Chapter 4 :: Hardware Description Languages

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris
Verilog Week 2
## Precedence

Defines the order of operations

<table>
<thead>
<tr>
<th>Highest</th>
<th>Lowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>~, NOT</td>
<td>?: ternary operator</td>
</tr>
<tr>
<td>*, /, %</td>
<td></td>
</tr>
<tr>
<td>+, -</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;&lt;, &gt;&gt;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;=, &gt;&gt;=</td>
<td></td>
</tr>
<tr>
<td>=, &lt;=, &gt;, &gt;=</td>
<td></td>
</tr>
<tr>
<td>==, !=</td>
<td></td>
</tr>
<tr>
<td>&amp;, ~&amp;</td>
<td></td>
</tr>
<tr>
<td>^, ~^</td>
<td></td>
</tr>
<tr>
<td>^, ~^</td>
<td></td>
</tr>
<tr>
<td></td>
<td>, ~</td>
</tr>
</tbody>
</table>

**Table:**

- **Highest:**
  - NOT
  - mult, div, mod
  - add, sub
  - shift
  - arithmetic shift
  - comparison
  - equal, not equal
  - AND, NAND
  - XOR, XNOR
  - OR, NOR

- **Lowest:**
  - ternary operator
Numbers

Format: \texttt{N'Bvalue}

\( N \) = number of bits, \( B \) = base

\texttt{N'B} is optional but recommended (default is decimal)

<table>
<thead>
<tr>
<th>Number</th>
<th># Bits</th>
<th>Base</th>
<th>Decimal Equivalent</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 3'b101 )</td>
<td>3</td>
<td>binary</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>`b11</td>
<td>unsized</td>
<td>binary</td>
<td>3</td>
<td>00…0011</td>
</tr>
<tr>
<td>8`b11</td>
<td>8</td>
<td>binary</td>
<td>3</td>
<td>000000011</td>
</tr>
<tr>
<td>8`b1010_1011</td>
<td>8</td>
<td>binary</td>
<td>171</td>
<td>10101011</td>
</tr>
<tr>
<td>3`d6</td>
<td>3</td>
<td>decimal</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>6`o42</td>
<td>6</td>
<td>octal</td>
<td>34</td>
<td>100010</td>
</tr>
<tr>
<td>8`hAB</td>
<td>8</td>
<td>hexadecimal</td>
<td>171</td>
<td>10101011</td>
</tr>
<tr>
<td>42</td>
<td>Unsizes</td>
<td>decimal</td>
<td>42</td>
<td>00…0101010</td>
</tr>
</tbody>
</table>
Bit Manipulations: Example 1

assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};

// if y is a 12-bit signal, the above statement produces:
y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0

// underscores (_) are used for formatting only to make it easier to read. Verilog ignores them.
Bit Manipulations: Example 2

Verilog:

module mux2_8(input [7:0] d0, d1,
               input s,
               output [7:0] y);

   mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
   mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule

Synthesis:
Verilog:

module tristate(input [3:0] a,
               input en,
               output [3:0] y);
assign y = en ? a : 4'bz;
endmodule

Synthesis:
module example(input  a, b, c,
              output y);
    wire ab, bb, cb, n1, n2, n3;
    assign #1 {ab, bb, cb} = ~{a, b, c};
    assign #2 n1 = ab & bb & cb;
    assign #2 n2 = a & bb & cb;
    assign #2 n3 = a & bb & c;
    assign #4 y = n1 | n2 | n3;
endmodule
module example(input  a, b, c,  
   output y);
    wire ab, bb, cb, n1, n2, n3;
    assign #1 {ab, bb, cb} =  
       ~{a, b, c};
    assign #2 n1 = ab & bb & cb;
    assign #2 n2 = a & bb & cb;
    assign #2 n3 = a & bb & c;
    assign #4 y = n1 | n2 | n3;
endmodule
Sequential Logic

• Verilog uses certain idioms to describe latches, flip-flops and FSMs
• Other coding styles may simulate correctly but produce incorrect hardware
Always Statement

General Structure:

always @(sensitivity list)
  statement;

Whenever the event in the sensitivity list occurs, the statement is executed
D Flip-Flop

module flop(input clk,
            input [3:0] d,
            output reg [3:0] q);

    always @ (posedge clk)
    q <= d;                // pronounced “q gets d”

endmodule

Any signal assigned in an always statement must be declared reg. In this case q is declared as reg

**Beware:** A variable declared reg is not necessarily a registered output. We will show examples of this later.
module flopr(input clk, 
    input reset, 
    input [3:0] d, 
    output reg [3:0] q);

    // synchronous reset 
    always @(posedge clk)
        if (reset) q <= 4'b0;
        else q <= d;

endmodule
Resettable D Flip-Flop

module flopr(input            clk,           
              input            reset,       
              input      [3:0] d,          
              output reg [3:0] q);          

    // asynchronous reset
    always @ (posedge clk, posedge reset) 
        if (reset) q <= 4'b0;       
        else       q <= d;         

endmodule
D Flip-Flop with Enable

module flopren(input            clk,
input            reset,
input            en,
input      [3:0] d,
output reg [3:0] q);

// asynchronous reset and enable
always @ (posedge clk, posedge reset)
if      (reset) q <= 4'b0;
else if (en)    q <= d;
endmodule
module latch(input clk,
            input [3:0] d,
            output reg [3:0] q);

    always @ (clk, d)
        if (clk) q <= d;

endmodule

Warning: We won’t use latches in this course, but you might write code that inadvertently implies a latch. So if your synthesized hardware has latches in it, this indicates an error.