NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject:

VERILOG

Hardware Description Language

Chapter4: Primitive Gates – Switches – User defined primitives

Lecturer: Lam Duc Khai

Agenda

1. Chapter 1: Introduction (Week1)
2. Chapter 2: Fundamental concepts   (Week1)
3. Chapter 3: Modules and hierarchical structure (Week2)
4. Chapter 4: Primitive Gates – Switches – User defined primitives (Week2)
5. Chapter 5: Structural model (Week3)
6. Chapter 6: Behavioral model – Combination circuit (Week4)
7. Chapter 7: Behavioral model – Sequential circuit (Week5)
8. Chapter 8: Tasks and Functions (Week6)
9. Chapter 9: State machines (Week6)
10. Chaper 10: Testbench and verification (Week7)

COMPUTER ENGINEERING

Agenda

4. Chapter 4: Primitive Gates – Switches – User defined primitives

- Primitive Gates
- Switches
- Logic strength modelling.
- Gate delays
- User-defined primitives

COMPUTER ENGINEERING

Primitive Gates

- There are 12 primitive logic gates predefined in the Verilog HDL to provide the gate - structural level modeling facility.
  - ✓ and, nand, nor, or, xor, xnor
  - ✓ buf, not
  - ✓ bufif1, bufif0, notif1, notif0
- Advantages:
  - ✓ Gates provide a much closer one-to-one mapping between the actual circuit and the model.
  - ✓ There is no continuous assignment equivalent to the bidirectional transfer gate.

COMPUTER ENGINEERING

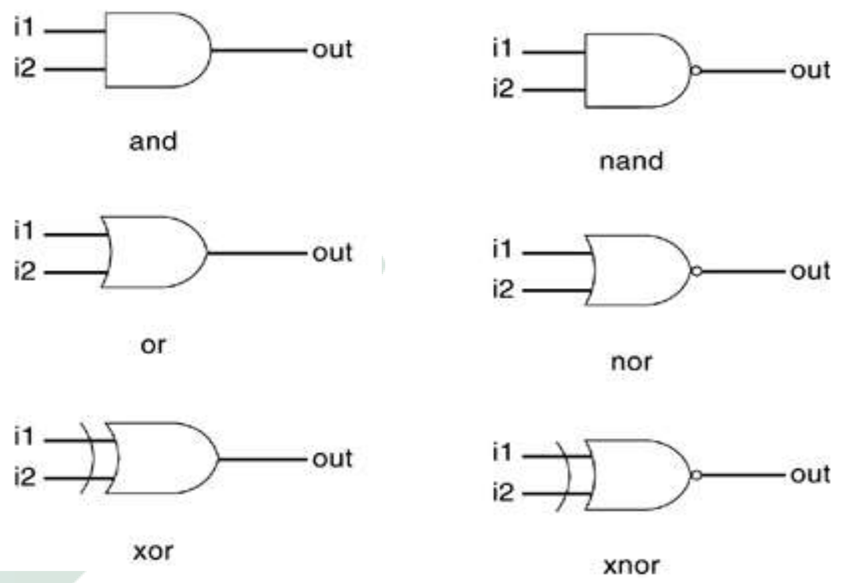
Primitive Gates (Cont'd)

➢ And/Or/Nand/Nor/Xor/Xnor Gates

One scalar output

Multiple scalar inputs

The first terminal in the list of gate terminals is an output and the other terminals are inputs



Terminal list

```
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);
```

Verilog automatically instantiates the appropriate gate.

```
// 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
```

COMPUTER ENGINEERING

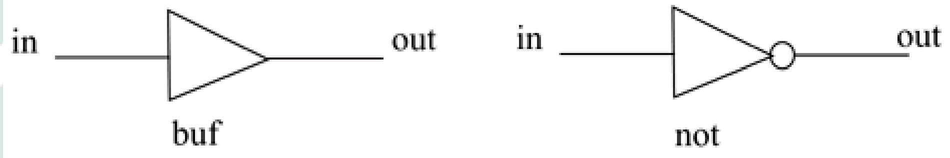
Primitive Gates (Cont'd)

➤ Buf/Not Gates

One scalar input

One or more scalar outputs

The last terminal in the port list is connected to the input



buf	in	out
	0	0
	1	1
	x	x
	z	x

not	in	out
	0	1
	1	0
	x	x
	z	x

```
// 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
```

COMPUTER ENGINEERING

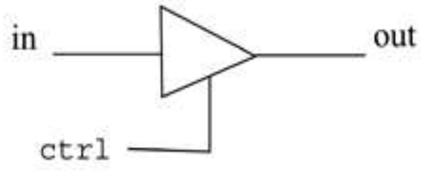
Primitive Gates (Cont'd)

➢ Bufif/notif

Gates with an additional control signal on **buf** and **not** gates

Propagate only if control signal is asserted.

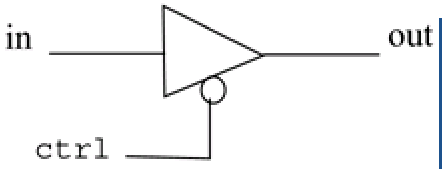
Propagate z if their control signal is deasserted



bufif1

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

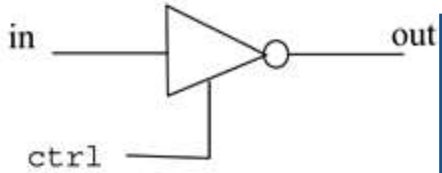
bufif1 b1 (out, in, ctrl);



bufif0

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

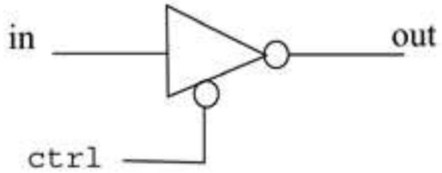
bufif0 b0 (out, in, ctrl);



notif1

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

notif1 n1 (out, in, ctrl);



notif0

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

notif0 n0 (out, in, ctrl);

