In this lecture, we will go over examples of VHDL in comparison to SystemVerilog

Examples taken from Ch. 4 of the Harris & Harris book 2\textsuperscript{nd} Edition (recommended but not required book for this class)
HDL Example 4.3  LOGIC GATES

SystemVerilog

```verilog
module gates(input logic [3:0] a, b,
             output logic [3:0] y1, y2,
             y3, y4, y5);

    // five different two input logic
    // gates acting on 4-bit busses */
    assign y1 = a & b; // AND
    assign y2 = a | b; // OR
    assign y3 = a ^ b; // XOR
    assign y4 = ~(a & b); // NAND
    assign y5 = ~(a | b); // NOR
endmodule
```

VHDL

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity gates is
    port(a, b: in STD_LOGIC_VECTOR(3 downto 0);
         y1, y2, y3, y4, y5: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of gates is
begin
    -- five different two input logic gates
    -- acting on 4-bit busses
    y1 <= a and b;
    y2 <= a or b;
    y3 <= a xor b;
    y4 <= a and not b;
    y5 <= a or not b;
end;
```

Slide derived from Harris & Harris book
Conditional Assignment

**HDL Example 4.5 2:1 MULTIPLEXER**

**SystemVerilog**
```verilog
classic module mux2(input logic [3:0] d0, d1,
                  input logic s,
                  output logic [3:0] y);
    assign y = s ? d1 : d0;
endmodule
```

**VHDL**
```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

dentity mux2 is
    port(d0, d1 : in STD_LOGIC_VECTOR(3 downto 0);
        s    : in STD_LOGIC;
        y    : out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
    y <= d1 when s else d0;
end;
```

![Figure 4.6 mux2 synthesized circuit](image-url)
More Assign Statements

**HDL Example 4.7 FULL ADDER**

<table>
<thead>
<tr>
<th><strong>SystemVerilog</strong></th>
<th><strong>VHDL</strong></th>
</tr>
</thead>
</table>
| module fulladder(input logic a, b, cin,  
|  
| logic p, g; 
| assign p = a ^ b; 
| assign g = a & b; 
| assign s = p ^ cin; 
| assign cout = g | (p & cin);  
| endmodule |  
| library IEEE; use IEEE.STD_LOGIC_1164.all;  
|  
| entity fulladder is 
| port(a, b, cin: in STD_LOGIC; 
|  
| s, cout: out STD_LOGIC);  
|  
| end;  
|  
| architecture synth of fulladder is  
| signal p, g: STD_LOGIC;  
| begin  
| p <= a xor b;  
| g <= a and b;  
| s <= p xor cin;  
| cout <= g or (p and cin);  
| end; |

![fulladder synthesized circuit](image)

**Figure 4.8** fulladder synthesized circuit

Slide derived from Harris & Harris book
### Operators

**HDL Example 4.8 OPERATOR PRECEDENCE**

<table>
<thead>
<tr>
<th>SystemVerilog</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table 4.1 SystemVerilog operator precedence</strong></td>
<td><strong>Table 4.2 VHDL operator precedence</strong></td>
</tr>
<tr>
<td><strong>Op</strong></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
<tr>
<td>*, /, %</td>
<td>MUL, DIV, MOD</td>
</tr>
<tr>
<td>+, -</td>
<td>PLUS, MINUS</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>Logical Left/Right Shift</td>
</tr>
<tr>
<td>&lt;&lt;&lt;&lt;, &gt;&gt;&gt;&gt;</td>
<td>Arithmetic Left/Right Shift</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>Relative Comparison</td>
</tr>
<tr>
<td>==, !=</td>
<td>Equality Comparison</td>
</tr>
<tr>
<td>&amp;, &amp;&amp;</td>
<td>AND, NAND</td>
</tr>
<tr>
<td>^, ~^</td>
<td>XOR, XNOR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>Conditional</td>
</tr>
</tbody>
</table>

Slide derived from Harris & Harris book
## Numbers

**HDL Example 4.9** NUMBERS

### SystemVerilog

#### Table 4.3 SystemVerilog numbers

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Bits</th>
<th>Base Val</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b101</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>'b11</td>
<td>?</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8'b11</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8'b1010_1011</td>
<td>8</td>
<td>2</td>
<td>171</td>
</tr>
<tr>
<td>3'd6</td>
<td>3</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>6'c42</td>
<td>6</td>
<td>8</td>
<td>34</td>
</tr>
<tr>
<td>8'hAB</td>
<td>8</td>
<td>16</td>
<td>171</td>
</tr>
<tr>
<td>42</td>
<td>?</td>
<td>10</td>
<td>42</td>
</tr>
</tbody>
</table>

### VHDL

#### Table 4.4 VHDL numbers

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Bits</th>
<th>Base Val</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>38&quot;101&quot;</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>B&quot;11&quot;</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>88&quot;11&quot;</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>88&quot;1010_1011&quot;</td>
<td>8</td>
<td>2</td>
<td>171</td>
</tr>
<tr>
<td>30&quot;6&quot;</td>
<td>3</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>60&quot;42&quot;</td>
<td>6</td>
<td>8</td>
<td>34</td>
</tr>
<tr>
<td>B&quot;AB&quot;</td>
<td>8</td>
<td>16</td>
<td>171</td>
</tr>
<tr>
<td>&quot;101&quot;</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>B&quot;101&quot;</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>X&quot;AB&quot;</td>
<td>8</td>
<td>16</td>
<td>171</td>
</tr>
</tbody>
</table>

---

Slide derived from Harris & Harris book
### HDL Example 4.12  BIT SWIZZLING

<table>
<thead>
<tr>
<th>SystemVerilog</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>assign y = (c[2:1], {3(d[0])}, c[0], 3'b101);</code></td>
<td><code>y &lt;= (c[2 downto 1], d(0), d(0), d(0), c(0), 3'b101);</code></td>
</tr>
</tbody>
</table>
HDL Example 4.14 STRUCTURAL MODEL OF 4:1 MULTIPLEXER

**SystemVerilog**

