Digital Design with Verilog

Verilog

Lecture 7: Gate-Level Modeling

Learning Objectives

- Identify logic gate primitives provided in Verilog.
- Understand instantiation of gates, gate symbols, and truth tables for and/or and buf/not type gates.
- Understand how to construct a Verilog description from the logic diagram of the circuit.
- Describe rise, fall, and turn-off delays in the gate-level design.
- Explain min, max, and typ delays in the gate-level design.

Gate Types

- Verilog supports basic logic gates as predefined primitives.
    - These primitives are instantiated like modules except that they are predefined in Verilog and do not need a module definition.

Gate Types

- There are two classes of basic gates: *and/or* gates and *buf/not* gates.

- And/Or Gates

  - and or xor
  - nand nor xnor

- Buf/Not Gates

  - buf not

Example: 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
```

Truth Table “and/or” Gate

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

Truth Table “nand/nor” Gate

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

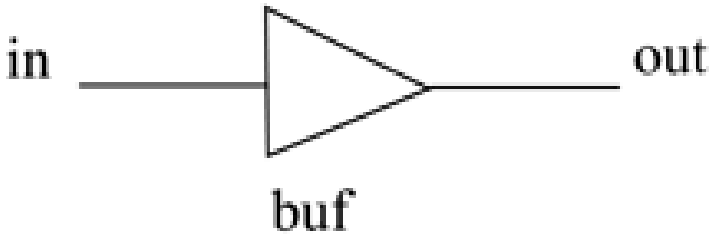
Truth Table “xor/xnor” Gate

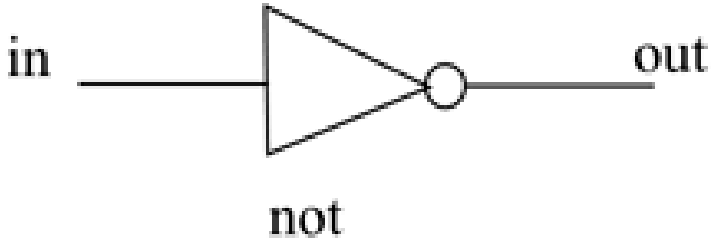
xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

xnor	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

Buf/Not Gates

- Buf/not gates have one scalar input and one or more scalar outputs.



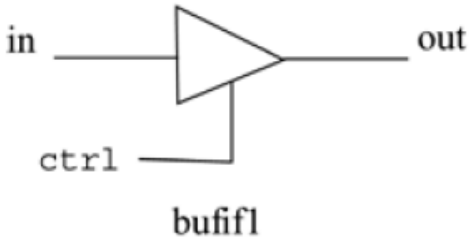



```
//Gate Instantiations of Buf/Not Gates
// basic gate instantiations.
buf b1(OUT1, IN);
not n1(OUT1, IN);
// More than two outputs
buf b1_2out(OUT1, OUT2, IN);
// gate instantiation without instance name
not (OUT1, IN); // legal gate instantiation
```

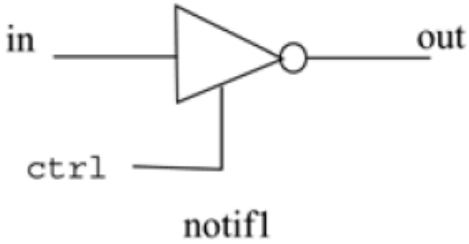
Truth Tables for Buf/Not Gates

buf	in	out	not	in	out
	0	0		0	1
	1	1		1	0
	x	x		x	x
	z	x		z	x

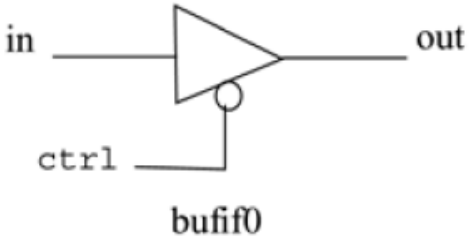
Bufif/notif Gates



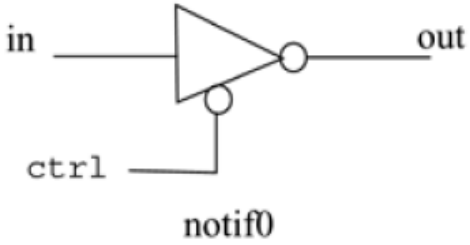
bufif1



notif1



bufif0



notif0

