COMP311-1 Logic Circuit Design
Chap. 5

Instructor: Taigon Song (송대건)

2019 Fall

Post-Homework #2

Design a 1:4 demux

- 4 bit input
- 4 bit output
    - Outputs must be in the name of ‘out(last 4 digits of your ID)_X
        - (e.g., out0885_0, out0885_1…)

- Use any design methodology you prefer
- Use $random for your input definition
- Use ‘for’ statement to change the select values

Levels of Design Abstraction

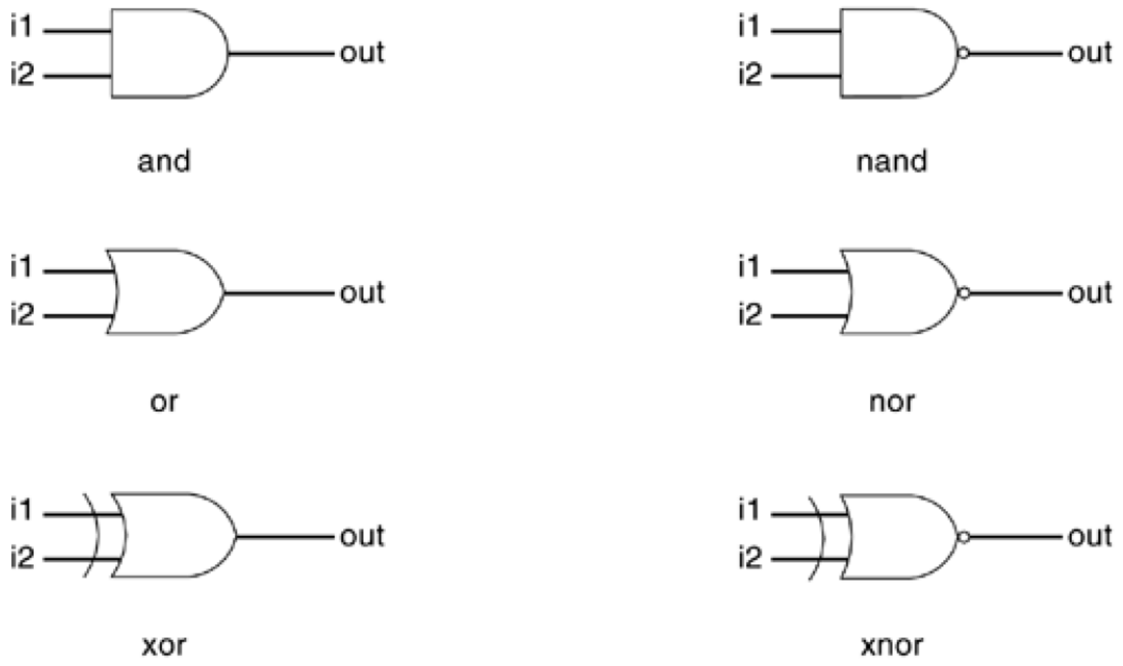
- Behavioral or algorithmic level
  - Highest level of abstraction. Almost as similar to C programming
- Dataflow level
  - How data flows between HW registers and how data is processed
- Gate level
  - AND, OR, INV…etc
- Switch level
  - Transistor level

5.1 Gate Types

5.1.1. AND/OR Gates

- One scalar output and multiple scalar inputs
  - Output first, then inputs
  - Output evaluated as soon as one of the input changes

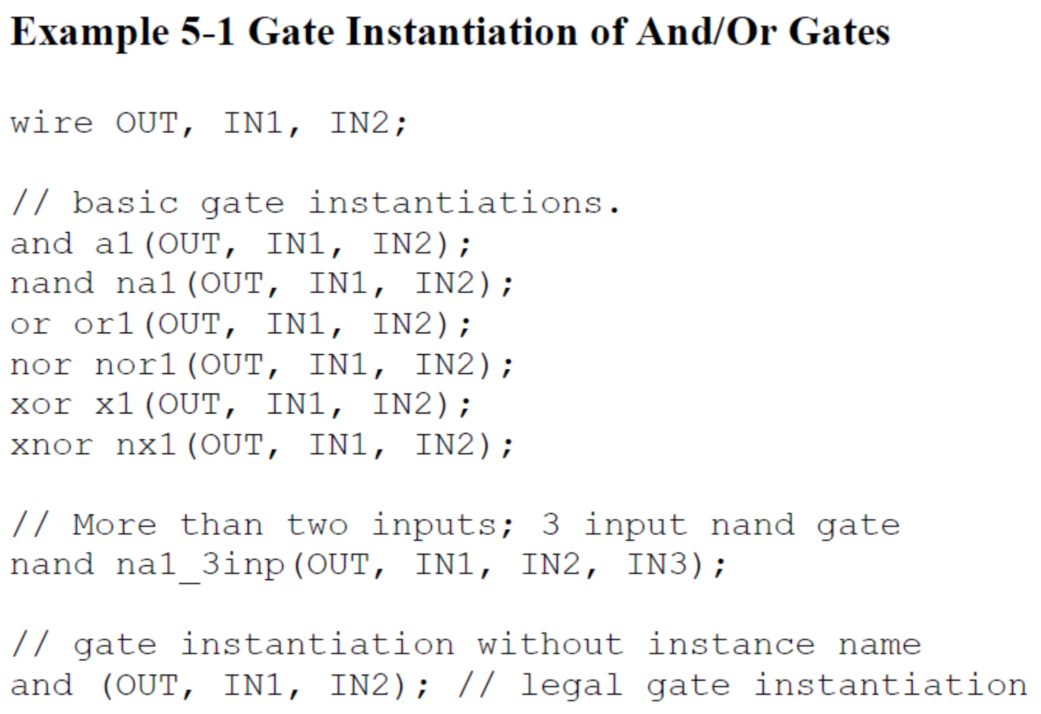
Figure 5-1. Basic Gates



5.1 Gate Types

5.1.1. AND/OR Gates

- Instance name is not a ‘must’ for primitives
  - However, this class asks the students to specify names for their instances for debugging purposes.