Primitive Gates (Cont’d)

➢ Array of Instances

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

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

The instances differ from each other only by the index of the vector to which they are connected

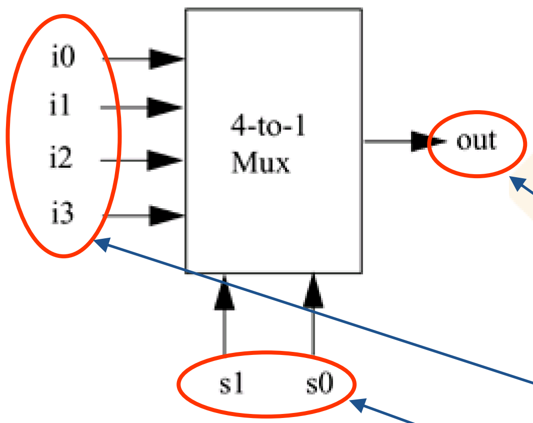
||

```
// 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]);
```

COMPUTER ENGINEERING

Primitive Gates (Cont'd)

➢ Example: Gate-level multiplexer



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

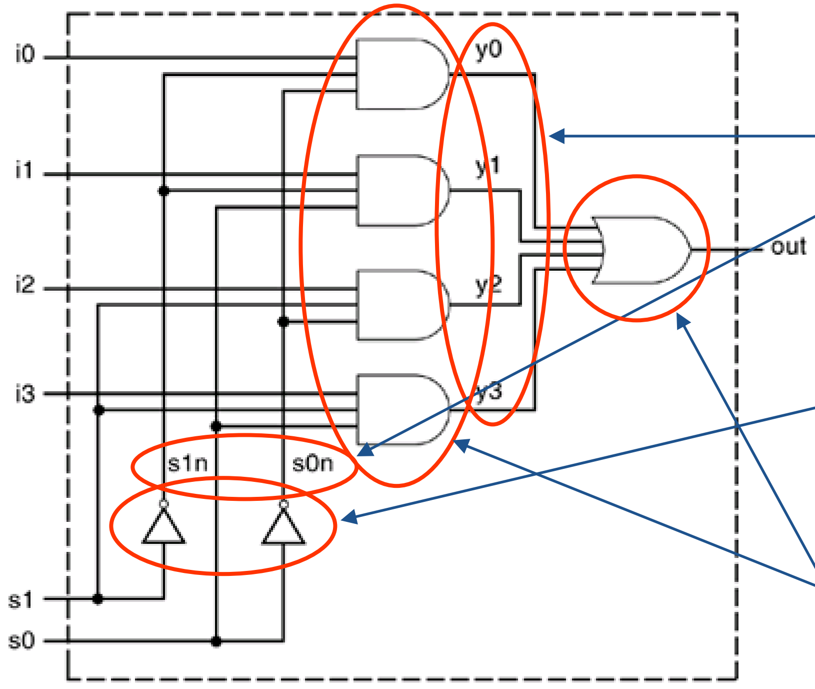
4-to-1 Multiplexer

```
// 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;
```

OMPUTER ENGINEERING

Primitive Gates (Cont'd)

➢ Example: Gate-level multiplexer (Cont'd)



Logic Diagram for 4-to-1 Multiplexer

```
// 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
```

Primitive Gates (Cont’d)

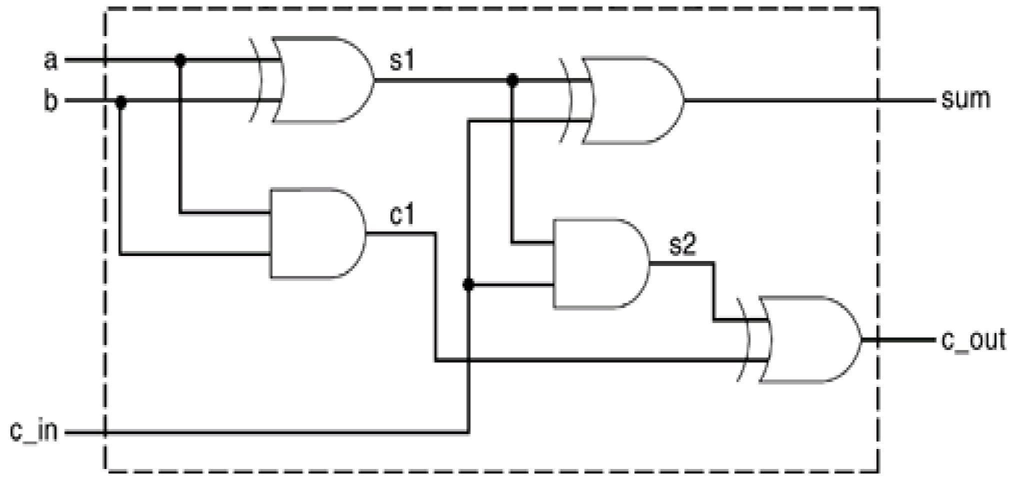
➢ Example: 4-bit Ripple Carry Full Adder

