Lecture 12: Testbenches and VHDL

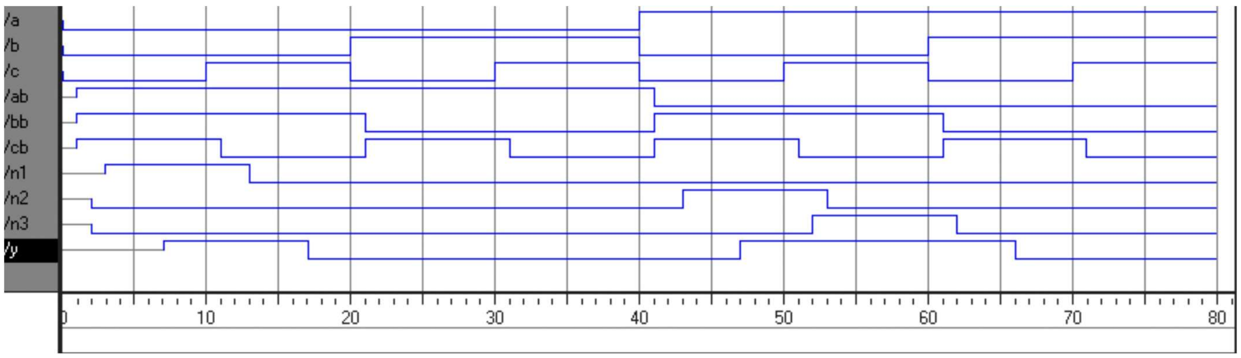
UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Testbenches

- HDL that tests another module: *device under test* (dut)
- Not synthesizeable
- Types:
  - Simple
  - Self-checking
  - Self-checking with testvectors

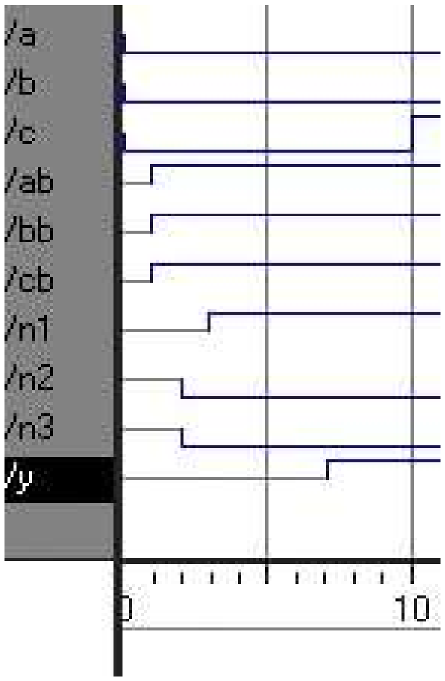
Delays

```
module example(input  logic a, b, c,
               output logic y);
  logic 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
```



Delays

```
module example(input  logic a, b, c,
               output logic y);
  logic 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
```



Testbench Example

- Write SystemVerilog code to implement the following function in hardware:

$$y = \overline{b}\,\overline{c} + a\overline{b}$$

- Name the module `sillyfunction`

Testbench Example

- Write SystemVerilog code to implement the following function in hardware:

$$y = \overline{b}\,\overline{c} + a\overline{b}$$

```
module sillyfunction(input  logic a, b, c,
                     output logic y);
  assign y = ~b & ~c | a & ~b;
endmodule
```

Simple Testbench

```
module testbench1();
  logic a, b, c;
  logic y;
  // instantiate device under test
  sillyfunction dut(a, b, c, y);
  // apply inputs one at a time
  initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
    a = 1; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
  end
endmodule
```