```verilog
module mux4(input logic [3:0] d0, d1, d2, d3,
            input logic [2:0] s,
            output logic [3:0] y);

logic [3:0] low, high;

mux2 lowmux(d0, d1, s[0], low);
mux2 highmux(d2, d3, s[0], high);
mux2 finalmux(low, high, s[1], y);
endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4 is
  port(d0, d1, d2, d3 : in STD_LOGIC_VECTOR(3 downto 0);
       s : in STD_LOGIC_VECTOR(1 downto 0);
       y : out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture struct of mux4 is
  component mux2
    port(d0, d1, s : in STD_LOGIC_VECTOR(3 downto 0);
         y : out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal low, high : STD_LOGIC_VECTOR(3 downto 0);
begin
  lowmux: mux2 port map(d0, d1, s[0], low);
highmux: mux2 port map(d2, d3, s[0], high);
finalmux: mux2 port map(low, high, s[1], y);
end;
```

![Figure 4.11 mux4 synthesized circuit](image)
HDL Example 4.16  ACCESSING PARTS OF BUSSES

SystemVerilog

module mux2_8(input logic [7:0] d0, d1, input logic s, output logic [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

VHDL

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2_8 is
    port(d0, d1: in STD_LOGIC_VECTOR(7 downto 0);
        s: in STD_LOGIC;
        y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux2_8 is
    component mux2
        port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
            s: in STD_LOGIC;
            y: out STD_LOGIC_VECTOR(3 downto 0));
    end component;
begin
    lsbmux: mux2
        port map(d0(3 downto 0), d1(3 downto 0),
            s, y(3 downto 0));
    msbmux: mux2
        port map(d0(7 downto 4), d1(7 downto 4),
            s, y(7 downto 4));
end;
HDL Example 4.17 REGISTER

SystemVerilog

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

  always_ff @(posedge clk)
    q <= d;

endmodule

VHDL

library IEEE; use IEEE.STD_LOGIC_1164.all;

dentity flop is
  part(clk: in STD_LOGIC;
       d: in STD_LOGIC_VECTOR(3 downto 0);
       q: out STD_LOGIC_VECTOR(3 downto 0));

end;

architecture synth of flop is
begin
  process(clk) begin
    if rising_edge(clk) then
      q <= d;
    end if;

  end process;

end;

Figure 4.14 flop synthesized circuit
Resettable Register

HDL Example 4.18  RESETTABLE REGISTER

**SystemVerilog**

```verilog
class flop
  input logic clk,
  input logic reset,
  input logic [3:0] d,
  output logic [3:0] q;

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

module flop
  input logic clk,
  input logic reset,
  input logic [3:0] d,
  output logic [3:0] q;

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

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

type flop is
  port(clk, reset: in STD_LOGIC;
    d: in STD_LOGIC_VECTOR(3 downto 0);
    q: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture asynchronous of flop is
begin
  process(clk, reset)
  begin
    if reset then
      q <= "0000";
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;

type flop is
  port(clk, reset: in STD_LOGIC;
    d: in STD_LOGIC_VECTOR(3 downto 0);
    q: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synchronous of flop is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if reset then q <= "0000";
    else q <= d;
    end if;
  end process;
end;
```

---

**Figure 4.15 flop synthesized circuit (a) asynchronous reset, (b) synchronous reset**

Slide derived from Harris & Harris book
HDL Example 4.19  RESETTABLE ENABLED REGISTER

**SystemVerilog**