```
//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);
```

Truth Tables: bufif1/notif1

ctrl

in

Bufif1 cntrl ->	0	1	x	z
0	z	0	L	L
1	z	1	H	H
x	z	x	x	x
z	z	x	x	x

ctrl

in

notif1 cntrl ->	0	1	x	z
0	z	1	H	H
1	z	0	L	L
x	z	x	x	x
z	z	x	x	x

Truth Tables: bufif0/notif0

ctrl

in: Bufif0 cntrl ->	0	1	x	z
0	0	z	L	L
1	1	z	H	H
x	x	z	x	x
z	x	z	x	x

ctrl

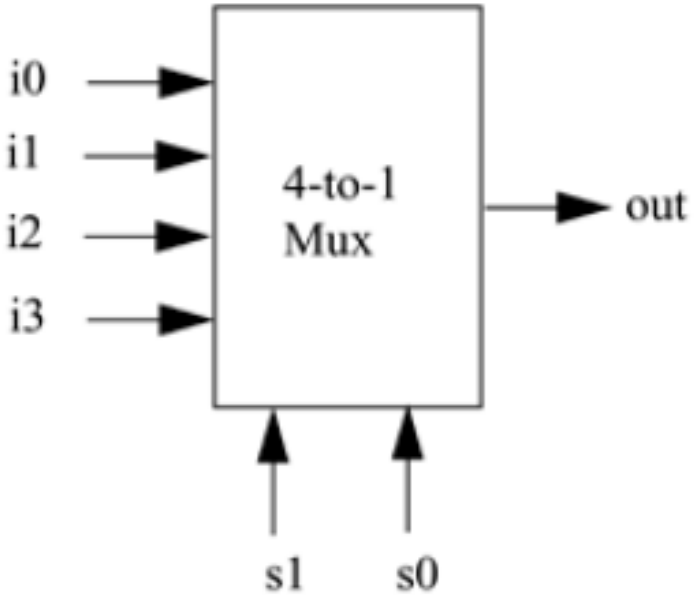
in: notif0 cntrl ->	0	1	x	z
0	1	z	H	H
1	0	z	L	L
x	x	z	x	x
z	x	z	x	x

Array of 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]);
```

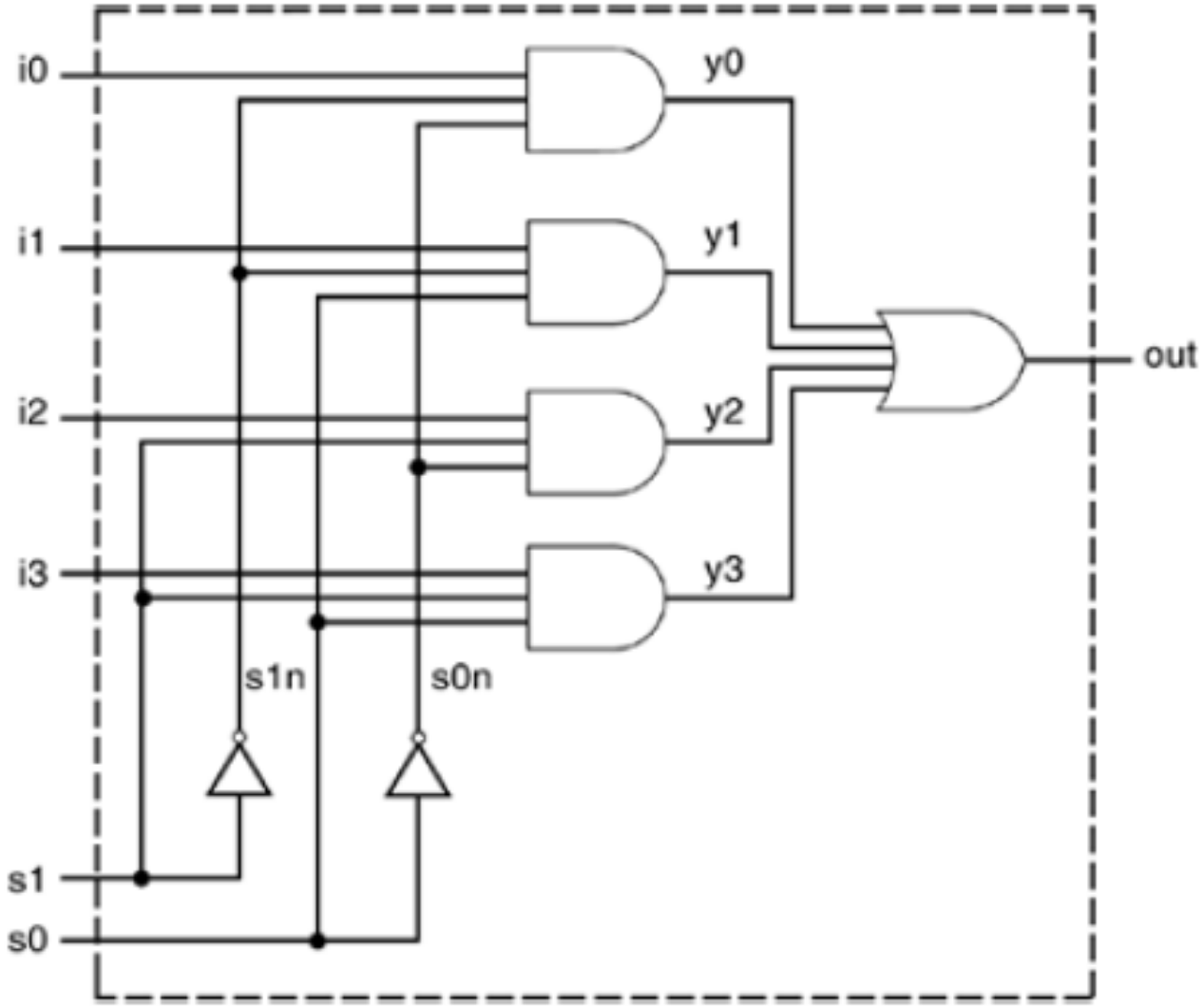
Example 1:

