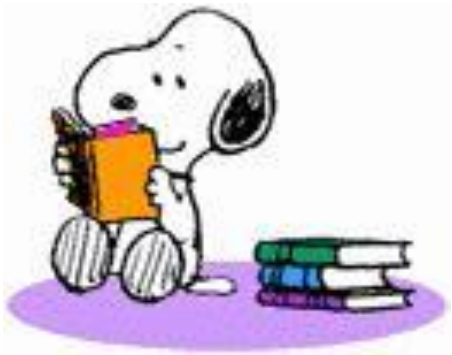
数字逻辑设计与微处理器系统

第五讲

Hardware Description Languages



(硬件描述语言)

1.1 Introduction

- **Hardware description language (HDL):**
  - **specifies logic function only**
  - **Computer-aided design (CAD) tool produces or *synthesizes* the optimized gates**
- **Most commercial designs built using HDLs**
- **Two leading HDLs:**
  - **SystemVerilog/Verilog**
    - **developed in 1984 by Gateway Design Automation**
    - **IEEE standard (1364) in 1995**
    - **Extended in 2005 (IEEE STD 1800-2009)**
  - **VHDL 2008**
    - **Developed in 1981 by the Department of Defense**
    - **IEEE standard (1076) in 1987**
    - **Updated in 2008 (IEEE STD 1076-2008)**

1.1 What & Why HDL?

- **Hardware Description Language(HDL)**
  - **A software programming language used to model the intended operation of a piece of hardware**
- **Why HDL?**
  - **Text-based design rather than Schematic design**
    - **ASIC complexity increase**
    - **faster time-to-market**
  - **Simulation**
  - **Logic Synthesis**
  - **Words are better than pictures**
  - **PLD, CPLD and FPGA became inexpensive and commonplace**

1.1 What is need for HDL ?

- **Model, Represent, And Simulate Digital Hardware**
  - **Hardware Concurrency**
  - **Parallel Activity Flow**
  - **Semantics for Signal Value And Time**
- **Special Constructs And Semantics**
  - **Edge Transitions**
  - **Propagation Delays**
  - **Timing Checks**

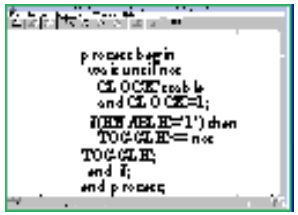
1.1 History of HDL

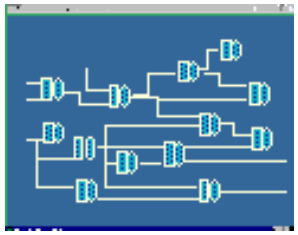
- First HDL
  - PALASM, 1980s--Logic equation
- Then more complex (minimization, if-then-else statement)
  - CUPL (Complier Universal for Programmable Logic)
  - ABEL (Advanced Boolean Equation Language)
- VHDL represents another high level language for digital system design.
  - VHDL = VHSIC HDL
  - VHSIC = Very High Speed Integrated Circuit

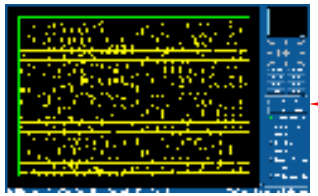
1.1 HDLs

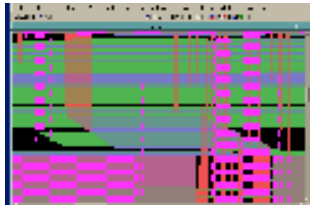
- Software: Assembler →high-level language, like C, C++, Java
- Hardware: block diagram & schematic →HDL
- Efficient →Large, complex, abstraction
- HDL Tool Suits
  - Text editor
  - Complier
  - Synthesizer
  - Simulator
  - Template generator
  - Schematic viewer
  - Translator
  - Timing analyzer
  - Back annotator