1-bit ripple carry 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
```

$$sum = (a \oplus b \oplus cin)$$

$$cout = (a \cdot b) + cin \cdot (a \oplus b)$$

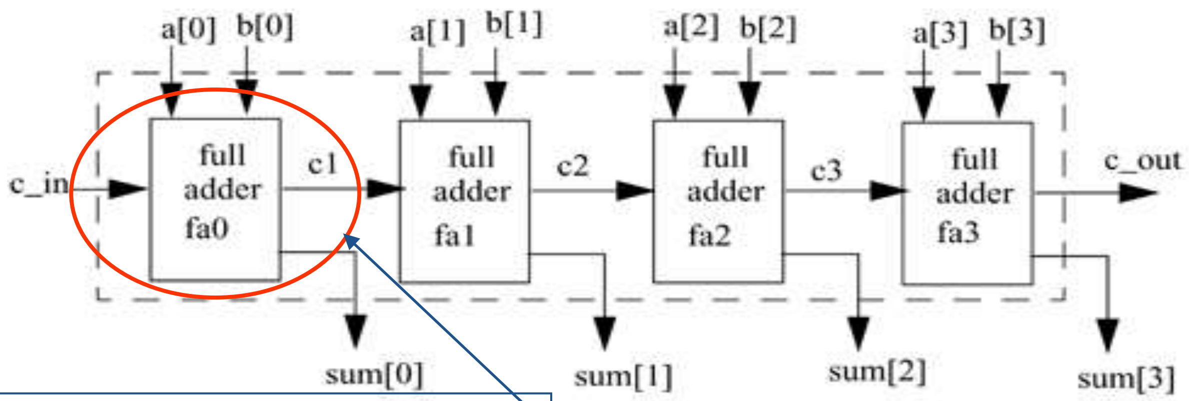


COMPUTER

Primitive Gates (Cont'd)

➢ Example: 4-bit Ripple Carry Full Adder

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
```

COMPUTER ENGINEERING

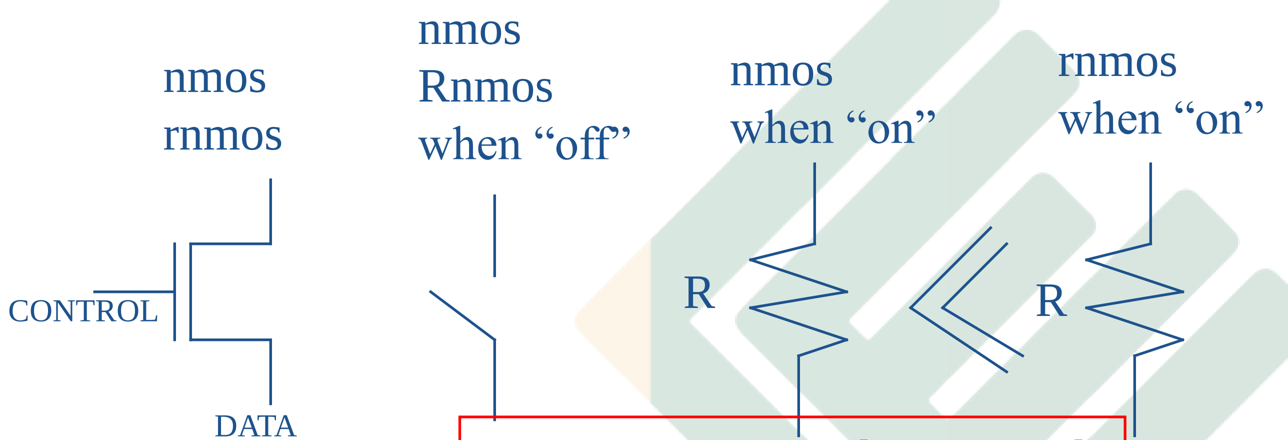
Switches

- ➢There are two kinds of switch:
  - ✓Mos switches : cmos, nmos, pmos, rcmos, rnmos, rpmos
  - ✓Bidirectional pass switches: tran, rtran, tranif1, rtranif1, tranif0, rtranif0.
- ➢Advantages:
  - ✓Gates provide a much closer one-to-one mapping between the actual circuit and the model.
- ➢There is no continuous assignment equivalent to the bidirectional transfer gate.

COMPUTER ENGINEERING

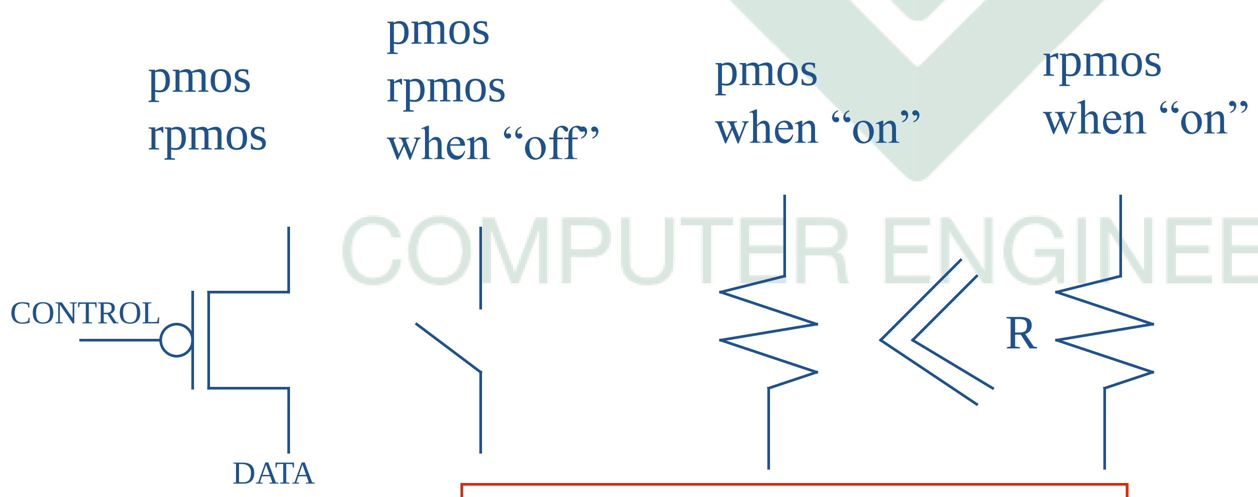
Switches (Cont'd)