Self-checking Testbench

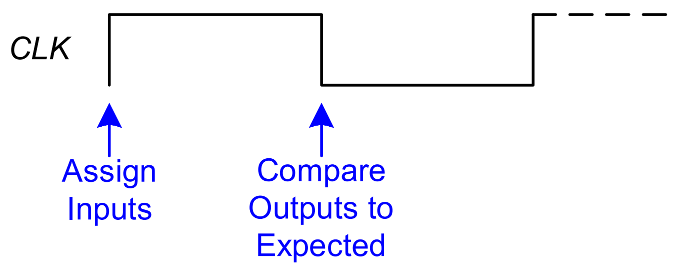
```
module testbench2();
  logic  a, b, c;
  logic y;
  sillyfunction dut(a, b, c, y);  // instantiate dut
  initial begin // apply inputs, check results one at a time
    a = 0; b = 0; c = 0; #10;
    if (y !== 1) $display("000 failed.");
    c = 1; #10;
    if (y !== 0) $display("001 failed.");
    b = 1; c = 0; #10;
    if (y !== 0) $display("010 failed.");
    c = 1; #10;
    if (y !== 0) $display("011 failed.");
    a = 1; b = 0; c = 0; #10;
    if (y !== 1) $display("100 failed.");
    c = 1; #10;
    if (y !== 1) $display("101 failed.");
    b = 1; c = 0; #10;
    if (y !== 0) $display("110 failed.");
    c = 1; #10;
    if (y !== 0) $display("111 failed.");
  end
endmodule
```

Testbench with Testvectors

- Testvector file: inputs and expected outputs
- Testbench:
  1. Generate clock for assigning inputs, reading outputs
  2. Read testvectors file into array
  3. Assign inputs, expected outputs
  4. Compare outputs with expected outputs and report errors

Testbench with Testvectors

- Testbench clock:
  – assign inputs (on rising edge)
  – compare outputs with expected outputs (on falling edge).



- Testbench clock also used as clock for synchronous sequential circuits

Testvectors File

- File: `example.tv`
- contains vectors of abc_y expected

```
000_1
001_0
010_0
011_0
100_1
101_1
110_0
111_0
```

1. Generate Clock

```
module testbench3();
  logic        clk, reset;
  logic        a, b, c, yexpected;
  logic        y;
  logic [31:0] vectornum, errors;    // bookkeeping variables
  logic [3:0]  testvectors[10000:0]; // array of testvectors

  // instantiate device under test
  sillyfunction dut(a, b, c, y);

  // generate clock
  always     // no sensitivity list, so it always executes
    begin
      clk = 1; #5; clk = 0; #5;
    end
```

2. Read Testvectors into Array

```
// at start of test, load vectors and pulse reset

initial
   begin
     $readmemb("example.tv", testvectors);
     vectornum = 0; errors = 0;
     reset = 1; #27; reset = 0;
   end


// Note: $readmemh reads testvector files written in
// hexadecimal
```

3. Assign Inputs & Expected Outputs

```
// apply test vectors on rising edge of clk
always @(posedge clk)
  begin
    #1; {a, b, c, yexpected} = testvectors[vectornum];
  end
```

4. Compare with Expected Outputs

```
// check results on falling edge of clk
   always @(negedge clk)
    if (~reset) begin // skip during reset
      if (y !== yexpected) begin
        $display("Error: inputs = %b", {a, b, c});
        $display("  outputs = %b (%b expected)",y,yexpected);
        errors = errors + 1;
      end

// Note: to print in hexadecimal, use %h. For example,
//       $display("Error: inputs = %h", {a, b, c});
```

4. Compare with Expected Outputs

```
// increment array index and read next testvector
      vectornum = vectornum + 1;
      if (testvectors[vectornum] === 4'bx) begin
          $display("%d tests completed with %d errors",
                vectornum, errors);
        $finish;
      end
    end
endmodule


// === and !== can compare values that are 1, 0, x, or z.
```

VHDL

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

VHDL

- We will go over examples of VHDL in comparison to SystemVerilog
- Examples taken from Ch. 4 of the Harris & Harris book 2nd Edition (recommended but not required book for this class)

Modules and Assign Statements

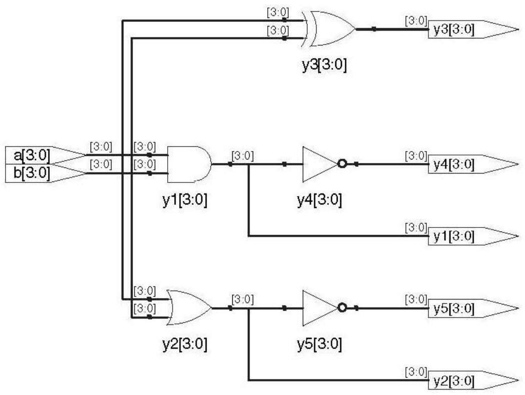
HDL Example 4.3 LOGIC GATES

SystemVerilog

```
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

```
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 nand b;
  y5 <= a nor b;
end;
```



Slide derived from Harris & Harris book

Figure 4.4 gates **synthesized circuit**

Conditional Assignment

HDL Example 4.5 2:1 MULTIPLEXER

SystemVerilog

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

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

VHDL

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

entity 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;
```

