Verilog HDL – I: Combinational Logic

Poras T. Balsara & Dinesh K. Bhatia
Center for Integrated Circuits and Systems
Department of Electrical Engineering
University of Texas at Dallas
Design Representation (Example 1)

- **Multiplexer**: Choose one of two inputs based on a control input
- **Sel**: Select line (it is a control input)
- **A, B**: Data Inputs
- **Out**: Multiplexer Output
If \( Sel \) is ‘0’, choose \( A \) and pass it on to \( Out \).

If \( Sel \) is ‘1’, choose \( B \) and pass it on to \( Out \).

The value of \( A \) and \( B \) does not matter.

- Pass both 0’s and 1’s
### Truth Table

<table>
<thead>
<tr>
<th>Sel</th>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Characteristic Table

<table>
<thead>
<tr>
<th>Sel</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

### Boolean Equation

\[ Out = \overline{Sel} \cdot A + Sel \cdot B \]
Gate Level Description

Diagram:

- Input A
- Input B
- Input Sel
- Output tmp0
- Output tmp1
- Output Selbar
- Output Out

Logic Gates and Connections:

1. A connected to an AND gate.
2. Sel connected to an inverter (Selbar).
3. tmp0 connected to an AND gate.
4. tmp1 connected to an OR gate.
5. Out connected to the output of the OR gate.

The diagram illustrates a circuit that uses logic gates to manipulate inputs A, B, and Sel to produce the outputs tmp0, tmp1, and Out.
Verilog Description (Netlist)

module multiplexer (A, B, Sel, Out);
    input A, B, Sel;
    output Out;
    wire tmp1, tmp0;
    and (tmp1, B, Sel);
    not (selbar, Sel);
    and (tmp0, A, selbar);
    or (out, tmp0, tmp1);
endmodule
What is Verilog?

- **Verilog** is a **Hardware Description Language (HDL)**
- **HDL**: A high level programming language used to model hardware.

**Hardware Description Languages**
- have special hardware related constructs.
- can be used to build models for simulation, synthesis and test.
- have been extended to the system design level
Verilog Language

- Concurrent hardware description language
  - Expresses parallelism in the hardware
- DO NOT code Verilog like a C or FORTRAN program
  - Serializes the hardware operations
  - Leads to a BIG increase in the amount of hardware synthesized
Simplified Verilog Syntax

- First line: Keyword `module` followed by design name and a list of ports
- Second line: List of inputs and outputs
  - Identify which are inputs and which are outputs
- **Ports**: Inputs and outputs of a module are called ports
  - Ports of a multiplexer
    - A, B, Sel are **input** ports
    - Out is an **output** port
  - Internal nodes are declared as **wire**
- Provide design description
- Last line: Keyword **endmodule**

```verilog
module multiplexer (A, B, Sel, Out);
    input A, B, Sel;
    output Out;
    wire tmp1, tmp0;
    and (tmp1, B, Sel);
    not (selbar, Sel);
    and (tmp0, A, selbar);
    or (out, tmp0, tmp1);
endmodule
```
Components of Verilog Module

module module_name[(port_name{,port_name})];
    [parameter declarations]
    [input declarations]
    [output declarations]
    [inout declarations]
    [wire or tri declarations]
    [reg or integer declarations]
    [function or task declarations]
    [assign continuous assignments]
    [initial block]
    [always blocks]
    [gate instantiations]
    [module instantiations]
endmodule
Example 2: 4-1 MUX (using logic equations)

module mux4x1 (d3, d2, d1, d0, sel, out);
    input d3, d2, d1, d0;
    input [1:0] sel;
    output out;

    assign out =
        d0 & (~sel[1] & ~sel[0]) |
                d1 & (~sel[1] & sel[0]) |
                d2 & (  sel[1] & ~sel[0]) |
                d3 & (  sel[1] &  sel[0]);
endmodule
Example 2: 4-1 MUX (using If-Then-Else)

module mux4x1 (d3, d2, d1, d0, sel, out);
  input d3, d2, d1, d0;
  input [1:0] sel;
  output out;
  reg out;

  always @ (d3 or d2 or d1 or d0 or sel)
  begin
    if (sel == 2'b00)
      out = d0;
    else if (sel == 2'b01)
      out = d1;
    if (sel == 2'b10)
      out = d2;
    else if (sel == 2'b11)
      out = d3;
    else
      out = 1'b0;
  end
endmodule
Example 2: 4-1 MUX (using case)

module mux4x1 (d3, d2, d1, d0, sel, out);
    input d3, d2, d1, d0;
    input [1:0] sel;
    output out;
    reg out;

    always @ (d3 or d2 or d1 or d0 or sel)
    begin
        case (sel)
            0 : out = d0;
            1 : out = d1;
            2 : out = d2;
            3 : out = d3;
            default : out = 1'b0;
        endcase
    end
endmodule
Hierarchical Design Using Verilog

- Verilog HDL supports design by creating modules of circuit components that can be used in another design.
Module Definition: A module definition is specified in a separate file.

module equiv (p, q, r)
  input p, q;
  output r;

  assign r = ~(p ^ q); //equivalence function (xnor)

endmodule
**Module Usage (instantiation):**

```verilog
module equiv4bit (a3,a2,a1,a0, b3,b2,b1,b0, eq4)
  input  a3,a2,a1,a0, b3,b2,b1,b0;
  output eq4;
  wire   r0, r1, r2, r3;

  // instantiation of modules
  equiv   equiv0 (a0,b0,r0);
  equiv   equiv1 (a1,b1,r1);
  equiv   equiv2 (a2,b2,r2);
  equiv   equiv3 (a3,br,r3);

  assign  eq4 = r0 & r1 & r2 & r3;
endmodule
```
Verilog Testbench

**Diagram:**
- **Test Inputs**
- **Device Under Test (DUT)**
- **Testbench**

Connections:
- Inputs from Test Inputs to DUT
- Outputs from DUT to Testbench
module or_gate (in1,in2,out);
  input  in1, in2;
  output out;
  assign out = in1 | in2;
endmodule

module or_test (t1, t2, z);
  output t1, t2;
  input  z;
  reg    t1, t2;
  initial
    begin
      t1 <= 1'b0;
      t2 <= 1'b0;
      #100;
      t1 <= 1'b0;
      t2 <= 1'b1;
      #100;
      t1 <= 1'b1;
      t2 <= 1'b0;
      #100;
      t1 <= 1'b1;
      t2 <= 1'b1;
      #100;
    end
endmodule

module testbench ();
  wire  n1, n2, n3;
  or_gate or_gate0(n1,n2,n3);
  or_test or_test0(n1,n2,n3);
endmodule
Testbench for 4-1 MUX

module mux4x1_test (d3, d2, d1, d0, sel, out);

    input   out;
    output  d3, d2, d1, d0;
    output  [1:0] sel;
    reg     d3, d2, d1, d0;
    reg     [1:0] sel;

    initial
        begin
            d0  <= 1'b0;
            d1  <= 1'b1;
            d2  <= 1'b0;
            d3  <= 1'b1;
            sel <= 2'd0;
            #100;
            sel <= 2'd1;
            #100;
            sel <= 2'd2;
            #100;
            sel <= 2'd3;
            #100;
        end

endmodule

module mux4x1_bench ();
    wire [1:0] sel;

    mux4x1_test tester (d3, d2, d1, d0, sel, out);
    mux4x1      mux0   (d3, d2, d1, d0, sel, out);

endmodule
4-1 MUX Simulation Results