```verilog
module flopenr(input logic clk,  
                   input logic reset,  
                   input logic en,  
                   input logic [3:0] d,  
                   output logic [3:0] q);

   // asynchronous reset  
   always_ff @(posedge clk, posedge reset)  
      if (reset) q <= 4'b0;  
      else if (en) q <= d;  
   endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopenr is  
   port(clk,  
         reset,  
         en: in STD_LOGIC;  
         d: in STD_LOGIC_VECTOR(3 downto 0);  
         q: out STD_LOGIC_VECTOR(3 downto 0));  
end;

architecture asynchronous of flopenr is  
   -- asynchronous reset  
begin  
   process(clk, reset) begin  
      if reset then  
         q <= "0000";
      elsif rising_edge(clk) then  
         if en then  
            q <= d;  
         end if;
      end if;
   end process;
end;
```

**Figure 4.16** flopenr synthesized circuit
**HDL Example 4.20** SYNCHRONIZER

**SystemVerilog**

```systemverilog
module sync(input logic clk,
            input logic d,
            output logic q);
  logic n1;
  always_ff@(posedge clk)
  begin
    n1 <= d; // nonblocking
    q <= n1; // nonblocking
  end
endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sync is
  port(clk: in STD_LOGIC;
       d: in STD_LOGIC;
       q: out STD_LOGIC);
end;

architecture good of sync is
  signal n1: STD_LOGIC;
begin
  process(clk) begin
    if rising_edge(clk) then
      n1 <= d;
      q <= n1;
    end if;
  end process;
end;
```

![Figure 4.18 sync synthesized circuit](image-url)
HDL Example 4.23  FULL ADDER USING always/process

**SystemVerilog**

```verilog
module fulladder(input logic a, b, cin, output logic s, cout);

logic p, g;

always_comb
begin
  p = a ^ b;          // blocking
  g = a & b;          // blocking
  s = p ^ cin;        // blocking
  cout = g | (p & cin); // blocking
end
endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(a, b, cin: in STD_LOGIC;
       s, cout: out STD_LOGIC);
end;

architecture synth of fulladder is
begin
  process(all)
  variable p, g: STD_LOGIC;
  begin
    p := a xor b; -- blocking
    g := a and b; -- blocking
    s := p xor cin;
    cout := g or (p and cin);
  end process;
end;
```
HDL Example 4.24 SEVEN-SEGMENT DISPLAY DECODER

**SystemVerilog**

```verilog
module sevenseg(input logic [3:0] data,
                output logic [6:0] segments);
    always_comb
    case(data)
        0:    segments = 7'b1111_1110;
        1:    segments = 7'b0011_0000;
        2:    segments = 7'b1011_1111;
        3:    segments = 7'b1111_1011;
        4:    segments = 7'b1011_0011;
        5:    segments = 7'b1011_0010;
        6:    segments = 7'b1110_1011;
        7:    segments = 7'b1111_0000;
        8:    segments = 7'b1111_1111;
        9:    segments = 7'b1111_1011;
        default: segments = 7'b0000_0000;
    endcase
endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity seven_seg_decoder is
    port(data: in STD_LOGIC_VECTOR(3 downto 0);
         segments: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of seven_seg_decoder is
begin
    process(all) begin
        case data is
            when '0' => segments <= "1111110";
            when '1' => segments <= "0110000";
            when '2' => segments <= "1111101";
            when '3' => segments <= "1111001";
            when '4' => segments <= "0110011";
            when '5' => segments <= "1011011";
            when '6' => segments <= "1011111";
            when '7' => segments <= "1110000";
            when '8' => segments <= "1111111";
            when '9' => segments <= "1110011";
            when others => segments <= "0000000";
        end case;
    end process;
end;
```

**Figure 4.20 sevenseg synthesized circuit**
**HDL Example 4.25  3:8 DECODER**

**SystemVerilog**