➢MOS Switches



Ex: nmos n1 (out, data, control)

nmos rnmos		CONTROL			
		0	1	x	z
D	0	z	0	L	L
A	1	z	1	H	H
T	x	z	x	x	x
A	z	z	z	z	z



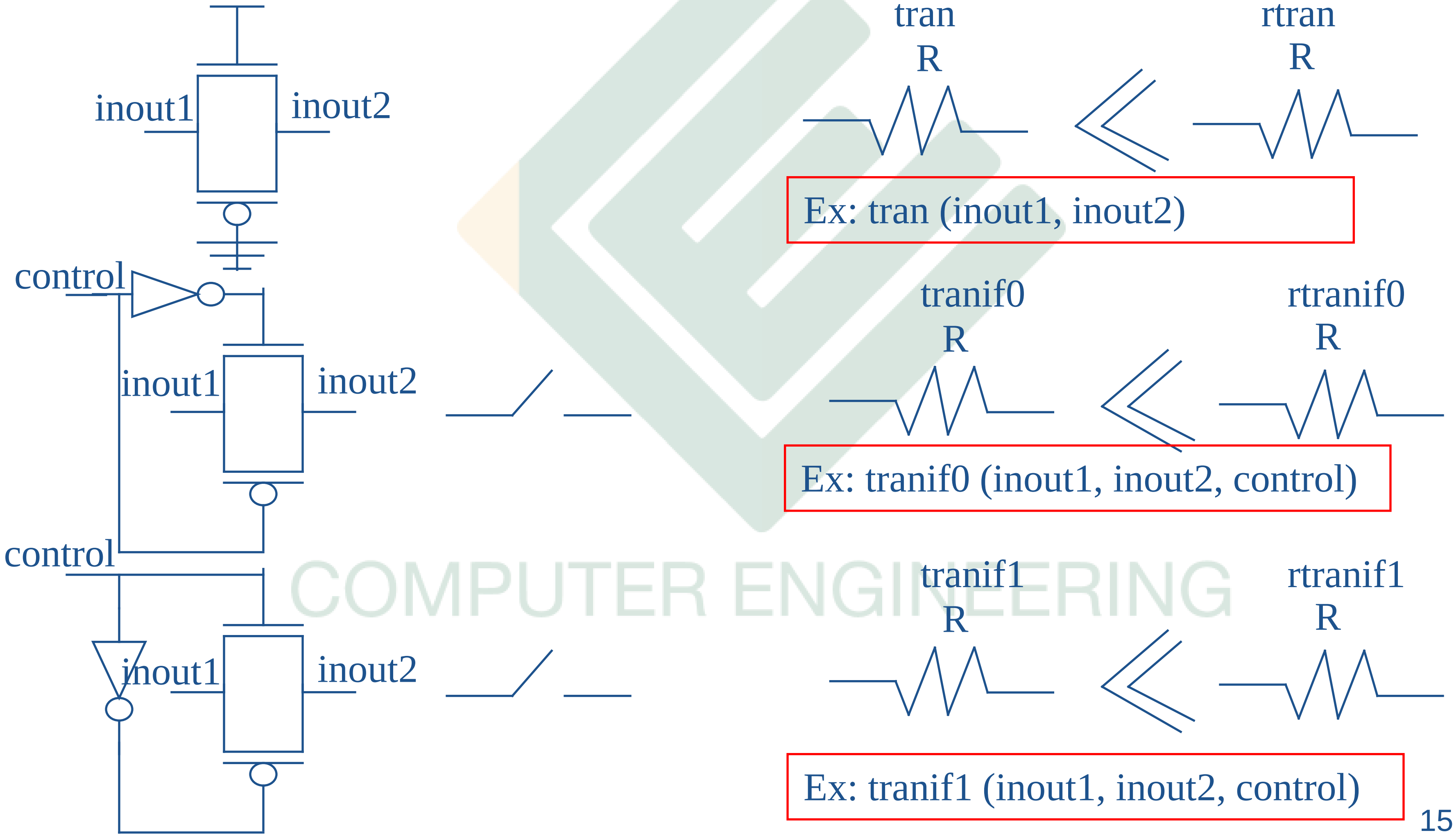
Ex: pmos p1 (out, data, control)

pmos rpmos		CONTROL			
		0	1	x	z
D	0	0	z	L	L
A	1	1	z	H	H
T	x	x	z	x	x
A	z	z	z	z	z

COMPUTER ENGINEE

Switches (Cont'd)

➢Bidirection Switches



Ex: tran (inout1, inout2)

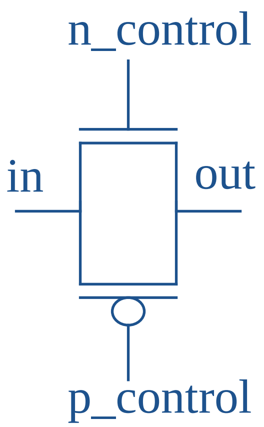
Ex: tranif0 (inout1, inout2, control)

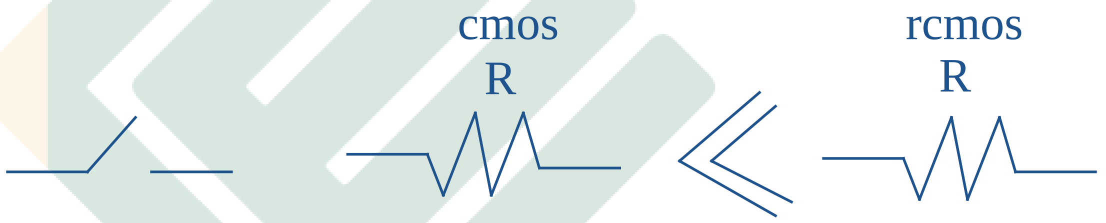
Ex: tranif1 (inout1, inout2, control)

COMPUTER ENGINEERING

Switches (Cont’d)

➢CMOS Switches





cmos (out, in, n_control, p_control)

COMPUTER ENGINEERING

Strength Modeling

✓Allows specification of drive strength for primitive **gate outputs** and **nets** or **assign** keyword**.**

✓**Gate output** or **net** signal strength values are specified in a set of parenthesis that include a strength value for logic **0** and one for logic **1**.

