Digital System Design
Verilog-Part I

Amir Masoud Gharehbaghi
amgh@mehr.sharif.edu
Verilog HDL Basics

- Syntax is like C Language
  - Case Sensitive
- Different from C (or any Software) Language
  - Timing
  - Concurrency
Verilog Lexical Elements

- Whitespaces
  - Blank Space (\b)
  - Tab (\t)
  - Newline (\n)
Verilog Identifiers

- Simple Identifiers
  - Begin with:
    - Underscore ( _ )
    - Alphabetic Characters ( a-z A-Z)
  - Contain:
    - Underscore ( _ )
    - Alphanumeric Characters ( a-z A-Z 0-9)
    - Dollar Sign ( $ )
  - Example: a1 clk _identifier_$1
Verilog Identifiers (cont.)

- Escaped Identifiers
  - Begin with Backslash ( \ )
  - End with Whitespace
  - Contain any Printable ASCII Character
  - Example:
    - \a+b-c
    - \**an_escaped_identifier**
Verilog Logic System

- 4 Value Logic System:
  - 0: Zero, Low, False
  - 1: One, High, True
  - X: Unknown, Conflict
  - Z: High Impedance, Disconnection
Verilog Data Types

Nets
- Connection between Hardware Elements
- Needs a Driver to Continuously Assign a value
- They are 1 Bit unless other specified
- "wire" is a kind of net
- Default value of a "wire" is X if Driven otherwise Z
- Example:
  - wire a = 0;
  - wire b1, b2, b3;
Verilog Data Types (cont.)

- Registers
  - Represent a Storage Element
    - Not a Hardware Register!
  - Hold the Last Value Assigned to them
  - "reg" is a 1 bit register
  - Default value of a "reg" is X
- Example:
  - reg reset, countup;
  - reg r1;
Verilog Data Types (cont.)

- Vectors
  - Nets and reg data types can have more than 1 bit width
  - Example:
    - wire [7:0] busA, busB, busC;
    - wire [0:31] data_out;
    - reg [0:7] addr;
  - Range is [msb:lsb]
Verilog Data Types (cont.)

- Arrays
  - Only One Dimensional
  - Can be Defined for registers and vectors of regs

Example:
  - `reg r1[0:100];`
  - `reg [7:0] mem [1023:0]`
Modules

- A Hardware Element is Modeled as a "module"
- Modules have ports to communicate with outside
- Behavior of a module is defined internally
Module Declaration

module MOD_NAME list_of_ports ;
//declarations come here

// behavior of module comes here

//* module body is concurrent
// so the order of appearing the constructs
// is not important! */

endmodule
Example: SR-Latch

module SR_Latch (q, q_bar, s_bar, r_bar);
    // port declarations
    input s_bar, r_bar;
    output q, q_bar;

    // functionality comes here

endmodule
Example: Module without port

module testbench ;
    // declarations come here

    // functionality comes here

endmodule
Port Declaration

- Ports can be:
  - input
  - output
  - inout

- Ports are 1 bit by default
  - They can be defined as vector

- Ports are of type Net by default
  - output ports can be defined of Register type
Example: 4 bit adder

module adder4 (s, cout, a, b, cin);
    input cin; // 1 bit
    input [3:0] a, b; // 4 bits
    output cout; // 1 bit
    output [3:0] s; // 4 bits
    reg [3:0] s;
    reg cout;
    // functionality comes here
endmodule
An Element is Defined as a Netlist or interconnection of other Elements

Elements are:
- Predefined
  - Gate Level
  - Switch Level
- Userdefined
  - Modules
Gate Types

- **N-Input 1-Output**
  - and, or, xor,
  - nand, nor, xnor

- **1-Input N-Output**
  - not, buf

- **Tri-State**
  - notif0, bufif0
  - notif1, bufif1
N-Input Gates

gate_type GATE_NAME (output, inputs);

Example:
    and a1 (o1, in1, in2);
    xnor (o2, in1, in2, in3, in4);
## Truth Table of 2 Input Gates

### AND

<table>
<thead>
<tr>
<th>and</th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### OR

<table>
<thead>
<tr>
<th>or</th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### XOR

