Overview

Part 1

- Verilog Basics
  - Notation
  - Keywords & Constructs
  - Operators

- Types of Descriptions
  - Structural
  - Dataflow
    - Boolean Equations
    - Conditions using Binary Combinations
    - Conditions using Binary Decisions

- Combinational Circuit Testbench
Verilog Notation - 1

- **Verilog is:**
  - Case sensitive
  - Based on the programming languages C and ADA

- **Comments**
  - Single Line
    ```markdown
    // [end of line]
    ```
  - Multiple Line
    ```
    /*
    */
    ```

- **List element separator:** `,`
- **Statement terminator:** `;`
Verilog Notation - 2

- **Binary Values for Constants and Variables**
  - 0
  - 1
  - X or x – Unknown
  - Z or z – High impedance state (open circuit)

- **Constants**
  - \( \text{n'b[integer]}: \) 1'b1 = 1, 8'b1 = 00000001, 4'b0101 = 0101, 8'bxxxxxxxx, 8'bxxxx = 0000xxxx
  - \( \text{n'h[integer]}: \) 8'hA9 = 10101001, 16'hf1 = 0000000011110001

- **Identifier Examples**
  - Scalar: A,C,RUN,stop,m,n
  - Vector: \text{sel[0:2]}, \text{f[0:5]}, \text{ACC[31:0]}, \text{SUM[15:0]}, \text{sum[15:0]}
Verilog Keywords & Constructs - 1

- Keywords are lower case

- **module** – fundamental building block for Verilog designs
  - Used to construct **design hierarchy**
  - Cannot be **nested**

- **endmodule** – ends a module – not a statement
  - => no “;”

- Module Declaration
  - **module** *module_name* (*module_port, module_port, ...);
  - Example: **module** *full_adder* (*A, B, c_in, c_out, S*);
Verilog Keywords & Constructs - 2

- **Input Declaration**
  - Scalar
    - `input list of input identifiers;`
    - Example: `input A, B, c_in;`
  - Vector
    - `input[range] list of input identifiers;`
    - Example: `input[15:0] A, B, data;`

- **Output Declaration**
  - Scalar Example: `output c_out, OV, MINUS;`
  - Vector Example: `output[7:0] ACC, REG_IN, data_out;`
Verilog Keywords & Constructs - 3

- **Primitive Gates**
  - `buf, not, and, or, nand, nor, xor, xnor`
  - **Syntax:** `gate_operator instance_identifier (output, input_1, input_2, ...)`
  - **Examples:**
    ```
    and A1 (F, A, B); // F = A B
    or O1 (w, a, b, c)
    O2 (x, b, c, d, e); // w = a + b + c, x = b + c + d + e
    ```
Verilog Operators - 1

- Bitwise Operators
  ~ NOT
  & AND
  | OR
  ^ XOR
  ^~ or ~^ XNOR

- Example:
  ```verilog
  input [3:0] A, B;
  output [3:0] Z;
  assign Z = A | ~B;
  ```
Verilog Operators - 2

- **Arithmetic Operators**
  - `+`, `-`, (plus others)

- **Logical & Relational Operators**
  - `!`, `&&`, `||`, `==`, `!=`, `>=`, `<=`, `>`, `<` (plus others)

- **Concatenation & Replication Operators**
  - `{identifier_1, identifier_2, ...}
  - `{n{identifier}}
  - Examples: `{REG_IN[6:0], Serial_in},
               {8 {1'b0}}`
### 1st Example to Illustrate Verilog Description Types

- **IC7283** – a 1-bit adder from a commercial IC

![IC7283 Circuit Diagram]

- **Circuit Diagram**
  - Inputs: `A0`, `B0`
  - Carries: `C0`, `C1`
  - Sum: `S0`
Structural Verilog

- Circuits can be described by a netlist as a text alternative to a diagram – Example IC7283

```verilog
module IC7283 (A0, B0, C0, C1, S0);
    input A0, B0, C0;
    output C1, S0;

    // Seven internal wires needed
    wire N1, N2, N3, N4, N5, N6, N7;

    // Ports on primitive gates - output port is listed first
    not G1 (N3,C0), G2 (N5,N2), G3 (N6,N3);
    nand G4 (N1,A0,B0);
    nor G5 (N2,A0,B0), G6 (C1,N2,N4);
    and G7 (N4,N1,N3), G8 (N7,N1,N5);
    xor G9 (S0,N6,N7);

endmodule
```
Dataflow Verilog - 1