✓Drive strengths for logic **0** (i.e., *strength0*) are **supply0**, **strong0**, **pull0**, **weak0**, and **highz0**.

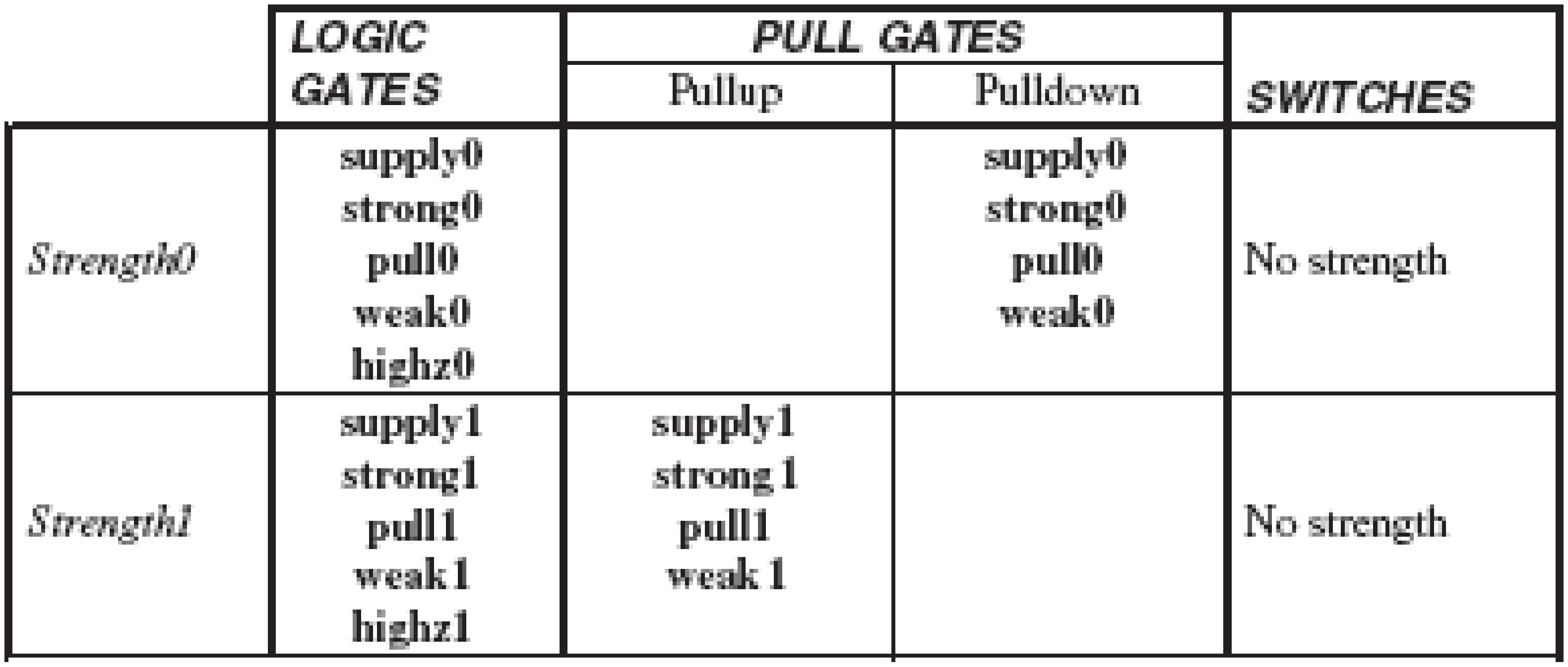
✓Drive strengths for logic **1** (*strength1*) are **supply1**, **strong1**, **pull1**, **weak1**, and **highz1**.

✓Charge strengths, representing the strength of a capacitive **net**, charge strength values are **large**, **medium**, and **small**.

COMPUTER ENGINEERING

Strength Modeling (Cont'd)

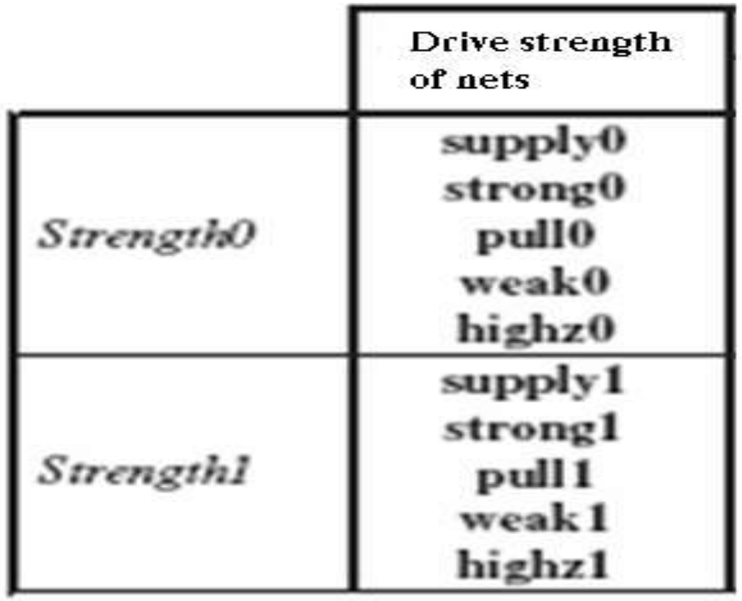
➢Drive strength value of primitive gate outputs