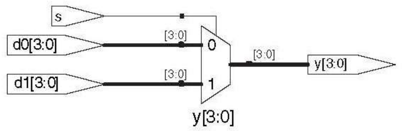


Figure 4.6 mux2 **synthesized circuit**

Slide derived from Harris & Harris book

More Assign Statements

HDL Example 4.7 FULL ADDER

SystemVerilog

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

  logic p, g;

  assign p = a ^ b;
  assign g = a & b;

  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```

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
  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;
```

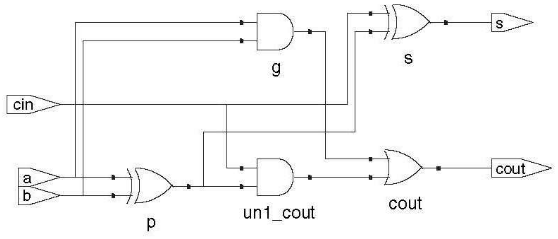


Figure 4.8 `fulladder` **synthesized circuit**

Slide derived from Harris & Harris book

Operators

HDL Example 4.8 OPERATOR PRECEDENCE

SystemVerilog

Table 4.1 **SystemVerilog operator precedence**

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	Conditional

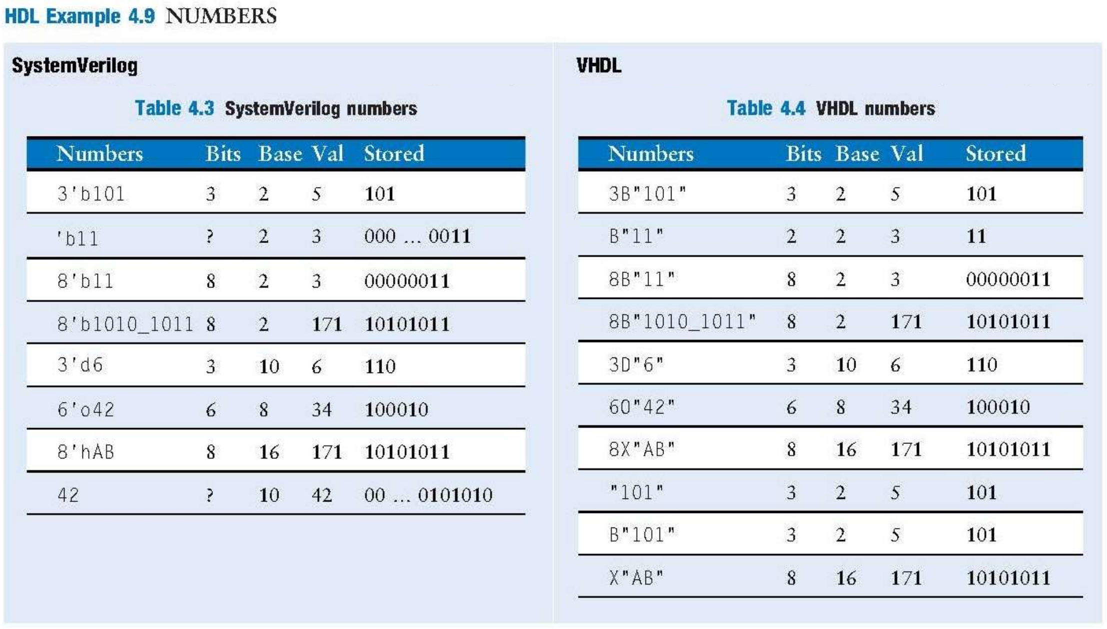
VHDL

Table 4.2 **VHDL operator precedence**

	Op	Meaning
Highest	not	NOT
	*, /, mod, rem	MUL, DIV, MOD, REM
	+, -	PLUS, MINUS
	rol, ror, srl, sll	Rotate, Shift logical
	<, <=, >, >=	Relative Comparison
	=, /=	Equality Comparison
Lowest	and, or, nand, nor, xor, xnor	Logical Operations

Slide derived from Harris & Harris book

Numbers



HDL Example 4.9 NUMBERS

SystemVerilog

Table 4.3 SystemVerilog numbers

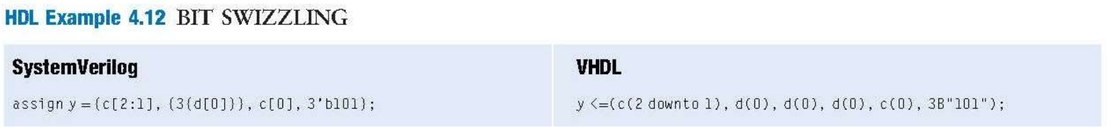
Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000 … 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00 … 0101010

VHDL

Table 4.4 VHDL numbers

Numbers	Bits	Base	Val	Stored
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011"	8	2	171	10101011
3D"6"	3	10	6	110
6O"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101"	3	2	5	101
X"AB"	8	16	171	10101011

Bit Manipulations



HDL Example 4.12 BIT SWIZZLING

SystemVerilog