```verilog
module decoder3_8(input logic[2:0] a,
                   output logic[7:0] y);
always_comb
  case(a)
    3'b000: y = 8'b00000001;
    3'b001: y = 8'b00000010;
    3'b010: y = 8'b00000100;
    3'b011: y = 8'b00001100;
    3'b100: y = 8'b00010000;
    3'b101: y = 8'b00010000;
    3'b110: y = 8'b00010000;
    3'b111: y = 8'b00010000;
  default: y = 8'bXXXXXXXX;
  endcase
endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity decoder3_8 is
  port(a: in STD_LOGIC_VECTOR(2 downto 0);
       y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture synth of decoder3_8 is
begin
  process(all) begin
    case a is
      when "000" => y <= "00000001";
      when "001" => y <= "00000010";
      when "010" => y <= "00000100";
      when "011" => y <= "00010000";
      when "100" => y <= "00100000";
      when "101" => y <= "01000000";
      when "110" => y <= "10000000";
      when "111" => y <= "00000000";
      when others => y <= "XXXXXXXX";
    end case;
  end process;
end;
```

Slide derived from Harris & Harris book
More If-Then-Else

**HDL Example 4.26** PRIORITY CIRCUIT

**SystemVerilog**

```verilog
module priorityckt(input logic [3:0] a,  
    output logic [3:0] y);

    always_comb
        if   (a[3]) y <= 4'b1000;
        else if (a[2]) y <= 4'b0100;
        else if (a[1]) y <= 4'b0010;
        else if (a[0]) y <= 4'b0001;
        else         y <= 4'b0000;
    endmodule
```

**VHDL**

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priorityckt is  
    port(e: in   STD_LOGIC_VECTOR(3 downto 0);
         y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of priorityckt is
begin
    process(all) begin
        if   e(3) then y <= "1000";
        elsif e(2) then y <= "0100";
        elsif e(1) then y <= "0010";
        elsif e(0) then y <= "0001";
        else         y <= "0000";
    end if;
end process;
end;
```

**Figure 4.22** priorityckt synthesized circuit
**HDL Example 4.27** PRIORITY CIRCUIT USING DON'T CARES

### SystemVerilog

```verilog
module priority_casez(input logic [3:0] a,
    output logic [3:0] y);

    always_comb
    casez(a)
    4'b1111: y <= 4'b1000;
    4'b0111: y <= 4'b0100;
    4'b0011: y <= 4'b0010;
    4'b0001: y <= 4'b0001;
    default: y <= 4'b0000;
    endcase
endmodule
```

### VHDL

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priority_casez is
    port(a: in STD_LOGIC_VECTOR(3 downto 0);
y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture dontcare of priority_casez is
begin
    process(a) begin
        case a is
            when "1111" => y <= "1000";
            when "0111" => y <= "0100";
            when "0011" => y <= "0010";
            when "0001" => y <= "0001";
            when others => y <= "0000";
        end case;
    end process;
end;
```

---

**Figure 4.23** priority_casez synthesized circuit
# Blocking vs. Non-Blocking

## Blocking and Nonblocking Assignment Guidelines

### SystemVerilog

1. Use `always_ff @(posedge clk)` and nonblocking assignments to model synchronous sequential logic.

```verilog
class_synchrony: Integer Assignment
always_ff @(posedge clk)
begin
    n1 <= d; // nonblocking
    q <= n1; // nonblocking
end
```

2. Use continuous assignments to model simple combinational logic.

```verilog
assign y = s ? d1 : d0;
```

3. Use `always_comb` and blocking assignments to model more complicated combinational logic where the always statement is helpful.

```verilog
always_comb
begin
    p = a ^ b; // blocking
    g = a & b; // blocking
    S = p ^ cin;
    cout = g | (p & cin);
end
```

4. Do not make assignments to the same signal in more than one `always` statement or continuous assignment statement.

### VHDL

1. Use `process(clk)` and nonblocking assignments to model synchronous sequential logic.

```vhdl
process(clk) begin
    if rising_edge(clk) then
        n1 <= d; -- nonblocking
        q <= n1; -- nonblocking
    end if;
end process;
```

2. Use concurrent assignments outside `process` statements to model simple combinational logic.

```vhdl
y <= d0 when s = '0' else d1;
```

3. Use `process(all)` to model more complicated combinational logic where the process is helpful. Use blocking assignments for internal variables.