	LOGIC GATES	*PULL GATES*		*SWITCHES*
		Pullup	Pulldown	
Strength0	supply0 strong0 pull0 weak0 highz0		supply0 strong0 pull0 weak0	No strength
Strength1	supply1 strong1 pull1 weak1 highz1	supply1 strong 1 pull1 weak 1		No strength

Strength Modeling (Cont'd)

- Strength value of nets
  - Drive strength



	Drive strength of nets
Strength0	supply0 strong0 pull0 weak0 highz0
Strength1	supply1 strong1 pull1 weak1 highz1

COMPUTER ENGINEERING

  - Charge strength

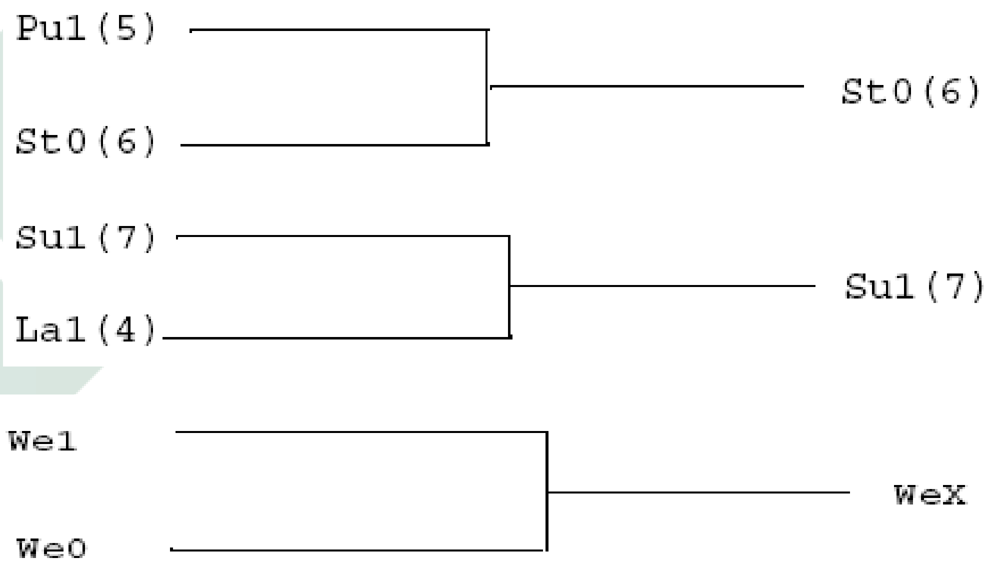
    Large, medium, small

Strength Modeling (Cont'd)

➢The strength level

Strength name	Strength level
supply0	7
strong0	6
pull0	5
large0	4
weak0	3
medium0	2
small0	1
highz0	0
highz1	0
small1	1
medium1	2
weak1	3
large1	4
pull1	5
strong1	6
supply1	7

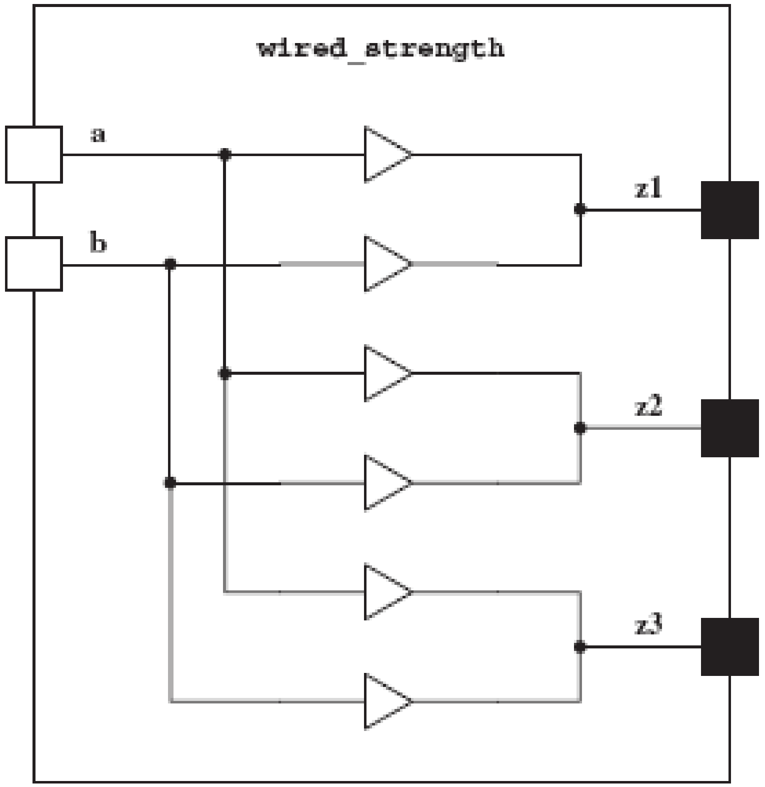
The stronger signal shall dominate all the weaker drivers and determine the result.



