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

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

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

<table>
<thead>
<tr>
<th>Time Units (ns)</th>
<th>Input ABC</th>
<th>Output y e x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>-000</td>
<td>1 0 1</td>
</tr>
<tr>
<td>Change</td>
<td>111</td>
<td>1 0 1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>111 0 0 1</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>111 0 0 1</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>111 0 1 0</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>111 0 1 0</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>111 0 1 1</td>
</tr>
</tbody>
</table>

Verilog Test Benches

- 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
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

Verilog Simulation with Testbench

• Most Simulators Allow for Graphic Timing Diagrams to be Used
  – SynaptiCAD (software packaged with text)
  – ModelTech (commercial simulator – free at www.model.com)
  – Cadence Verilog XL (commercial tool – used for lab/assignments)
Model Abstractions in Verilog

• Previous Examples are “netlists”
  – Contain enough information to construct in lab
  – structural modeling
  – Commonly “Lowest” level of abstraction

• 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

More on Gate Level Modeling

Gate Primitives

<table>
<thead>
<tr>
<th>Gate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>N-input AND gate</td>
</tr>
<tr>
<td>nand</td>
<td>N-input NAND gate</td>
</tr>
<tr>
<td>or</td>
<td>N-input OR gate</td>
</tr>
<tr>
<td>nor</td>
<td>N-input NOR gate</td>
</tr>
<tr>
<td>xor</td>
<td>N-input XOR gate</td>
</tr>
<tr>
<td>xnor</td>
<td>N-input XNOR gate</td>
</tr>
</tbody>
</table>
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 Parantheses

• Previous Example as Expression:

  \[
  \text{assign } x = (A \& B) | (\sim C);
  \]

Verilog Example

\[
\begin{align*}
  x &= A + BC + \overline{B}D \\
  y &= \overline{BC} + B\overline{C}D
\end{align*}
\]

// 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) | (\sim B & D);
  assign y = (\sim B & C) | (B & \sim C & \sim D);
endmodule
Verilog Operators

Reduction Operators
~ negation
& bitwise AND
| bitwise OR
~& bitwise NAND
~| bitwise NOR
^ bitwise XOR
~^ bitwise XNOR

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

Logical Operators
! logical negation
&& logical AND
|| logical OR

Relational Operators
!= not equal
== equal
>= greater or equal
<= less or equal
> greater
< less

Shift Operators
>> right shift
<< left shift

Concatenation Operators
\{\ldots,\ldots\}
i.e. \{a,b[3:0],2'b00\}

Replication Operators
\{n\{m\}\}
i.e. \{3\{a\}\} : \{a,a,a\}

Conditional Operators
\text{cond\_expr ? true\_expr : false\_expr}

Table: Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>+, −</td>
</tr>
<tr>
<td>Multiply</td>
<td>*, /, %</td>
</tr>
<tr>
<td>Divide</td>
<td>\ldots</td>
</tr>
<tr>
<td>Add, Subtract, Shift</td>
<td>+, −, &lt;&lt;, &gt;&gt;</td>
</tr>
<tr>
<td>Relation, Equality</td>
<td>=, !, &amp;=, |, \ldots</td>
</tr>
<tr>
<td>Reduction</td>
<td>&amp; ^, \ldots</td>
</tr>
<tr>
<td>Logic</td>
<td>&amp;</td>
</tr>
<tr>
<td>Conditional</td>
<td>?:</td>
</tr>
</tbody>
</table>

Courtesy of www.asic-world.com
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: 

\[ \text{[size]} \text{'} \text{base value} \]

- 5’O37  5-bit octal
- 4’D2  4-bit decimal
- 4’B1x_01  4-bit binary
- 7’Hx  7-bit x (x extended) xxxxxxx
- 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 xxxxxxx0x1
- 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 - wired outputs OR together (models ECL)
  wand, triand - wired outputs AND together (models open-collector)
  trireg - retain last value, when driven by \( z \) (tristate)
  tri1, tri0 - wire pull-up or pull-down when no drivers
  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:

  ```vhdl
  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

---

Conditional Statement: if-else

- The if - else statement controls the execution of other statements

  ```vhdl
  Syntax : if     if (condition)statements;
  Syntax : if-else
  if (condition)
  statements;
  else
  statements;

  Syntax : nested if-else-if
  if (condition)
  statements;
  else if (condition)
  statements;
  ............
  ............
  else
  statements;
  ```
**Conditional Statement: Case**

- The case statement compares an expression to a series of cases and executes the statement or statement group associated with the first matching case:
  - case statement supports single or multiple statements.
  - Group multiple statements using begin and end keywords.
- Syntax of a case statement look as shown below.

```plaintext
case ()
  < case1 > : < statement >
  < case2 > : < statement >
  ....
  default : < statement >
endcase
```

**Examples**

```
Example - simple if

module simple_if();
  reg latch;
  wire enable,din;
  always @(enable or din)
    if (enable) begin
      latch <= din;
    end
endmodule
```

Courtesy of www.asic-world.com
Examples

+ Example: case

```
module max (a, b, c, d, sel, y);
  input a, b, c, d;
  input [1:0] sel;
  output y;

  reg y;

  always @ (a or b or c or d or sel)
  case (sel)
    0 : y = a;
    1 : y = b;
    2 : y = c;
    3 : y = d;
    default : $display("Error in SEL");
  endcase

endmodule
```

Courtesy of www.asic-world.com

Procedure Blocks

- Easier to model sequential design and large combinational circuits
- Verilog behavior code are specified in procedure blocks
- There are two types of procedural blocks in Verilog:
  - initial: initial blocks execute only once at time zero (start execution at time zero).
  - always: always blocks loop to execute over and over again; in other words, as the name suggests, it executes always.
Behavioral Modeling

• 2 Kinds of Behavioral Keywords
  \textit{initial} \hspace{1cm} \textit{always}
• Keywords Followed by a Statement of Block
• Block is Group of Statements Enclosed by \textit{begin}
  and \textit{end} Keywords
• Can Have More Than One \textit{initial} or \textit{always} in a module
• Statements within a Block Execute Sequentially
  Unless You Control them with Delays
• Unlike Statements not Occurring in a Block; they can Execute in ANY Order

\textbf{initial} Blocks/Statements

• Scheduled to Simulate at time=0 Only
• Used to Initiate a Simulation
• We Have Used These in testbenches to Supply Input Signal Values
• We also Used Explicit Delays to Control When Each Input Signal Value Changed
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 #(20) g2(y,C);
or #(10) g3(x,e,y);
endmodule

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