```vhdl
process(all)
begin
    variable p, g: STD_LOGIC;
    p := a xor b; -- blocking
    g := a and b; -- blocking
    S := p xor cin;
    cout <= g or (p and cin);
end process;
```

4. Do not make assignments to the same variable in more than one `process` or concurrent assignment statement.
**HDL Example 4.31** PATTERN RECOGNIZER MOORE FSM

### SystemVerilog

```verilog
module patternMoore(input logic clk, 
                     input logic reset, 
                     input logic a, 
                     output logic y);

typedef enum logic [1:0] (S0, S1, S2) statetype;
statetype state, nextstate;

// state register 
always_ff @(posedge clk, posedge reset)
  if (reset) state <= S0;
  else state <= nextstate;

// next state logic
always_comb
  case (state)
    S0: if (a) nextstate = S0;
        else nextstate = S1;
    S1: if (a) nextstate = S2;
        else nextstate = S1;
    S2: if (a) nextstate = S0;
        else nextstate = S1;
    default: nextstate = S0;
  endcase

// output logic
assign y = (state == S2);
endmodule
```

### VHDL

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity patternMoore is 
  port(clk, reset: in STD_LOGIC;
        a: in STD_LOGIC;
        y: out STD_LOGIC);
end;

architecture synth of patternMoore is 
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- state register 
  process(clk, reset) begin
    if reset then state <= S0;
    else if rising_edge(clk) then state <= nextstate;
  end if;
  end process;

  -- next state logic
  process(all) begin
    case state is
      when S0 =>
        if a then nextstate <= S0;
        else nextstate <= S1;
      end if;
      when S1 =>
        if a then nextstate <= S2;
        else nextstate <= S1;
      end if;
      when S2 =>
        if a then nextstate <= S0;
        else nextstate <= S1;
      end if;
      when others =>
        nextstate <= S0;
    end case;
  end process;

  -- output logic
  y <= '1' when state = S2 else '0';
end;
```

---

*Figure 4.26* patternMoore synthesized circuit

---

Slide derived from Harris & Harris book
Parameterized Modules

**HDL Example 4.34** PARAMETERIZED N-BIT 2:1 MULTIPLEXERS

```
**SystemVerilog**

module mux2
    #(parameter width=8)
    (input logic [width-1:0] d0, d1,
     input logic s,
     output logic [width-1:0] y);
    assign y=s?d1 : d0;
endmodule

module mux4_3(input logic [7:0] d0, d1, d2, d3,
               input logic [1:0] s,
               output logic [7:0] y);
    logic [7:0] low, hi;
    mux2 lowmux(d0, d1, s[0], low);
    mux2 himux(d2, d3, s[0], hi);
    mux2 outmux(low, hi, s[1], y);
endmodule

**VHDL**

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
    generic(width: integer := 8);
    port(d0,
         d1: in STD_LOGIC_VECTOR(width 1 downto 0);
         s: in STD_LOGIC;
         y: out STD_LOGIC_VECTOR(width 1 downto 0));
end;

architecture synth of mux2 is
begin
    y <= d1 when s else d0;
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4_8 is
    port(d0, d1, d2,
         d3: in STD_LOGIC_VECTOR(7 downto 0);
         s: in STD_LOGIC_VECTOR(1 downto 0);
         y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux4_8 is
    component mux2
        generic(width: integer := 8);
        port(d0,
             d1: in STD_LOGIC_VECTOR(width 1 downto 0);
             s: in STD_LOGIC;
             y: out STD_LOGIC_VECTOR(width 1 downto 0));
    end component;
    signal low, hi: STD_LOGIC_VECTOR(7 downto 0);
begin
    lowmux: mux2 port map(d0, d1, s(0), low);
    himux: mux2 port map(d2, d3, s(0), hi);
    outmux: mux2 port map(low, hi, s(1), y);
end;
```
Parameterized Modules

HDL Example 4.34  PARAMETERIZED N-BIT 2:1 MULTIPLEXERS

SystemVerilog

module mux4_12(input logic [11:0] d0, d1, d2, d3,
               input logic [1:0] s,
               output logic [11:0] y);

logic [11:0] low, hi;

mux2 #(12) lowmux(d0, d1, s[0], low);
mux2 #(12) himux(d2, d3, s[0], hi);
mux2 #(12) outmux(low, hi, s[1], y);
endmodule

Figure 4.29 mux4_12 synthesized circuit

VHDL

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4_12 is
  port(d0, d1, d2,
       d3: in STD_LOGIC_VECTOR(7 downto 0);
       s: in STD_LOGIC_VECTOR(1 downto 0);
       y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux4_12 is
  component mux2
    generic(width: integer := 8);
    port(d0,
         d1: in STD_LOGIC_VECTOR(width 1 downto 0);
         s: in STD_LOGIC;
         y: out STD_LOGIC_VECTOR(width 1 downto 0));
  end component;
  signal low, hi: STD_LOGIC_VECTOR(7 downto 0);
begin
  lowmux: mux2 generic map(12)
    port map(d0, d1, s[0], low);
  himux: mux2 generic map(12)
    port map(d2, d3, s[0], hi);
  outmux: mux2 generic map(12)
    port map(low, hi, s[1], y);
end: