

Verilog Example

```
// Description of simple circuit Fig. 3-37
module smpl_circuit (A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and g1 (e,A,B);
  not g2 (y,C);
  or g3 (x,e,y);
endmodule
```

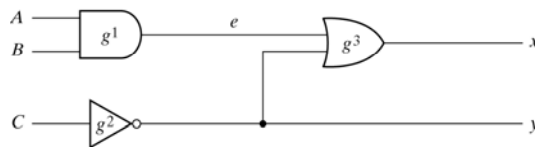


Fig. 3-37 Circuit to Demonstrate HDL

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Some Verilog Syntax

- Approximately 100 keywords (lowercase)
 - Verilog IS case-sensitive
 - Predefined identifiers Used for Basic Language Constructs
- Comments are:
 - // to end of line
 - /* **comment here** */
- Simulator Directives
 - Technically not part of language, but Standard
 - Begin with a \$
 - Example **\$finish;**
- Simulator Directives not used for:
 - Documentation
 - Synthesis

Verilog Example

```
// Description of simple circuit Fig. 3-37
module smpl_circuit (A,B,C,x,y);
    input A,B,C;
    output x,y;
    wire e;
    and g1(e,A,B);
    not g2(y,C);
    or g3(x,e,y);
endmodule
```

- Basic Unit is the **module**
- Must Declare **inputs** and **outputs**
 - indicate when events are generated
- Must declare “data type”
 - for now **wire**
- One module can Cause Events in another

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Delay Modeling

- Timescale Directive
 - ``timescale 1ns/100ps`
 - First number is unit of measurement for delays
 - Second number is precision (round-off increments)
- We will use Default Increments
 - # Directive for Delay Modeling
 - Timing Allows for More Accurate Modeling
 - Concept of Back Annotation

Verilog Example with Delay

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and  #(30) g1(e,A,B);
    not  #(10) g2(y,C);
    or   #(20) g3(x,e,y);
endmodule
```

Where does this
come from?



	Time Units (ns)	Input ABC	Output y e x
Initial	-	000	1 0 1
Change	-	111	1 0 1
	10	111	0 0 1
	20	111	0 0 1
	30	111	0 1 0
	40	111	0 1 0
	50	111	0 1 1

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example with Delay

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and  #(30) g1(e,A,B);
    not  #(10) g2(y,C);
    or   #(20) g3(x,e,y);
endmodule
```

t=0:

events: A:0/1 B:0/1 C:0/1

simulations: g1(AND) A=1,B=1,t=30(30)

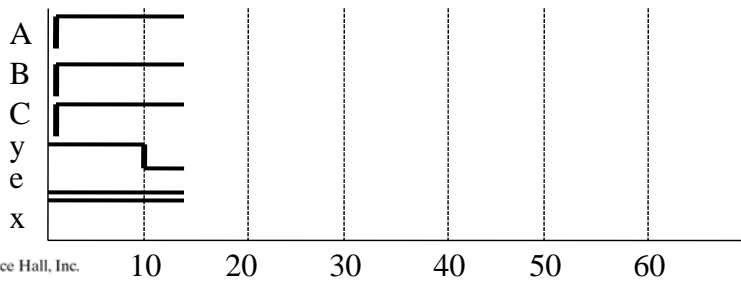
g2(NOT) C=1,t=10(10)

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Simulation Scheduled at 10 Simulation Scheduled at 30

Verilog Example (Waveform)

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and  #(30) g1(e,A,B);
    not  #(10) g2(y,C);
    or   #(20) g3(x,e,y);
endmodule
```



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example with Delay

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and  #(30) g1(e,A,B);
    not  #(10) g2(y,C);
    or   #(20) g3(x,e,y);
endmodule
```

t=10:

events: y:1/0

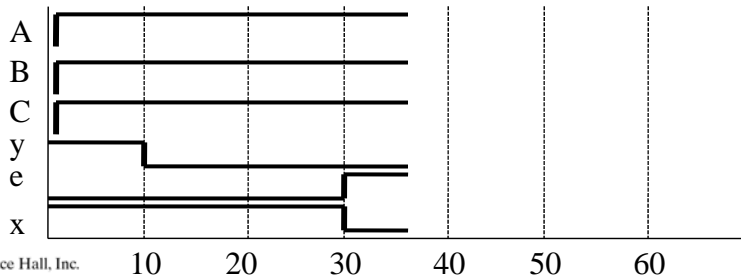
simulations: g3(OR) e=0,y=0,t=20(30)

Another Simulation Scheduled at 30

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example (Waveform)

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and   #(30) g1(e,A,B);
    not   #(10) g2(y,C);
    or    #(20) g3(x,e,y);
endmodule
```



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example with Delay

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire  e;
    and   #(30) g1(e,A,B);
    not   #(10) g2(y,C);
    or    #(20) g3(x,e,y);
endmodule
```

t=30:

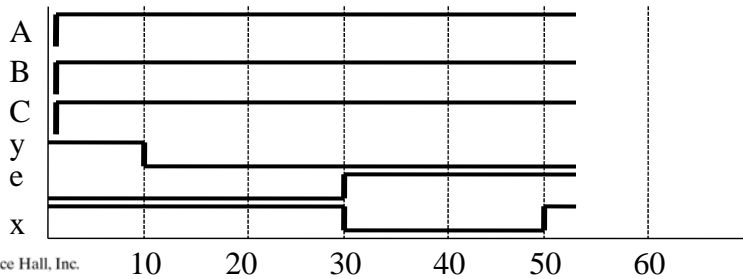
events: e:0/1 x:1/0

simulations: g3(OR) e=1,y=0,t=20(50)

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example (Waveform)

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire e;
    and # (30) g1 (e,A,B);
    not # (10) g2 (y,C);
    or # (20) g3 (x,e,y);
endmodule
```



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Example with Delay

```
// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
    input  A,B,C;
    output x,y;
    wire e;
    and # (30) g1 (e,A,B);
    not # (10) g2 (y,C);
    or # (20) g3 (x,e,y);
endmodule
```

t=50:

events: x:0/1

simulations: NONE

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Test Benches

- Only Used FOR SIMULATION
- Another Verilog Module that:
 - Provides Input
 - Allows Various Modules to be Interconnected
 - Contains Simulation Specific Commands
- Separation into Different Modules Important:
 - Can Synthesize Modules Directly
 - Can Change Abstraction of Modules
 - “Top-Level” Design
 - Allows Large Design Teams to Work Concurrently

`always` and `initial` blocks

- Both Types of Blocks Contain Procedural (Sequential) Statements
- If More than One Procedural Statement use `begin - end`
- `Always` Block is Scheduled to Simulate/Execute at EVERY Simulation Time Cycle
- `Initial` Block is Scheduled to Simulate Only at the First Time Epoch

Verilog Example with Testbench

```

// Stimulus for simple circuit
module stimcrct;
reg A, B, C;
wire x, y;
circuit_with_delay cwd(A, B, C, x, y);
initial
  begin
    A=1'b0; B=1'b0; C=1'b0;
    #100
    A=1'b1; B=1'b1; C=1'b1;
    #100    $finish;
  end
endmodule

// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
  input  A,B,C;
  output x,y;
  wire e;
  and  # (30) g1 (e,A,B);
  not  # (10) g2 (y,C);
  or   # (20) g3 (x,e,y);
endmodule

```

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Verilog Simulation with Testbench

- Most Simulators Allow for Graphic Timing Diagrams to be Used
 - SynaptiCAD (software packaged with M. Mano textbook)
 - ModelTech (commercial simulator – free at www.model.com)
 - Cadence Verilog XL (commercial tool)
 - QuartusII (built-in timing simulator with Altera toolset)

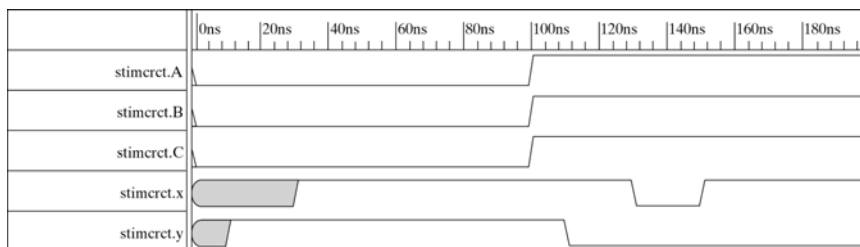


Fig. 3-38 Simulation Output of HDL Example 3-3

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Model Abstractions in Verilog

- Netlist Level Models
 - Contain enough information to construct in lab
 - structural modeling
 - Commonly “Lowest” level of abstraction
 - Useful for Transfer Among CAD tools
- RTL (register transfer language) Level
 - Composed of Boolean Expressions and Registers
 - Can be Automatically Synthesized to a netlist
 - We will work mostly at this level
- Behavioral Level
 - High-level Constructs that only Describe Functionality
 - Automatic Behavioral Synthesis Tools do Exist

Boolean Expressions in Verilog

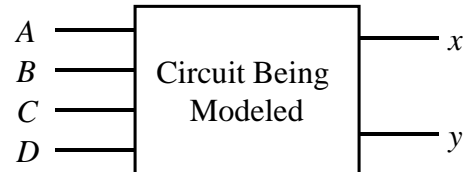
- Use the Continuous Assignment Statement
 - Keyword is **assign**
 - Boolean Operators (normal precedence):
 - & - AND
 - | - OR
 - ~ - NOT (invert)
 - When in Doubt about Precedence Use Parentheses
- Previous Example as Expression:

```
assign x = (A & B) | (~C);
```

Verilog Example

$$x = A + BC + \bar{B}D$$

$$y = \bar{B}C + B\bar{C}\bar{D}$$



```
// Circuit specified with Boolean expressions
module circuit_bln (x, y, A, B, C, D);
    input A,B,C,D;
    output x,y;
    assign x = A | (B & C) | (~B & D);
    assign y = (~B & C) | (B & ~C & ~D);
endmodule
```

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

User Defined Primitives (UDPs)

- Keywords **and**, **or**, **not**, **xor**, etc. are System Primitives
- Can Define your Own Primitives (UDPs)
- Can do this in a variety of ways including Truth Tables
- Instead of **module/endmodule** use the keywords **primitive/endprimitive**
- Only one output and must be listed first
- Keywords **table** and **endtable** used
- Input values listed in order with a **:**
- Output is always last entry followed by **;**

UDP Verilog Example

```
// User defined primitive (UDP)
primitive crctp (x, A, B, C);
  output x;
  input A, B, C;
  // Truth table for x(A,B,C)=Σ(0,2,4,6,7)
  table
  //   A B C : x (note: this is a comment)
      0 0 0 : 1;
      0 0 1 : 0;
      0 1 0 : 1;
      0 1 1 : 0;
      1 0 0 : 1;
      1 0 1 : 0;
      1 1 0 : 1;
      1 1 1 : 1;
  endtable
endprimitive

// Instantiate primitive
module declare_crctp;
  reg x,y,z;
  wire w;
  crctp (w, x, y, z);
endmodule
```

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Some Verilog Operators

Reduction Operators

~	negation
&	bitwise AND
	bitwise OR
~&	bitwise NAND
~	bitwise NOR
^	bitwise XOR
~^	bitwise XNOR
^~	bitwise XNOR

Arithmetic Operators

+	unary (sign) plus
-	unary (sign) minus
+	binary plus (add)
-	binary minus (sub)
*	multiply
/	divide
%	modulus

Logical Operators

!	logical negation
==	logical equality
!=	logical inequality
&&	logical AND
	logical OR

Verilog Values and Constants

Four Basic Values

0 logic-0 or false
1 logic-1 or true
x unknown value
z high-impedance (open)

(z at input usually treated as x)

Constants

integers
reals
strings

_ can be embedded

Specifying Values

Simple Decimal
int, real
Base Format Notation
int
Scientific
real
Double Quotes
strings

Base Format Notation Examples

Format is:

	[size]'base value
5'O37	5-bit octal
4'D2	4-bit decimal
4'B1x_01	4-bit binary
7'Hx	7-bit x (x extended) xxxxxxxx
4'hZ	4-bit z (z extended) zzzz
4'd-4	ILLEGAL : value cannot be negative
8'h 2A	spaces allowed between size and ` and between base and value
'o721	9-bit octal
'hAF	8-bit hex
10'b10	10-bit padded to left 000000010
10'bx0x1	10-bit padded to left xxxxxxxx0x1
3'b1001_0011	same as 3'b011
5'H0FFF	same as 5'H1F

Verilog Data Types

- Net Types
 - Represents Physical Connection Between Structural Elements
 - Value is Determined from Value of Drivers
 - Continuous `assign` Statement
 - Output of Gate or UDP
 - If no Driver is Present Defaults to value of `z`
- Register Type
 - Abstract Data Storage Element
 - Assigned Values Only within `always` or `initial` statement
 - Value is Saved from one Assignment to the Next
 - Default value is `x`

Verilog Data Types

- Net Types
 - `wire, tri` - most common, default is `z`
 - `wor, trior` - emulates wired-OR with multiple drivers
 - `wand, triand` - emulates wired-AND with multiple drivers
 - `triereg` - stores a value like `reg` for modeling capacitive nets
 - `tri1, tri0` - default values are 1(0)
 - `supply0, supply1` - used to model power connections for 0 and 1 values
- Register Type
 - `reg` - most common, default is `x`
 - `integer` - used for storing integers, typical use in behavioral model
 - `time` - used for storing/manipulating time values
 - `real` - used storing reals, typical use in behavioral model
 - `realtime` - same as `real`

Busses and Multi-bit Registers

- Can use “array-type” Notation
- Examples:

```
wire [2:0] Bname           // A 3-bit bus called Bname  
  
reg [7:0] Accumulator     // An 8-bit register named Accumulator
```

- Suggestions
 - Always number from MSb to LSb
 - Matches the Radix Power in Radix Polynomial
 - Consistency Helps to Avoid Bugs

4-Valued Net Fanin Tables

wire/ tri	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

- Two Names for Same Data type

Wired Nets

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

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

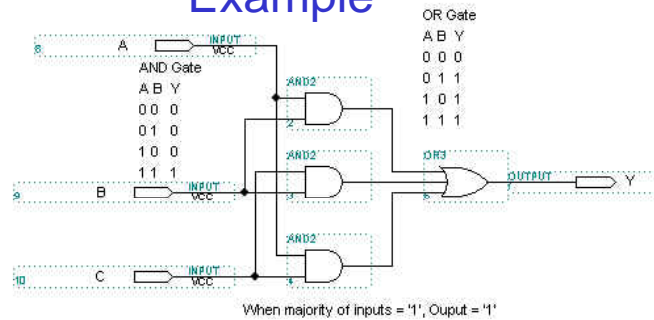
- Used to Model Wired Logic
- Assumes Equal Strength Drivers

Verilog Logic Gate Primitives

Name	Description	Usage
and	$f = (a \cdot b \dots)$	and (<i>f</i> , <i>a</i> , <i>b</i> ,...)
nand	$f = \overline{(a \cdot b \dots)}$	nand (<i>f</i> , <i>a</i> , <i>b</i> ,...)
or	$f = (a + b + \dots)$	or (<i>f</i> , <i>a</i> , <i>b</i> ,...)
nor	$f = \overline{(a + b + \dots)}$	nor (<i>f</i> , <i>a</i> , <i>b</i> ,...)
xor	$f = (a \oplus b \oplus \dots)$	xor (<i>f</i> , <i>a</i> , <i>b</i> ,...)
xnor	$f = (a \odot b \odot \dots)$	xnor (<i>f</i> , <i>a</i> , <i>b</i> ,...)
not	$f = a$	not (<i>f</i> , <i>a</i>)
buf	$f = a$	buf (<i>f</i> , <i>a</i>)
notif0	$f = (!e?a : 'bz)$	notif0 (<i>f</i> , <i>a</i> , <i>e</i>)
notif1	$f = (e?a : 'bz)$	notif1 (<i>f</i> , <i>a</i> , <i>e</i>)
buf0	$f = (!e?a : 'bz)$	buf0 (<i>f</i> , <i>a</i> , <i>e</i>)
buf1	$f = (e?a : 'bz)$	buf1 (<i>f</i> , <i>a</i> , <i>e</i>)

Table A.2. Verilog gates.

Example



```
// Combinational Logic Circuit
module maj_circ(Y, A, B, C);
  input A, B, C;
  output Y;
  and U1 (x1,A,B);
  and U2 (x2,A,C);
  and U3 (x3,B,C);
  or U4 (Y,x1,x2,x3);
endmodule
```

Majority Gate with Temporary Signals

The following version of the majority gate uses some temporary “wires”

```
// Majority Logic Circuit
module maj_circ(Y, A, B, C);
  input A, B, C;
  output Y;
  wire x1, x2, x3; //Optional
  assign x1 = A&B;
  assign x2 = A&C;
  assign x3 = B&C;
  assign Y = x1|x2|x3;
endmodule
```

Concurrent Assignment with Ternary Select

```
assign mux_out = (select==1'b0) ? q0 : q1;
```

`if` statement is procedural (sequential)

Used inside `begin – end` Block

Similar to `if` Statement but Concurrent Version

Majority Gate with conditional statement

The following version of the majority gate uses a conditional concurrent statement:

```
// Majority Logic Circuit
module maj_circ(Y, A, B, C);
    input A, B, C;
    output Y;
    assign Y = ((A&&B) || (A&&C) || (B&&C))
        ? 1'b1 : 1'b0;
endmodule
```

You will find that there are many different ways to accomplish the same result in Verilog. There is usually no best way; just use one that you feel most comfortable with.

Majority Gate using *always* block and *if* statement

```
// Majority Logic Circuit
module maj_circ(Y, A, B, C);
    input A, B, C;
    output Y;
    reg Y;
    always @(A or B or C)
    begin
        if((A==1'b1)&&(B==1'b1))
            Y = 1'b1;
        else if ((A==1'b1)&&(C==1'b1))
            Y = 1'b1;
        else if ((B==1'b1)&&(C==1'b1))
            Y = 1'b1;
        else
            Y = 1'b0;
    end
endmodule
```