Example 5-1 Gate Instantiation of And/Or Gates

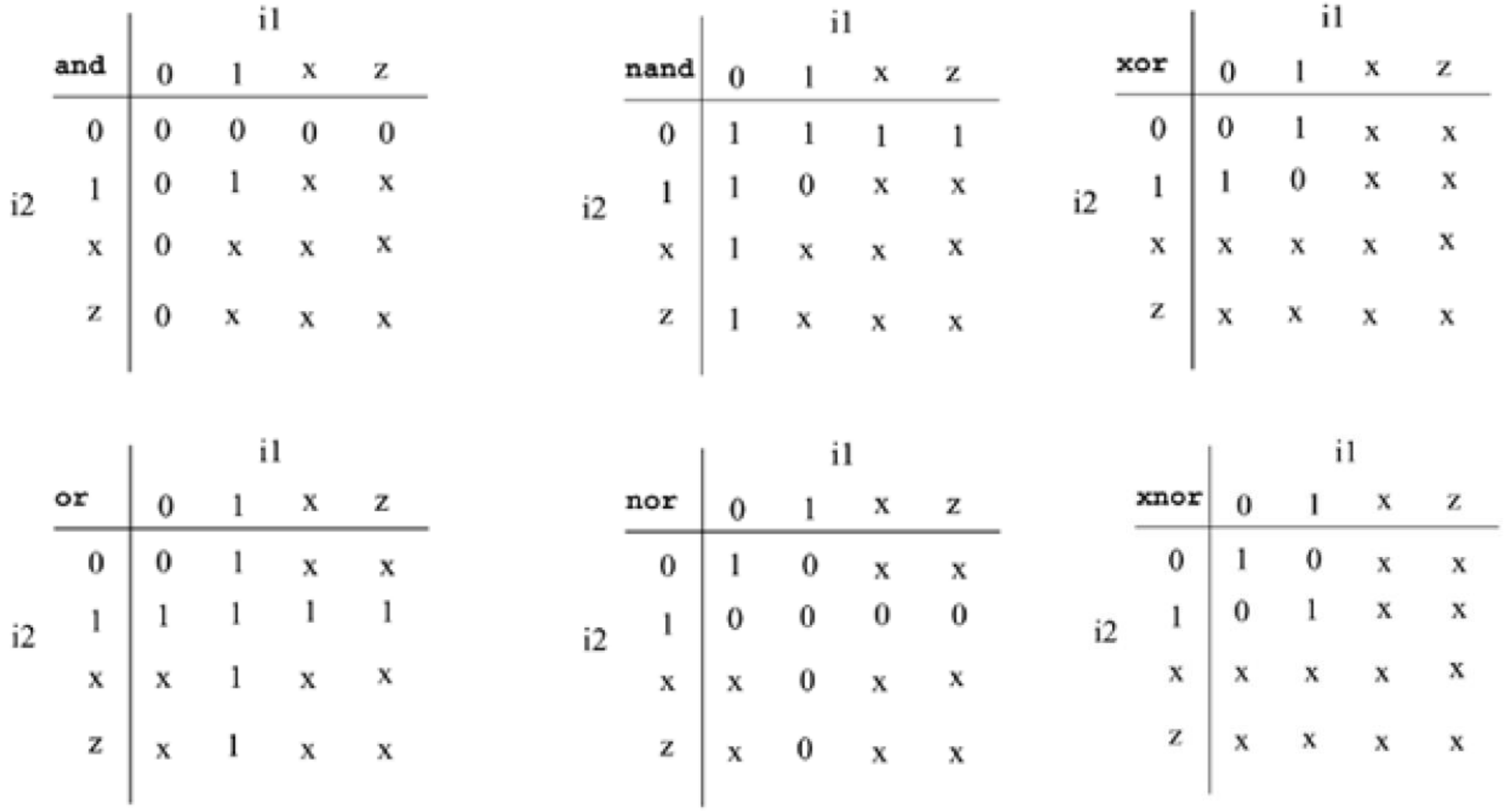
```
wire OUT, IN1, IN2;

// basic gate instantiations.
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
nor nor1(OUT, IN1, IN2);
xor x1(OUT, IN1, IN2);
xnor nx1(OUT, IN1, IN2);

// More than two inputs; 3 input nand gate
nand na1_3inp(OUT, IN1, IN2, IN3);

// gate instantiation without instance name
and (OUT, IN1, IN2); // legal gate instantiation
```

5.1 Gate Types

5.1.1. AND/OR Gates

- Truth table of these gates

Table 5-1. Truth Tables for And/Or



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

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

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

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

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

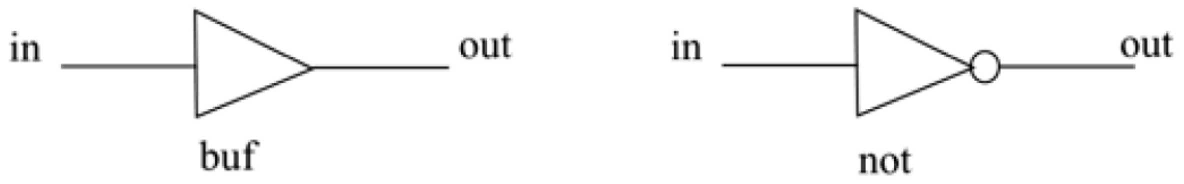
		i1			
	xnor	0	1	x	z
	0	1	0	x	x
i2	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

5.1 Gate Types

5.1.2 BUF/NOT Gates

- One scalar input and one (or more) scalar outputs
  - Last port in the list is connected to the input

Figure 5-2. Buf and Not Gates



Example 5-2 Gate Instantiations of Buf/Not Gates

```
// basic gate instantiations.
buf b1(OUT1, IN);
not n1(OUT1, IN);

// More than two outputs
buf b1_2out(OUT1, OUT2, IN);

// gate instantiation without instance name
not (OUT1, IN); // legal gate instantiation
```

