More Verilog for
Combinational Circuits

Behavioral Modeling
## Some Verilog Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bitwise xor</td>
</tr>
<tr>
<td>~</td>
<td>Bitwise not</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>==</td>
<td>Equality</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>{}</td>
<td>Concatenate</td>
</tr>
<tr>
<td>{ {}}</td>
<td>{n{m}} replicate m n times</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Shift right</td>
</tr>
</tbody>
</table>
4 bit comparator circuit
module Comp4(A, B, AgtB, AltB, AeqB);
// Compare magnitudes of A and B
input [3:0] A, B;
output AgtB, AltB, AeqB;

assign AgtB = (A>B);
assign AltB = (A<B);
assign AeqB = (A==B);
endmodule

We, generally, don’t care what the synthesized circuit looks like. It is shown here just to show the result of synthesis.
x = 1'b0;
y = 2'b11;
z = 7'b1010101;

d = {y, x};       // d = 3'b110;
e = {y, z[4:2]};  // e = 5'b11101
{f, g} = z[2:1];  // f=1; g=0;

h = {3{y}};       // h = 6'b111111
Variable types

• So far, all of our variables have been wires; this is a “net” (or network) data type.
• “net” types are assigned continuously (as if a net or wire is attaching them).
• Also available are “reg” (or register) data types.
• “reg” types hold their value until a new value is assigned.
• For combinational circuits the difference is sometimes not important; but is for sequential circuits.
reg types

• reg types can only be assigned as part of an "always" block. Examples will follow.
• "net" types may not be assigned in an always block.
• We will use always block with syntax:

```
always @(*)
begin
  ...
  statements....
  ...
end
```

• If there is only a single statement, "begin" and "end" are not necessary (but be careful doing this).
Quartus “if” template
“if” must be within an “always” block

Simple “if...then”

```vhdl
if(<expression>)
begin
    // Statements
end
```

“begin” and “end” not necessary if there is only a single statement.

“if... then... else”

```vhdl
if(<expression>)
begin
    // Statements
end
else
begin
    // Statements
end
```
Always block with “if...else” and don’t cares

```verbatim
table 4.8

truth table of a priority encoder

<table>
<thead>
<tr>
<th>inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

reg x,y,v;

always @ (*)
begin
  if (d3 == 1)
    {x,y,v} = 3'b111;
  else
    if (d2 == 1)
      {x,y,v} = 3'b101;
    else
      if (d1 == 1)
        {x,y,v} = 3'b011;
      else
        if (d0 == 1)
          {x,y,v} = 3'b001;
        else
          {x,y,v} = 3'bxx0;
end
```

Note: “begin” and “end” statements on “if” not included.
Note also that all variables on left hand side of equations must be “reg” variables.
Case Statement

“case” must be within an “always” block

case (variable)
  expression1 :
  begin
    statement1;
    statement2;
    ... 
    statementn;
  end
expression2 :
begin
  statements...;
end
expression3, expression4,... expressionm :  // multiple cases
begin
  statements...;
end
default : statement;
endcase
module mux4( s, i, y );

input[1:0] s;
input[3:0] i;
output reg y;

always @(*)
begin
  case(s) // output "y" is determined by "s"
    0 : y = i[0]; // if s==0, y=i[0]...
    1 : y = i[1];
    2 : y = i[2];
    3 : y = i[3];
  endcase
end
endmodule

Note: “begin” and “end” statements on “if” not included.
Note also that all variables on left hand side of equations must be “reg” variables.
module mux4( s, i, y );

input[1:0] s;
input[3:0] i;
output reg y;

always @(*)
begin
   case( s )
      0 : y = i[0];
      1 : y = i[1];
      2 : y = i[2];
      3 : y = i[3];
   endcase
end
endmodule