Strength Modeling (Cont'd)

➢Example

Buffer producing multiple drivers:



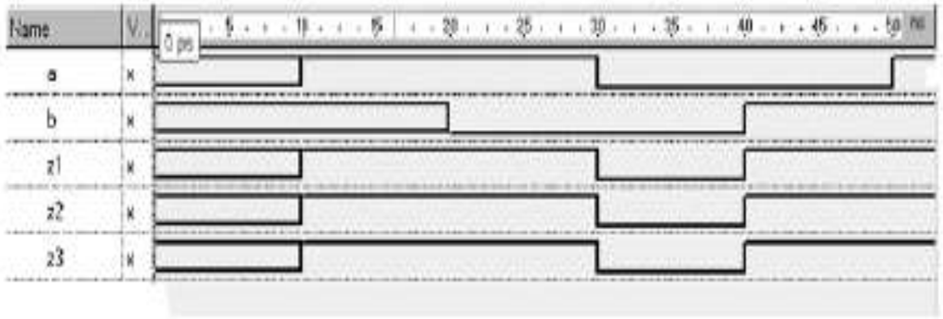
COM NG

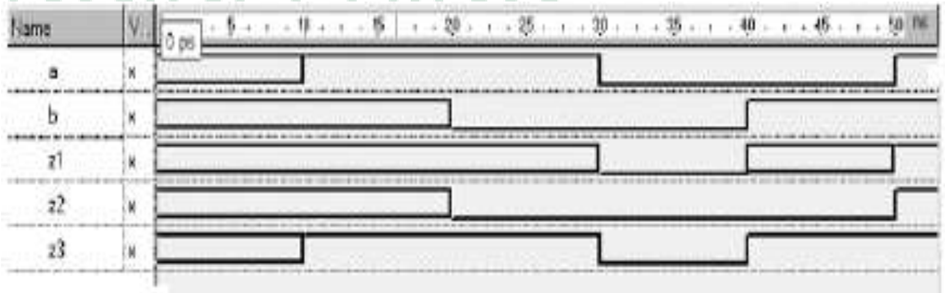
Strength Modeling (Cont'd)

➢Example (Cont'd):

```
`timescale 1ns/100ps

module wired_strength (input a, b, output z1, z2, z3);

   wire z1;
   wand z2;
   wor z3;

   // Wired logic
   buf (pull1, weak0) (z1, a);
   buf (pull1, weak0) (z1, b);h
   // Wired-and logic
   buf (pull1, weak0) (z2, a);
   buf (pull1, weak0) (z2, b);
   // Wired-or logic
   buf (pull1, weak0) a3 (z3, a);
   buf (pull1, weak0) b3 (z3, b);

endmodule
```

```
`timescale 1ns/100ps

module wired_strength (input a, b, output z1, z2, z3);

   wire z1;
   wand z2;
   wor z3;

   // Wired logic
   buf (pull1, pull0) (z1, a);
   buf (pull1, pull0) (z1, b);
   // Wired-and logic
   buf (pull1, pull0) (z2, a);
   buf (pull1, pull0) (z2, b);
   // Wired-or logic
   buf (pull1, pull0) (z3, a);
   buf (pull1, pull0) (z3, b);

endmodule
```

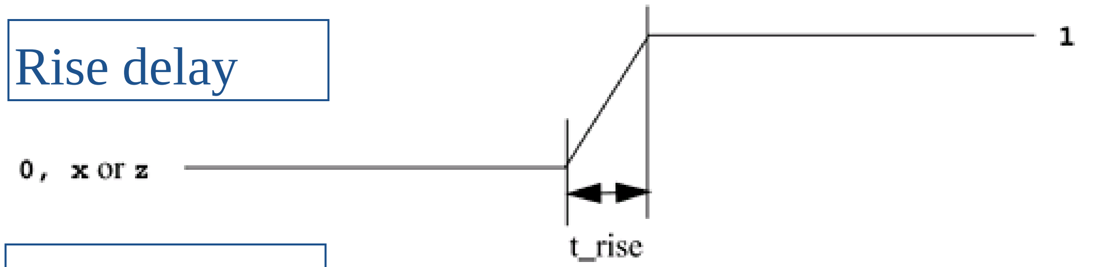




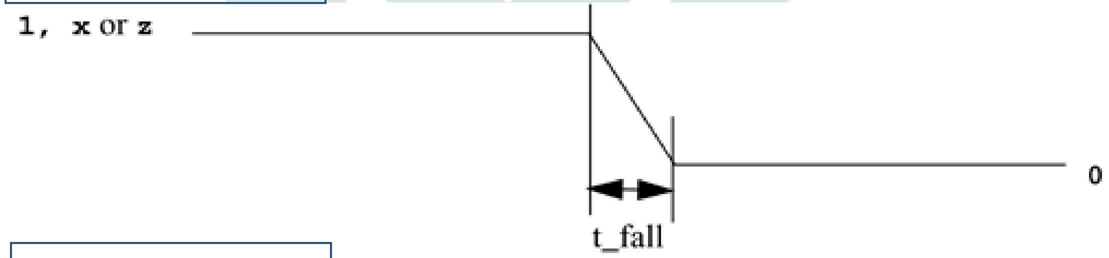
Gate Delays

➢ Rise, Fall, and Turn-off Delays

Rise delay



Fall delay



Turn-off delay

$1, 0, x \longrightarrow z$

Note: If a value changes to **x**, the minimum of the three delays is considered

COMPUTER ENGINEERING

Gate Delays (Cont’d)

➢ Rise, Fall, and Turn-off Delays (Cont’d)

If only *one* delay is specified, this value is used for all transitions.

If *two* delays are specified, they refer to the rise and fall delay values. The turn-off delay is the minimum of the two.

If all *three* delays are specified, they refer to rise, fall, and turn-off delay values.

If *no* delays are specified, the default value is zero.