5.1 Gate Types

5.1.2 BUF/NOT Gates

- Truth table

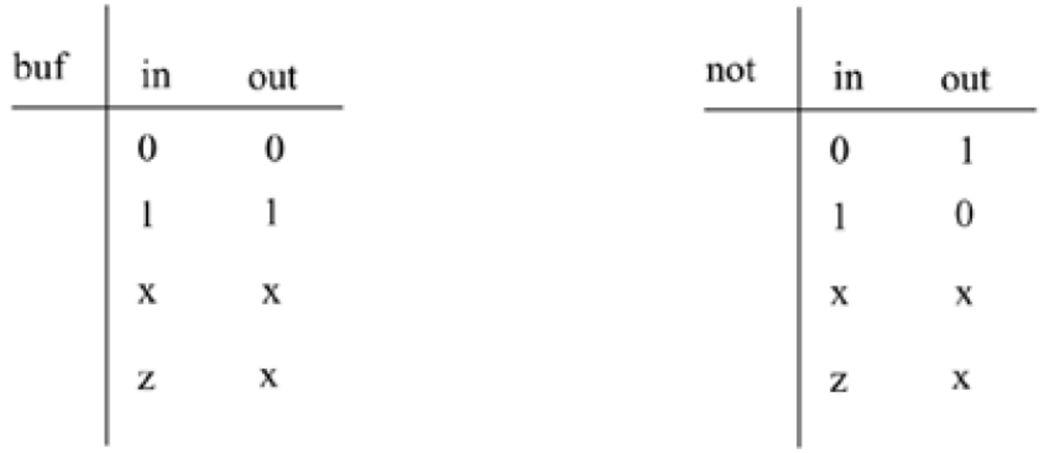


Table 5-2. Truth Tables for Buf/Not Gates

buf	in	out
	0	0
	1	1
	x	x
	z	x

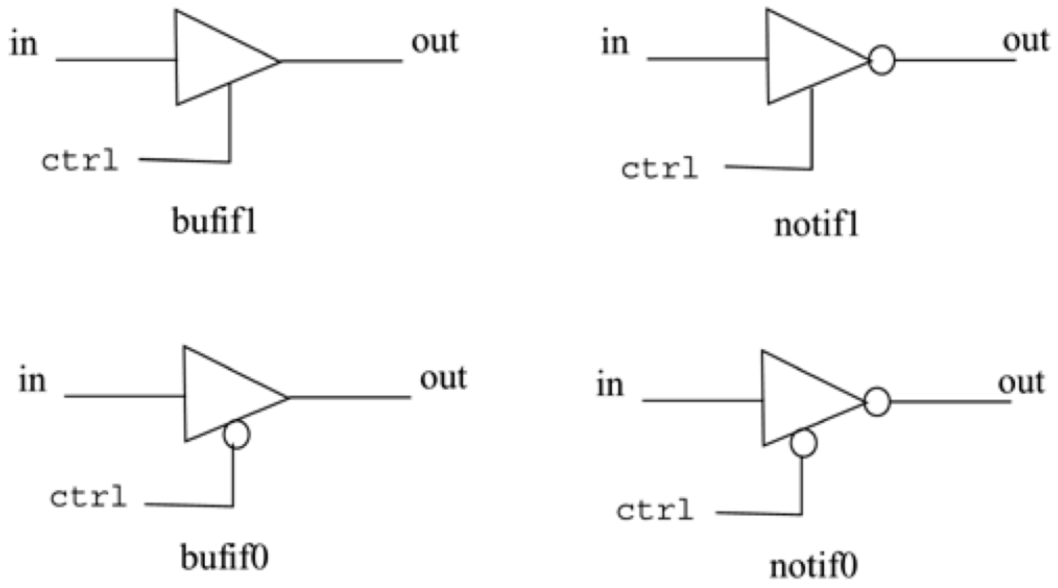
not	in	out
	0	1
	1	0
	x	x
	z	x

5.1 Gate Types

5.1.2 BUF/NOT Gates

- BUFIF/NOTIF
  - Gates with an additional control signal
  - These gate propagate only if their control signal is inserted
  - High-Z when control signal deasserted

