard_dynamic(w, x, y, z, f);
input   w, x, y, z;
output f;
// internal declaration
wire  a, b, c, d, e;
// logic circuit body
    nand  #5  nand1 (b, x, w);
    not     #5  n1     (a, w);
    nand  #5  nand2 (c, a, y);
    nand  #5  nand3 (d, b, c);
    nand  #5  nand4 (e, w, z);
    nand  #5  nand5 (f, d, e);
endmodule
```

A DYNAMIC HAZARD EXAMPLE

```
`timescale 1ns / 1ns
module hazard_dynamic_tb;
reg   w, x, y, z;
wire f;
// Unit Under Test port map
    hazard_dynamic UUT ( .w(w),.x(x),.y(y),.z(z),.f(f));
initial begin
                w = 1'b0;   x = 1'b0;   y = 1'b0;  z = 1'b0;
        #5       x = 1'b1;   y = 1'b1;   z = 1'b1;
        #30     w = 1'b1;
        #20     w = 1'b0;
        #190  $finish;
    end
initial $monitor($realtime,,"ns %h %h %h %h %h ",w,x,y,z,f);
endmodule
```