<table>
<thead>
<tr>
<th>xor</th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Example: SR-Latch

module SR_Latch (q, q_bar, s_bar, r_bar);
    // port declarations
    input s_bar, r_bar;
    output q, q_bar;
    // functionality comes here
    nand n1 (q, s_bar, q_bar);
    nand n2 (q_bar, r_bar, q);
endmodule
N-Output Gates

gate_type GATE_NAME (outputs, input);

Example:

not n1 (o, i1);
buf (o1, o2, o3, in);
<table>
<thead>
<tr>
<th>buf</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>not</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Tri-State Gates

gate_type GATE_NAME (output, input, ctrl);

Example:

notif1 n1 (o1, i1, c1);
bufif0 (outp, inp, control);
**Truth Table of Tri-State Gates**

<table>
<thead>
<tr>
<th>Input</th>
<th>Ctrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>notif1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
</tr>
</tbody>
</table>
Gate Delays

- Three Types of delays:
  - Rise Delay: Transition to 1 from any value
  - Fall Delay: Transition to 0 from any value
  - Turn-off Delay: Transition to Z from any value

- Delay Elements:
  - min Value: Minimum Delay
  - typ Value: Typical Delay
  - max Value: Maximum Delay
Gate Delay Models

- **1 delay**  #(delay)
  - One Delay for all output changes

- **2 delays**  #(rise, fall)
  - Rise and Fall Delays are Specified
  - Turn-off Delay is the Minimum of Rise and Fall

- **3 delays**  #(rise, fall, turn-off)
  - Rise, Fall, and Turn-off Delays are Specified
  - Delay of Transition to X is always Minimum of the other Three Delays
Delay Specification

gate_type delay GATE_NAME (ports);

Example:

and #(10) (m, n, p);

or #(2:3:5) o1 (a1, a2, a3);

nand #(5, 7) nn1 (outp, in1, in2, in3);

notif1 #(1:2:3, 2:3:4, 3:4:5) (o1, i1, c);
Example: 2 to 1 Multiplexer

module Mux2x1 (o, i1, i2, ctrl);
    input i1, i2, ctrl;
    output o;
    bufif0 #(2, 2, 3) (o, i1, ctrl);
    bufif1 #(2, 3, 3) (o, i2, ctrl);
endmodule
Port Connection Rules
Methods of Port Connection

- By Ordered List
  - Signals are Connected to the Corresponding Ports in the Same Order of Appearing

- By Port Name
  - Signals are Explicitly Specified to which port they are connected
  - Order is not important
Example: 4 to 1 Multiplexer

module Mux4x1 (o, inp, ctrl);
  output o; input [3:0] inp; input [1:0] ctrl;
  wire [1:0] mid;
  Mux2x1 m0 (mid[0], inp[0], inp[1], ctrl[0]);
  Mux2x1 m1 (mid[1], inp[2], inp[3], ctrl[0]);
  Mux2x1 m2 (o, mid[0], mid[1], ctrl[1]);
endmodule
Example: 4 to 1 Multiplexer

module Mux4x1 (o, inp, ctrl);
    output o; input [3:0] inp; input [1:0] ctrl;
    wire [1:0] mid;
    Mux2x1 (.o(mid[0]), .i1(inp[0]), .i2(inp[1]), .ctrl(ctrl[0]));
    Mux2x1 (.ctrl(ctrl[0]), .i1(inp[2]), .i2(inp[3]), .o(mid[1]));
    Mux2x1 (o, mid[0], mid[1], ctrl[1]);
endmodule
Example: 4 bit adder

module adder4 (s, cout, a, b, cin);
    input cin; input [3:0] a,b;
    output cout; output [3:0] s; wire c1, c2, c3;
    FA f4 (s[3], cout, a[3], b[3], c3);
    FA f3 (s[2], c3, a[2], b[2], c2);
    FA f2 (s[1], c2, a[1], b[1], c1);
    FA f1 (s[0], c1, a[0], b[0], cin);
endmodule
Example: 8 to 1 Multiplexer

module Mux8x1 (o, inp, ctrl);
    output o; input [7:0] inp, input [2:0] ctrl;
    wire o1, o2;
    Mux4x1 (o1, inp[3:0], ctrl[1:0]);
    Mux4x1 (o2, inp[7:4], ctrl[1:0]);
    Mux2x1 (o, o1, o2, ctrl[2]);
endmodule