- 4x1 multiplexer



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

Example 1: (Contd.)



Example 1: (Contd.)

```
// 4-to-1 multiplexer.
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 1: (Contd.)

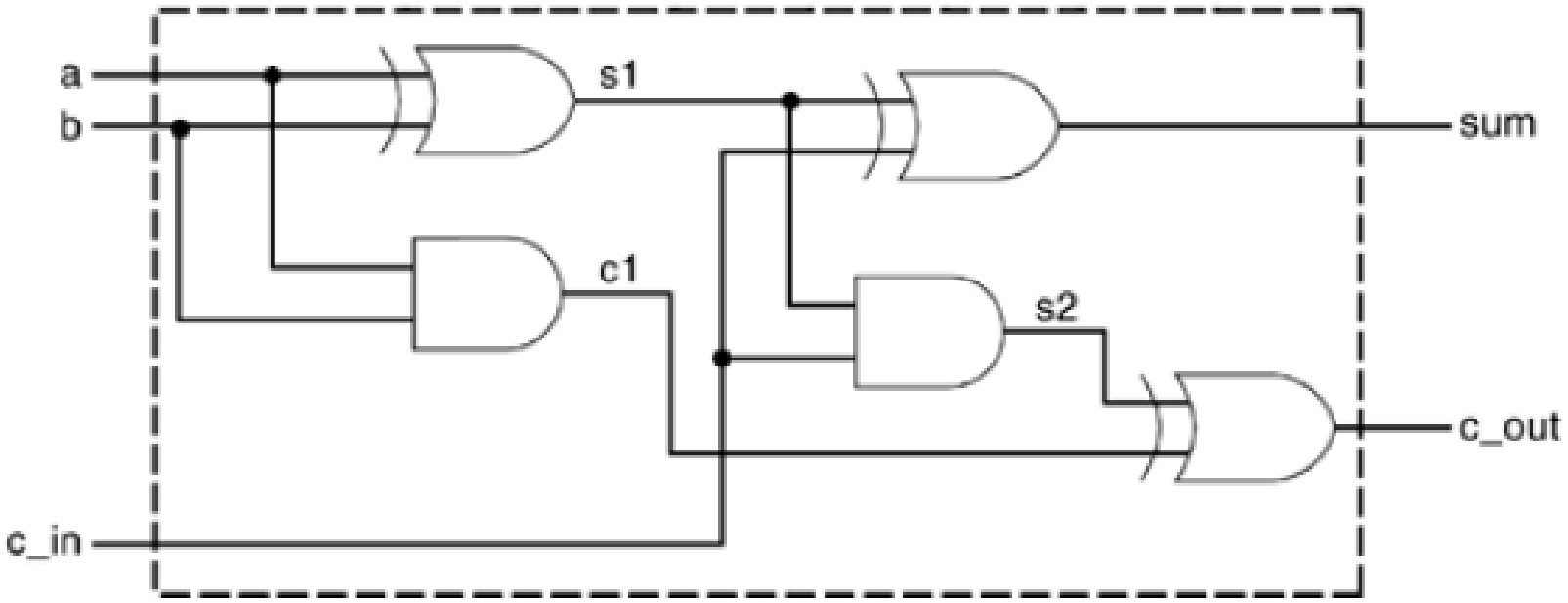
```
// Define the stimulus module (no ports)
module stimulus;