- Circuit function can be described by `assign` statements using Boolean equations – Example IC7283

```verilog
module IC7283_df1 (A0, B0, C0, C1, S0);

input A0, B0, C0;
output C1, S0;
wire N1, N2;

assign N1 = ~(A0 & B0); //Note: // Cannot write ~& for NAND
assign N2 = ~(A0 | B0);
assign C1 = ~((N1 & ~C0) | N2);
assign S0 = (~N2 & N1)^(~(~C0));

endmodule
```
## 2nd Example to Illustrate Verilog Description Types

### Priority Encoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4</td>
<td>D3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>
Dataflow Verilog - 2

- Circuit function can be described by `assign` statements using the conditional operator with binary combinations as in a truth table – Example Priority Encoder

```verilog
module priority_encoder_df2 (D, A, V);
    input [4:0] D;
    output [2:0] A;
    output V;
    // Conditional: (X) ? Y : Z - if X is true, then Y, else Z
    assign A = (D[4] == 1'b1) ? 3'b100 :
                   (D[3] == 1'b1) ? 3'b011 :
                   (D[2] == 1'b1) ? 3'b010 :
                   (D[1] == 1'b1) ? 3'b001 :
                   (D[0] == 1'b1) ? 3'b000 : 3'bxxx;
    assign V = (D == 5'b00000) ? 1'b0 : 1'b1;
endmodule
```
Dataflow Verilog - 3

- Circuit function can be described by `assign` statements using the `conditional operator` for `binary decisions` on inputs

```verilog
module priority_encoder_df3 (D, A, V);

input [4:0] D;
output [2:0] A;
output V;

// Conditional: (X) ? Y: Z If X is true,
// then Y, else Z
    ? 3'b010: D[1] ? 3'b001: D[0] ? 3'b000: 3'bxxx;
assign V = (D == 5'b00000) ? 1'b0: 1'b1;
endmodule
```
Combinational Circuit Testbench

- **Testbench** – a program that is used to verify the correctness of an HDL representation for a circuit
  
  - Applies a set of input combinations to the circuit that thoroughly tests the HDL representation
  - Assists in verifying the correctness of the corresponding outputs:
    - By displaying the inputs and outputs for manual verification
    - By comparing the outputs to those of a known correct representation for the circuit
Combinational Testbench Example - 1

- A Verilog testbench for a combinational circuit with $n$ inputs and $m$ outputs that:
  - applies all possible binary input combinations as stimuli to the circuit
  - provides the circuit outputs for manual verification

- Stimuli Generation
  - Uses a clocked binary up counter to generate stimuli
  - Stops the simulation when all combinations have been applied
Combinational Testbench Example - 2

- **Verilog code:**

```verilog
`timescale 1 ns / 100 ps //sets time unit /
  // simulation interval (ps - picoseconds (10^-12 sec)
module priority_encoder_testbench;
  reg clk; //clock to advance counter
  reg[4:0] stim; //stimuli counter storage for D[4:0]
  wire[3:0] results; //wires to hold A[1:0],V
  //Following instantiates the circuit being tested
  priority_encoder_df3 x1 (stim[4:0], results [3:1], results [0]);
  initial //initializes clk and stim to 0 and
  //stops simulation after 33 combinations applied
  begin
    clk <= 0;
    stim <= 4'b0;
    #330 $stop;
  end
```

Verilog - Part 1  18
Combinational Testbench Example - 3

- Verilog code (continued):

```verilog
always
begin
#5 //waits 5 time units before toggling clock
forever
#5 clk <= ~clk; //toggles clock every 5 time units
end

always@(posedge clk)//conditions increment on
  // the positive edge of clk
begin
stim <= stim + 1; //increments counter by 1
end
endmodule
```
Terms of Use

- © 2004 by Pearson Education, Inc. All rights reserved.
- The following terms of use apply in addition to the standard Pearson Education Legal Notice.
- Permission is given to incorporate these materials into classroom presentations and handouts only to instructors adopting Logic and Computer Design Fundamentals as the course text.
- Permission is granted to the instructors adopting the book to post these materials on a protected website or protected ftp site in original or modified form. All other website or ftp postings, including those offering the materials for a fee, are prohibited.
- You may not remove or in any way alter this Terms of Use notice or any trademark, copyright, or other proprietary notice, including the copyright watermark on each slide.
- Return to Title Page