Figure 5-3. Gates Bufif and Notif

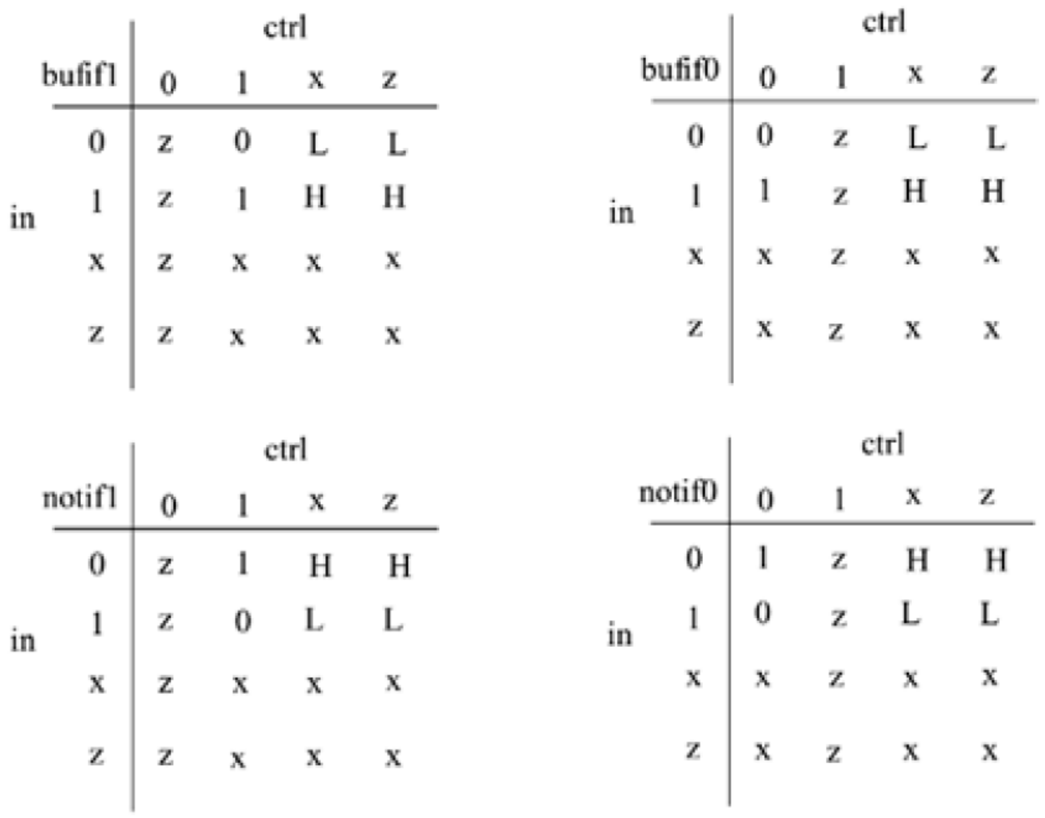


5.1 Gate Types

5.1.2 BUF/NOT Gates

- BUFIF/NOTIF
  - Truth table

Table 5-3. Truth Tables for Bufif/Notif Gates



		ctrl			
	bufif1	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

		ctrl			
	bufif0	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

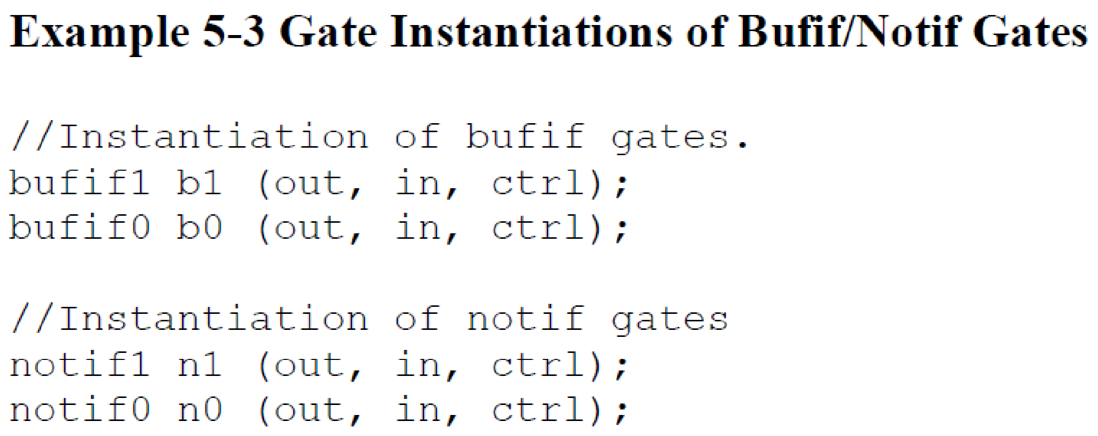
		ctrl			
	notif1	0	1	x	z
in	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

		ctrl			
	notif0	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

5.1 Gate Types

5.1.2 BUF/NOT Gates

- BUFIF/NOTIF
  - Used when multiple signal drives the BUF/INV



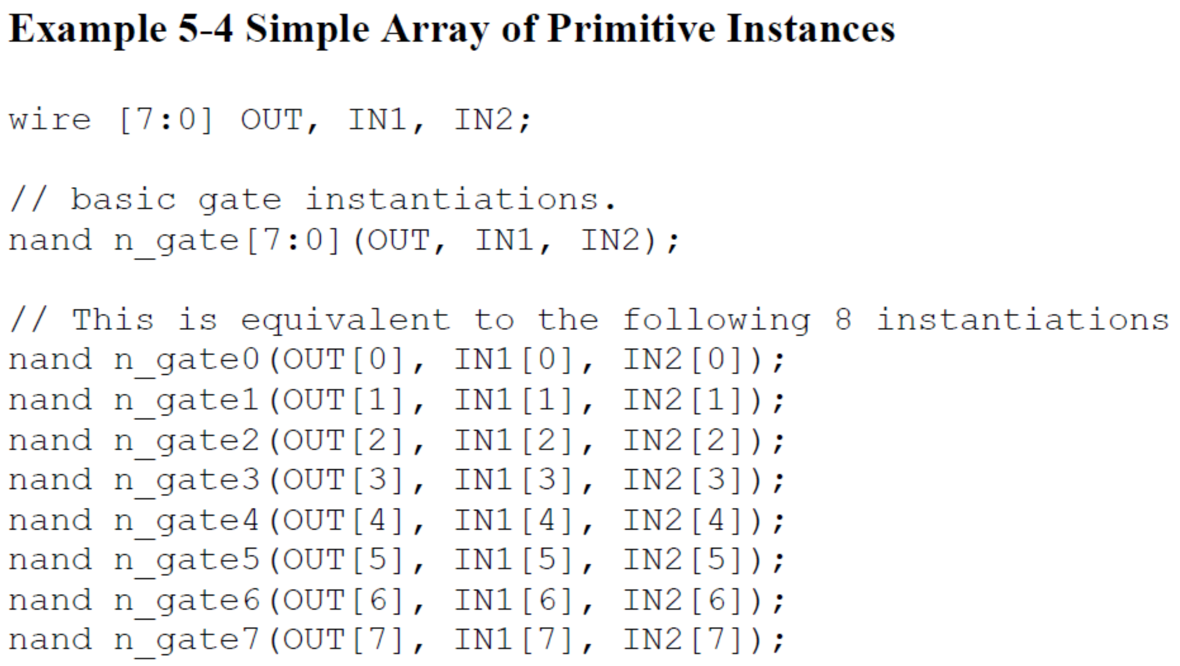
Example 5-3 Gate Instantiations of Bufif/Notif Gates

```
//Instantiation of bufif gates.
bufif1 b1 (out, in, ctrl);
bufif0 b0 (out, in, ctrl);

//Instantiation of notif gates
notif1 n1 (out, in, ctrl);
notif0 n0 (out, in, ctrl);
```

5.1 Gate Types

5.1.3 Array of Instances

- Situations occur when repetitive instances must be defined
  - These instances differ from each other only by the index of the vector