// 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
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
```

4-bit Ripple Carry Full Adder

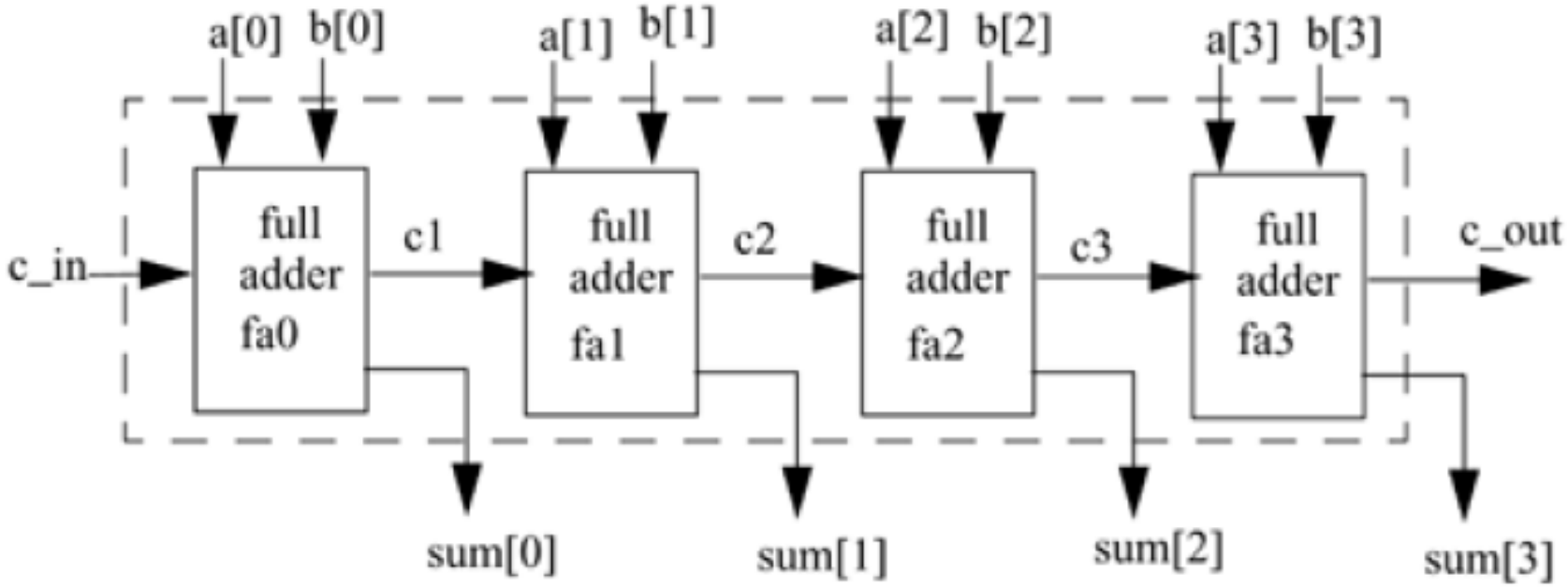


1-bit Full Adder

4-bit Ripple Carry Full Adder (Contd.)

```
// 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);
or   (c_out, c2, c1);
endmodule
```

4-bit Ripple Carry Full Adder (Contd.)



4-bit Ripple Carry Full Adder (Contd.)

```
// 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
```

4-bit Ripple Carry Full Adder (Contd.)

```
// Define the stimulus (top level module)
module stimulus;

// Set up variables
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;

// Instantiate the 4-bit full adder. call it FA1_4
fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);

// Setup the monitoring for the signal values
initial
begin
    $monitor($time," A= %b, B=%b, C_IN= %b,, C_OUT= %b, SUM= %b\n",
                                        A, B, C_IN, C_OUT, SUM);
end
// Stimulate inputs
initial
begin
    A = 4'd0; B = 4'd0; C_IN = 1'b0;

    #50 A = 4'd3; B = 4'd4;

    #50 A = 4'd2; B = 4'd5;

    #50 A = 4'd9; B = 4'd9;

    #50 A = 4'd10; B = 4'd15;

    #50 A = 4'd10; B = 4'd5; C_IN = 1'b1;
end
```

References

- Chapter 5, Verilog HDL by Samir Palnitkar, Second Edition.

Thank you