Ex:

```
and #(5) a1(out, i1, i2); // Rise = 5, Fall = 5, Turn-off = 5

and #(4,6) a2(out, i1, i2); // Rise = 4, Fall = 6, Turn-off = 4

bufif0 #(3,4,5) b1 (out, in, control); // Rise = 3, Fall = 4, Turn-off = 5
```

COMPUTER ENGINEERING

Gate Delays (Cont'd)

➢ Min/Typ/Max Values

For each type of delay three values, *min*, *typ*, and *max*, can be specified.

Min/typ/max values are used to model devices whose delays vary within a [min max] range because of the IC fabrication process variations.

min	The minimum delay value that the designer expects the gate to have
typ	The typical delay value that the designer expects the gate to have
max	The maximum delay value that the designer expects the gate to have

COMPUTER ENGINEERING

Min, *typ*, or *max* values can be chosen at Verilog run time.

Method of choosing a *min/typ/max* value may vary for different simulators or operating systems

Gate Delays (Cont'd)

➢ Min/Typ/Max Values (Cont'd)

```
// 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);
```

With Verilog XL™

COMPUTER ENGINEERING

User-Defined Primitives

- The set of predefined gate primitives by designing and specifying new primitive elements
- Instances of these new UDPs can be used in exactly the same manner as the gate primitives
- Way to define combinational and sequential elements using a truth table
- Each UDP has exactly one output, which can be in one of three states: 0, 1, or x.
- Often simulate faster than using expressions, collections of primitive gates, etc.
- Gives more control over behavior with X inputs
- Most often used for specifying custom gate libraries

COMPUTER ENGINEERING

User-Defined Primitives (Cont'd)

➢Example: Combinational UDPs

```
primitive multiplexer (mux, control, dataA, dataB);
output mux;
input control, dataA, dataB;
table
//          control dataA dataB mux
            0 1 0 : 1 ;
            0 1 1 : 1 ;
            0 1 x : 1 ;
            0 0 0 : 0 ;
            0 0 1 : 0 ;
            0 0 x : 0 ;
            1 0 1 : 1 ;
            1 1 1 : 1 ;
            1 x 1 : 1 ;
            1 0 0 : 0 ;
            1 1 0 : 0 ;
            1 x 0 : 0 ;
            x 0 0 : 0 ;
            x 1 1 : 1 ;
endtable
endprimitive
```

```
output mux;
input control, dataA, dataB;
table
//          control dataA dataB mux
            0 1 ? : 1 ; // ? = 0 1 x
            0 0 ? : 0 ;
            1 ? 1 : 1 ;
            1 ? 0 : 0 ;
            x 0 0 : 0 ;
            x 1 1 : 1 ;
endtable
endprimitive
```

COMPUTER ENGINEERING

User-Defined Primitives (Cont’d)

➢**Example: Level-sensitive sequential UDPs**

```
primitive latch (q, clock, data);
output q; reg q;
input clock, data;
table
// clock data q q+
0 1 : ? : 1 ;
0 0 : ? : 0 ;
1 ? : ? : - ; // - = no change
endtable
endprimitive
```

COMPUTER ENGINEERING

29

User-Defined Primitives (Cont'd)

- **Example: Edge-sensitive sequential UDPs**

```
primitive d_edge_ff (q, clock, data);
output q; reg q;
input clock, data;
table
// obtain output on rising edge of clock
//    clock data q q+
      (01) 0 : ? : 0 ;
      (01) 1 : ? : 1 ;
      (0?) 1 : 1 : 1 ;
      (0?) 0 : 0 : 0 ;
// ignore negative edge of clock
      (?0) ? : ? : - ;
// ignore data changes on steady clock
       ?     ? : ? : - ; // - = no change
endtable
endprimitive
```

COMPUTER ENGINEERING

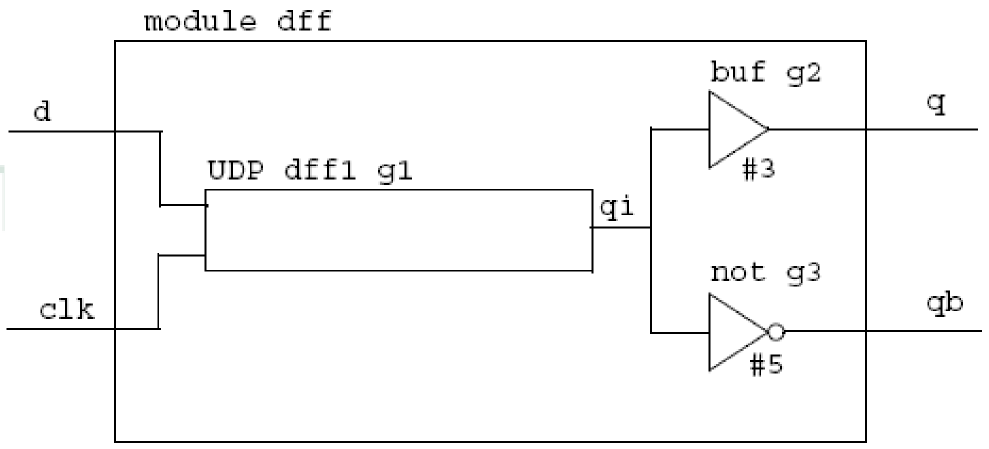
User-Defined Primitives (Cont'd)

➢Initial statement in UDPs and instances

```
primitive dff1 (q, clk, d);
input clk, d;
output q; reg q;
initial q = 1'b1;
table
// clk d q q+
r 0 : ? : 0 ;
r 1 : ? : 1 ;
f ? : ? : - ;
? * : ? : - ;
endtable
endprimitive
```

```
module dff (q, qb, clk, d);
input clk, d;
output q, qb;
dff1 g1 (qi, clk, d);
buf #3 g2 (q, qi);
not #5 g3 (qb, qi);
endmodule
```

COMPUT



END

COMPUTER ENGINEERING