```
assign y = {c[2:1], {3{d[0]}}, c[0], 3'b101};
```

VHDL

```
y <=(c(2 downto 1), d(0), d(0), d(0), c(0), 3B"101");
```

Module Instantiations

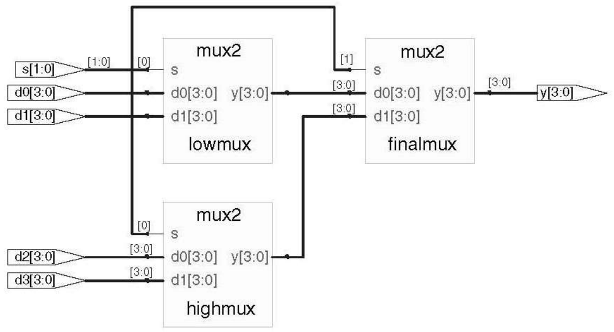
HDL Example 4.14 STRUCTURAL MODEL OF 4:1 MULTIPLEXER

SystemVerilog

```
module mux4(input  logic [3:0] d0, d1, d2, d3,
            input  logic [1:0] 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

```
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: in  STD_LOGIC_VECTOR(3 downto 0);
         s:  in  STD_LOGIC;
         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;
```



Slide derived from Harris & Harris book

Figure 4.11 mux4 **synthesized circuit**

Module Instantiations

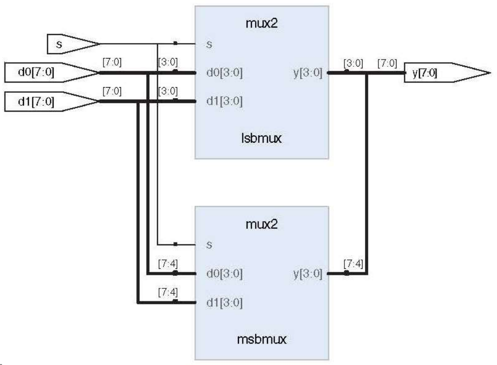
HDL Example 4.16 ACCESSING PARTS OF BUSSES

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;
```



Slide derived from Harris & Harris book

Figure 4.13 mux2_8 **synthesized circuit**

Register

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;

entity flop is
  port(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;
```

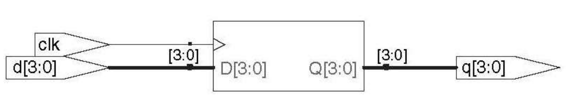


Figure 4.14 `flop` **synthesized circuit**

Resettable Register

HDL Example 4.18 RESETTABLE REGISTER

SystemVerilog

```
module flopr(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 flopr(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

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

entity flopr 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 flopr 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;

entity flopr 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 flopr is
begin
  process(clk) begin
    if rising_edge(clk) then
      if reset then q <= "0000";
      else q <= d;
      end if;
    end if;
  end process;
end;
```

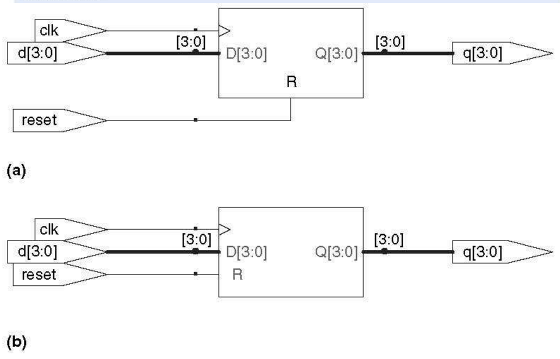


Figure 4.15 `flopr` **synthesized circuit (a) asynchronous reset, (b) synchronous reset**

Resettable Enabled Register

HDL Example 4.19 RESETTABLE ENABLED REGISTER

SystemVerilog

```
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

```
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;
```

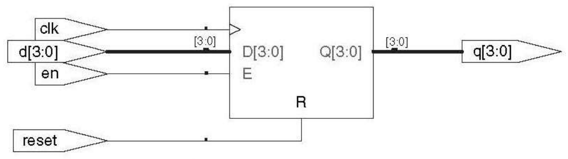


Figure 4.16 `flopenr` **synthesized circuit**

Multiple Registers

HDL Example 4.20 SYNCHRONIZER

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

```
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;
```

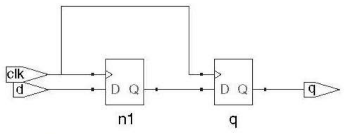


Figure 4.18 `sync` **synthesized circuit**

Always Comb

HDL Example 4.23 FULL ADDER USING `always/process`

SystemVerilog

```
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

```
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;
```

Case Statement

HDL Example 4.24 SEVEN-SEGMENT DISPLAY DECODER

SystemVerilog

```
module sevenseg(input  logic [3:0] data,
                output logic [6:0] segments);
  always_comb
    case(data)
      //                       abc_defg
      0:       segments = 7'b111_1110;
      1:       segments = 7'b011_0000;
      2:       segments = 7'b110_1101;
      3:       segments = 7'b111_1001;
      4:       segments = 7'b011_0011;
      5:       segments = 7'b101_1011;
      6:       segments = 7'b101_1111;
      7:       segments = 7'b111_0000;
      8:       segments = 7'b111_1111;
      9:       segments = 7'b111_0011;
      default: segments = 7'b000_0000;
    endcase
endmodule
```

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
      --                          abcdefg
        when X"0" =>    segments <= "1111110";
        when X"1" =>    segments <= "0110000";
        when X"2" =>    segments <= "1101101";
        when X"3" =>    segments <= "1111001";
        when X"4" =>    segments <= "0110011";
        when X"5" =>    segments <= "1011011";
        when X"6" =>    segments <= "1011111";
        when X"7" =>    segments <= "1110000";
        when X"8" =>    segments <= "1111111";
        when X"9" =>    segments <= "1110011";
        when others =>  segments <= "0000000";
    end case;
  end process;
end;
```

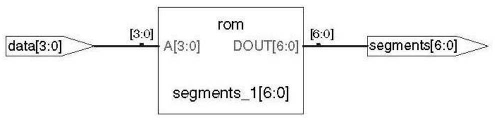


Figure 4.20 `sevenseg` **synthesized circuit**

Case Statement

HDL Example 4.25 3:8 DECODER

SystemVerilog

```
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'b00001000;
      3'b100:  y = 8'b00010000;
      3'b101:  y = 8'b00100000;
      3'b110:  y = 8'b01000000;
      3'b111:  y = 8'b10000000;
      default: y = 8'bxxxxxxxx;
    endcase
endmodule
```

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 <= "00001000";
      when "100" =>  y <= "00010000";
      when "101" =>  y <= "00100000";
      when "110" =>  y <= "01000000";
      when "111" =>  y <= "10000000";
      when others => y <= "XXXXXXXX";
    end case;
  end process;
end;
```

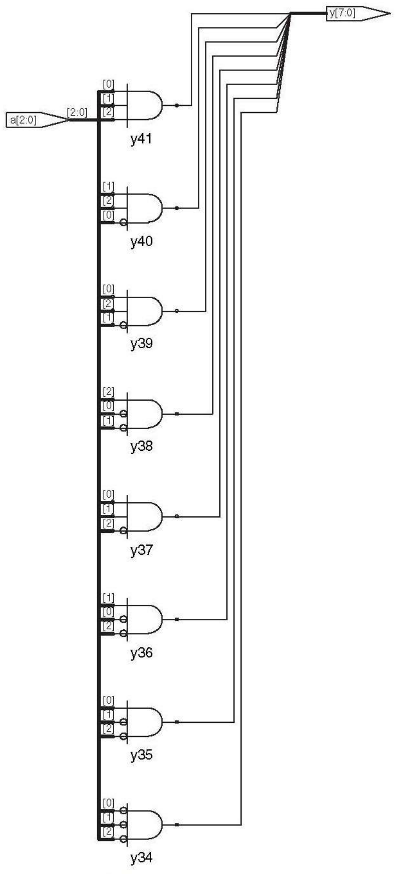


Figure 4.21 `decoder3_8` **synthesized circuit**

Slide derived from Harris & Harris book

More If-Then-Else

HDL Example 4.26 PRIORITY CIRCUIT

SystemVerilog

```
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

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

entity priorityckt is
  port(a: 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    a(3) then y <= "1000";
    elsif a(2) then y <= "0100";
    elsif a(1) then y <= "0010";
    elsif a(0) then y <= "0001";
    else            y <= "0000";
    end if;
  end process;
end;
```

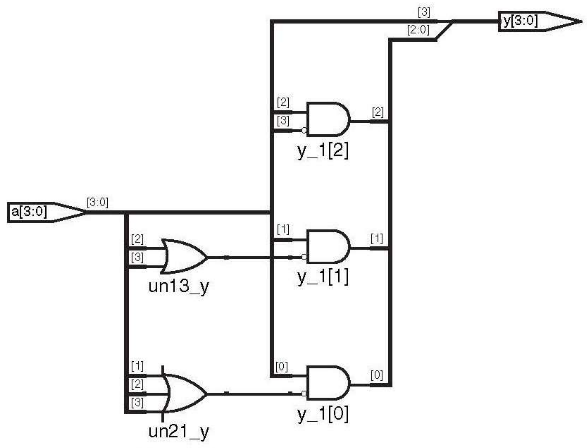


Figure 4.22 `priorityckt` **synthesized circuit**

Slide derived from Harris & Harris book

Casez Statement

HDL Example 4.27 PRIORITY CIRCUIT USING DON'T CARES

SystemVerilog

```
module priority_casez(input  logic [3:0] a,
                      output logic [3:0] y);
  always_comb
    casez(a)
      4'b1???: y <= 4'b1000;
      4'b01??: y <= 4'b0100;
      4'b001?: y <= 4'b0010;
      4'b0001: y <= 4'b0001;
      default: y <= 4'b0000;
    endcase
endmodule
```

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(all) begin
    case? a is
      when "1---" => y <= "1000";
      when "01--" => y <= "0100";
      when "001-" => y <= "0010";
      when "0001" => y <= "0001";
      when others => y <= "0000";
    end case?;
  end process;
end;
```

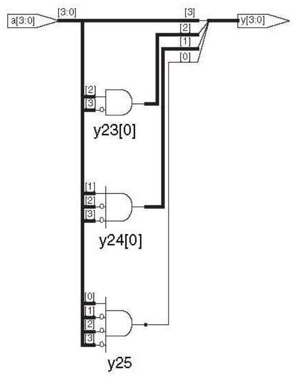


Figure 4.23 `priority_casez` **synthesized circuit**

Blocking vs. Non-Blocking

BLOCKING AND NONBLOCKING ASSIGNMENT GUIDELINES

SystemVerilog

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

```
always_ff @(posedge clk)
  begin
    n1 <= d; // nonblocking
    q <= n1; // nonblocking
  end
```

2. Use continuous assignments to model simple combinational logic.

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

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

```
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.

```
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.

```
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.

```
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;
```

4. Do not make assignments to the same variable in more than one `process` or concurrent assignment statement.

Finite State Machine

HDL Example 4.31 PATTERN RECOGNIZER MOORE FSM

SystemVerilog

```
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
```

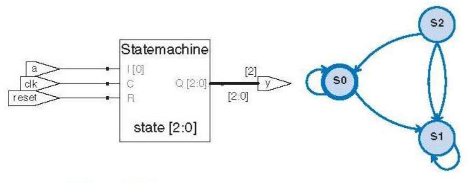


Figure 4.26 patternMoore **synthesized circuit**

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;
    elsif 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;
```

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_8(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
```

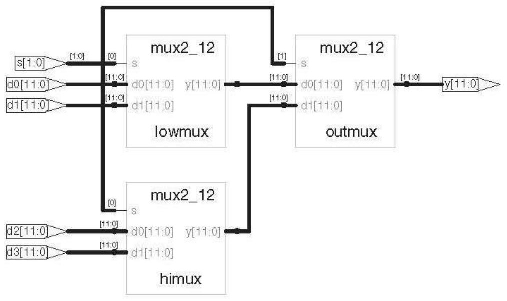


Figure 4.29 `mux4_12` **synthesized circuit**

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;
```