Example 5-4 Simple Array of Primitive Instances

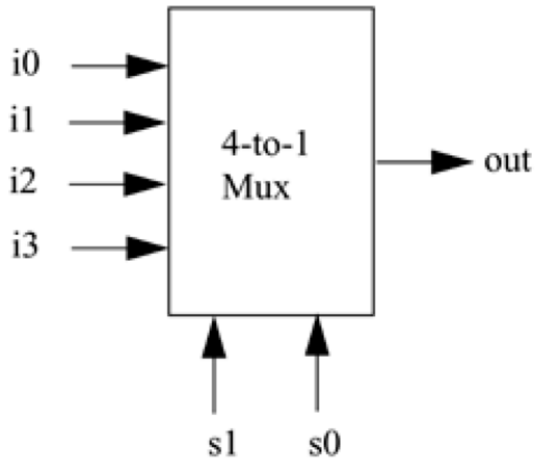
```
wire [7:0] OUT, IN1, IN2;

// basic gate instantiations.
nand n_gate[7:0](OUT, IN1, IN2);

// This is equivalent to the following 8 instantiations
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]);
nand n_gate4(OUT[4], IN1[4], IN2[4]);
nand n_gate5(OUT[5], IN1[5], IN2[5]);
nand n_gate6(OUT[6], IN1[6], IN2[6]);
nand n_gate7(OUT[7], IN1[7], IN2[7]);
```

5.1 Gate Types

5.1.4 Examples

- Steps for designing a 4:1 MUX
  1. I/O diagram
  2. Truth table
  3. Methodology selection
     - (Behavioral/algorithmic level? Dataflow level? Gate level? Switch level?)

Figure 5-4. 4-to-1 Multiplexer



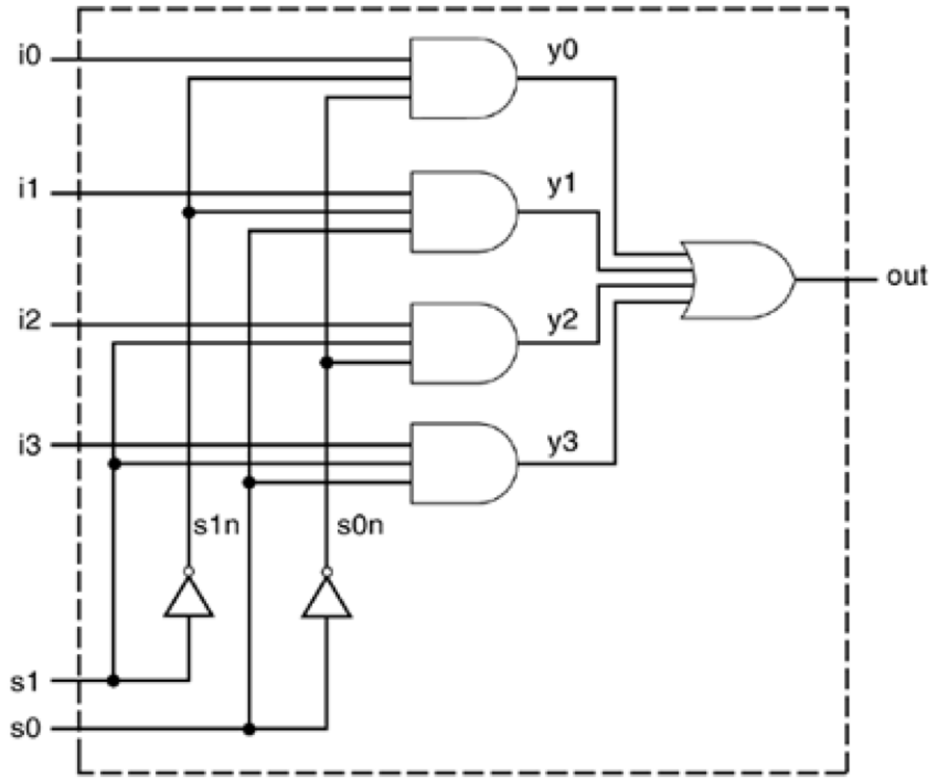
s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

5.1 Gate Types

5.1.4 Examples

- Logic diagram for 4:1 MUX

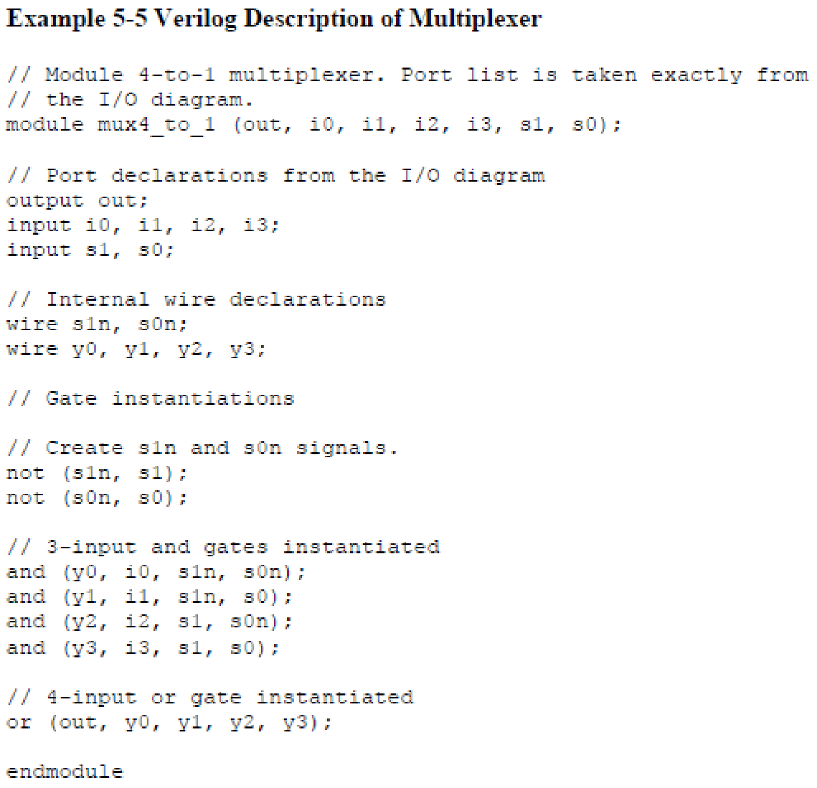
Figure 5-5. Logic Diagram for Multiplexer



5.1 Gate Types

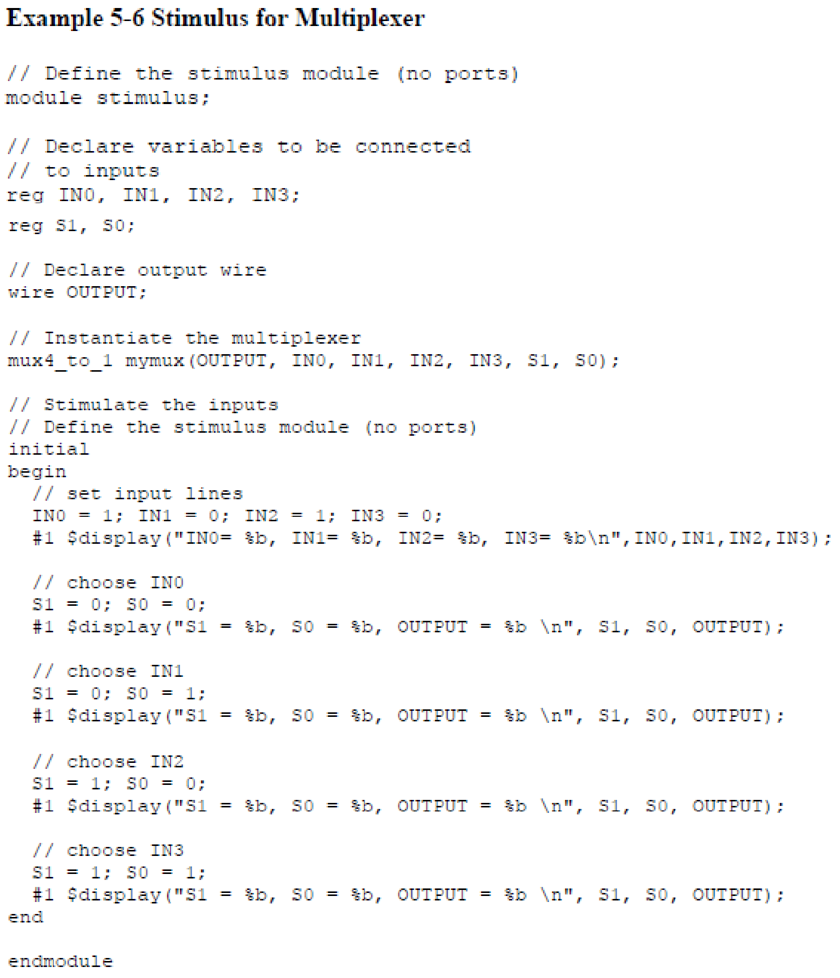
5.1.4 Examples

- Verilog definition for 4:1 MUX



Example 5-5 Verilog Description of Multiplexer

```
// 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);

// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;

// Internal wire declarations
wire s1n, s0n;
wire y0, y1, y2, y3;

// Gate instantiations

// Create s1n and s0n signals.
not (s1n, s1);
not (s0n, s0);

// 3-input and gates instantiated
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);

// 4-input or gate instantiated
or (out, y0, y1, y2, y3);

endmodule
```



Example 5-6 Stimulus for Multiplexer

```
// Define the stimulus module (no ports)
module stimulus;

// Declare variables to be connected
// to inputs
reg IN0, IN1, IN2, IN3;
reg S1, S0;

// Declare output wire
wire OUTPUT;

// Instantiate the multiplexer
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);

// Stimulate the inputs
// Define the stimulus module (no ports)
initial
begin
  // set input lines
  IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
  #1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n",IN0,IN1,IN2,IN3);

  // choose IN0
  S1 = 0; S0 = 0;
  #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);

  // choose IN1
  S1 = 0; S0 = 1;
  #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);

  // choose IN2
  S1 = 1; S0 = 0;
  #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);

  // choose IN3
  S1 = 1; S0 = 1;
  #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
end

endmodule
```

5.1 Gate Types

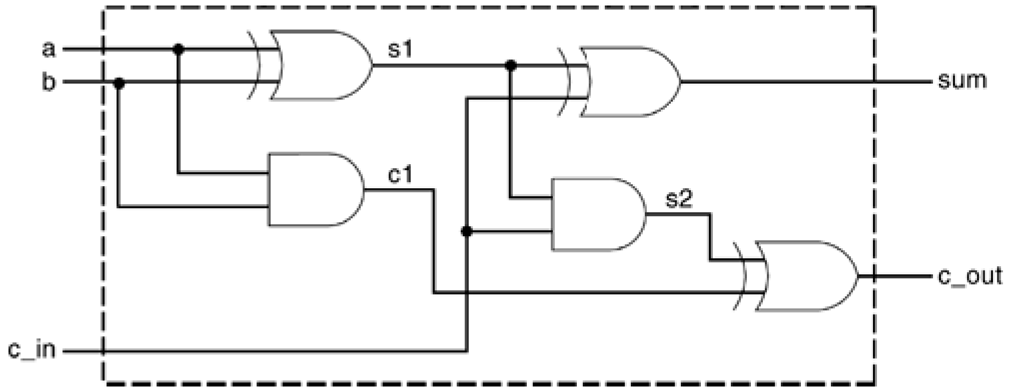
5.1.4 Examples

- 4-bit ripple carry full adder
  - 1-bit full adder
    - sum = (a ⊕ b ⊕ cin)
    - cout = (a * b) + cin * (a ⊕ b)

Example 5-7 Verilog Description for 1-bit Full Adder

```
// Define a 1-bit full adder
module fulladd(sum, c_out, a, b, c_in);

// I/O port declarations
output sum, c_out;
input a, b, c_in;

// Internal nets
wire s1, c1, c2;

// Instantiate logic gate primitives
xor (s1, a, b);
and (c1, a, b);

xor (sum, s1, c_in);
and (c2, s1, c_in);

xor  (c_out, c2, c1);

endmodule
```

Figure 5-6. 1-bit Full Adder



5.1 Gate Types

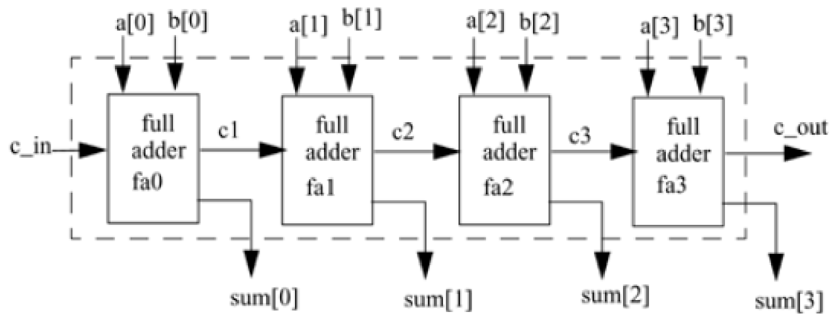
5.1.4 Examples

- 4-bit ripple carry full adder

Example 5-8 Verilog Description for 4-bit Ripple Carry Full Adder

```
// Define a 4-bit full adder
module fulladd4(sum, c_out, a, b, c_in);

// I/O port declarations
output [3:0] sum;
output c_out;
input[3:0] a, b;
input c_in;

// Internal nets
wire c1, c2, c3;

// Instantiate four 1-bit full adders.
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c_out, a[3], b[3], c3);

endmodule
```

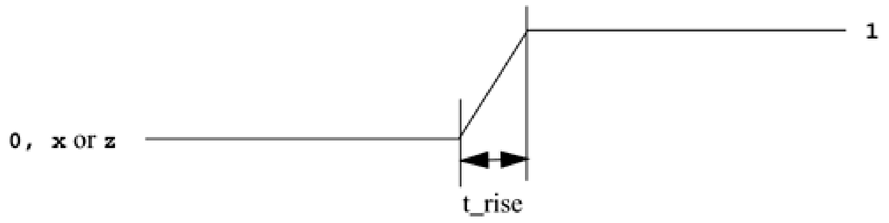
Figure 5-7. 4-bit Ripple Carry Full Adder



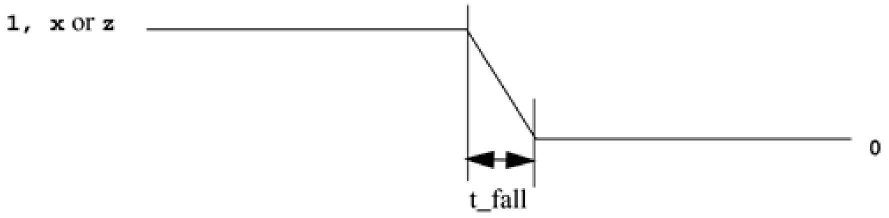
5.2 Gate Delays

5.2.1 Rise, Fall, and Turn-off Delays

- Real circuits have circuit delays
  - Rise/Fall/Turn-off Delays
  - Pin-to-pin delays described in chapter 10
  - Rise



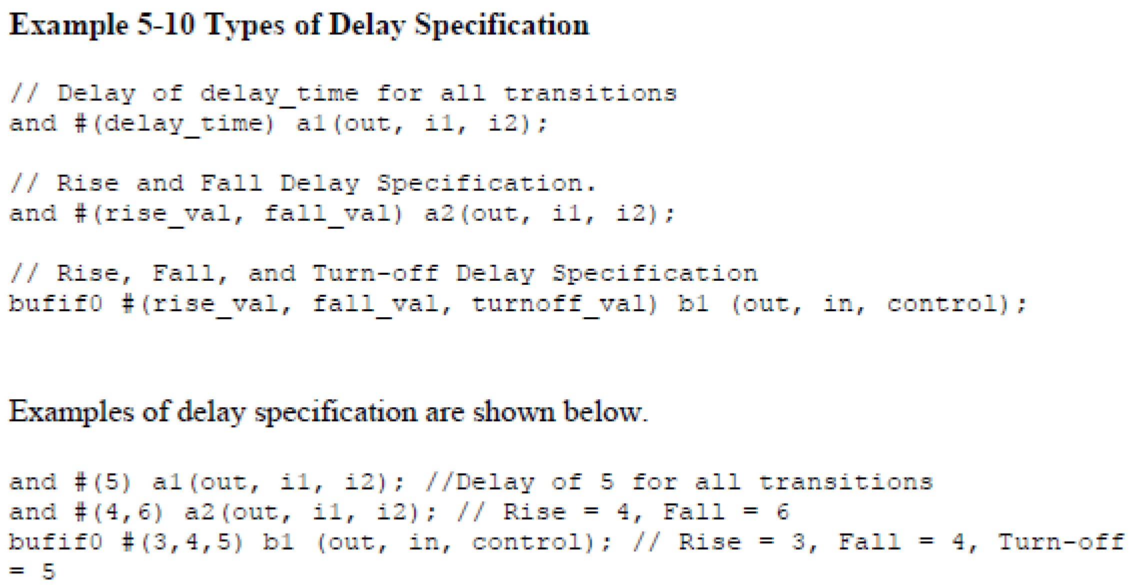
  - Fall



5.2 Gate Delays

5.2.1 Rise, Fall, and Turn-off Delays

- Real circuits have circuit delays
  - Turn-off delay: a gate output transition to the high impedance value (z) from another value



Example 5-10 Types of Delay Specification

```
// Delay of delay_time for all transitions
and #(delay_time) a1(out, i1, i2);

// Rise and Fall Delay Specification.
and #(rise_val, fall_val) a2(out, i1, i2);

// Rise, Fall, and Turn-off Delay Specification
bufif0 #(rise_val, fall_val, turnoff_val) b1 (out, in, control);
```

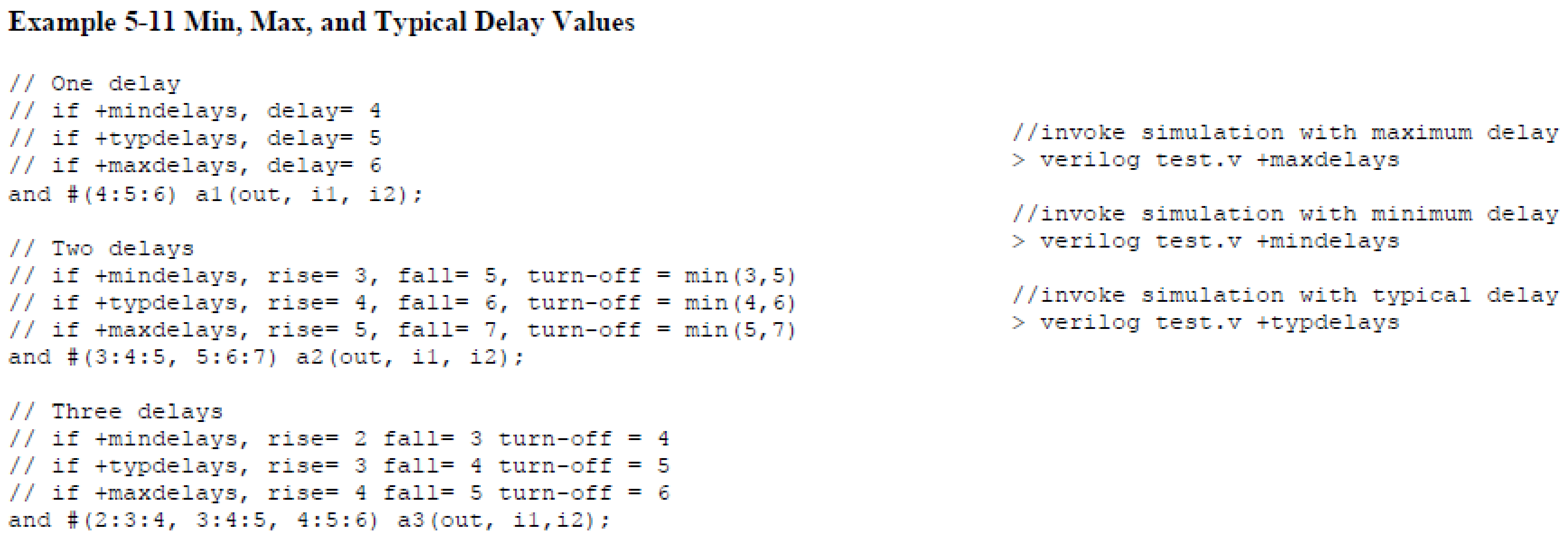
Examples of delay specification are shown below.

```
and #(5) a1(out, i1, i2); //Delay of 5 for all transitions
and #(4,6) a2(out, i1, i2); // Rise = 4, Fall = 6
bufif0 #(3,4,5) b1 (out, in, control); // Rise = 3, Fall = 4, Turn-off
= 5
```

5.2 Gate Delays

5.2.2 Min/Typ/Max Values

- Additional level of control for each type of delay
  - Min, Typ, Max



Example 5-11 Min, Max, and Typical Delay Values

```
// One delay
// if +mindelays, delay= 4
// if +typdelays, delay= 5
// if +maxdelays, delay= 6
and #(4:5:6) a1(out, i1, i2);

// Two delays
// if +mindelays, rise= 3, fall= 5, turn-off = min(3,5)
// if +typdelays, rise= 4, fall= 6, turn-off = min(4,6)
// if +maxdelays, rise= 5, fall= 7, turn-off = min(5,7)
and #(3:4:5, 5:6:7) a2(out, i1, i2);

// Three delays
// if +mindelays, rise= 2 fall= 3 turn-off = 4
// if +typdelays, rise= 3 fall= 4 turn-off = 5
// if +maxdelays, rise= 4 fall= 5 turn-off = 6
and #(2:3:4, 3:4:5, 4:5:6) a3(out, i1,i2);
```

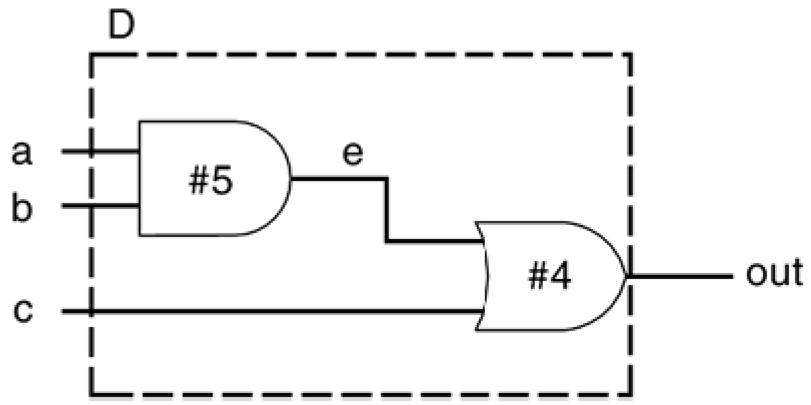
```
//invoke simulation with maximum delay
> verilog test.v +maxdelays

//invoke simulation with minimum delay
> verilog test.v +mindelays

//invoke simulation with typical delay
> verilog test.v +typdelays
```

5.2 Gate Delays

5.2.3 Delay Example

- An example with delay specifications
  - out = (a * b) + c

Figure 5-8. Module D



Example 5-12 Verilog Definition for Module D with Delay

```
// Define a simple combination module called D
module D (out, a, b, c);

// I/O port declarations
output out;
input a,b,c;

// Internal nets
wire e;

// Instantiate primitive gates to build the circuit
and #(5) a1(e, a, b); //Delay of 5 on gate a1
or  #(4) o1(out, e,c); //Delay of 4 on gate o1

endmodule
```

Example 5-13 Stimulus for Module D with Delay

```
// Stimulus (top-level module)
module stimulus;

// Declare variables
reg A, B, C;
wire OUT;

// Instantiate the module D
D d1( OUT, A, B, C);

// Stimulate the inputs. Finish the simulation at 40 time units.
initial
begin
  A= 1'b0; B= 1'b0; C= 1'b0;

  #10 A= 1'b1; B= 1'b1; C= 1'b1;

  #10 A= 1'b1; B= 1'b0; C= 1'b0;

  #20 $finish;
end

endmodule
```