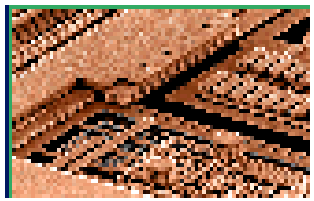
1.1 Design flow

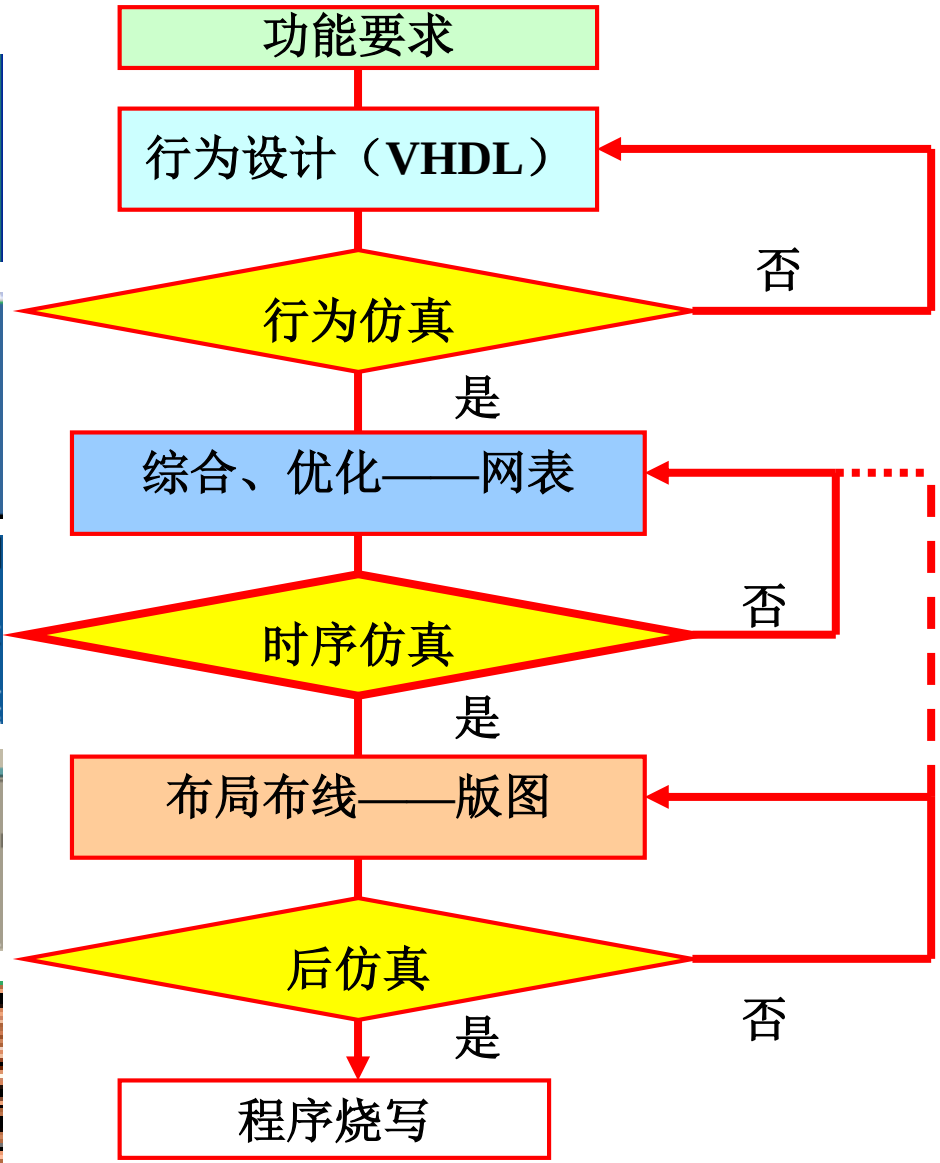










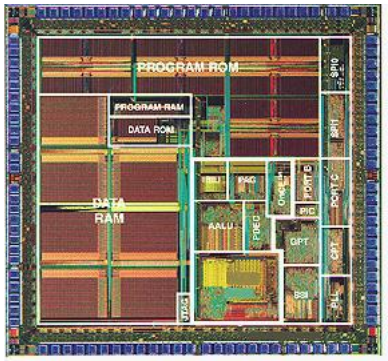


设计创意

+

仿真验证

Levels of Abstraction



RTL SW

System Level

RTL

RT Level (Module)

Gate Level

Circuit Level

Abstraction

1970 1980 1990 2000+

Role of HDLs

- Design Specification
  - unambiguous definition of components, functionality and interfaces
- Design Simulation
  - verify system/subsystem performance and functional correctness prior to design implementation
- Design Synthesis
  - automated generation of a hardware design

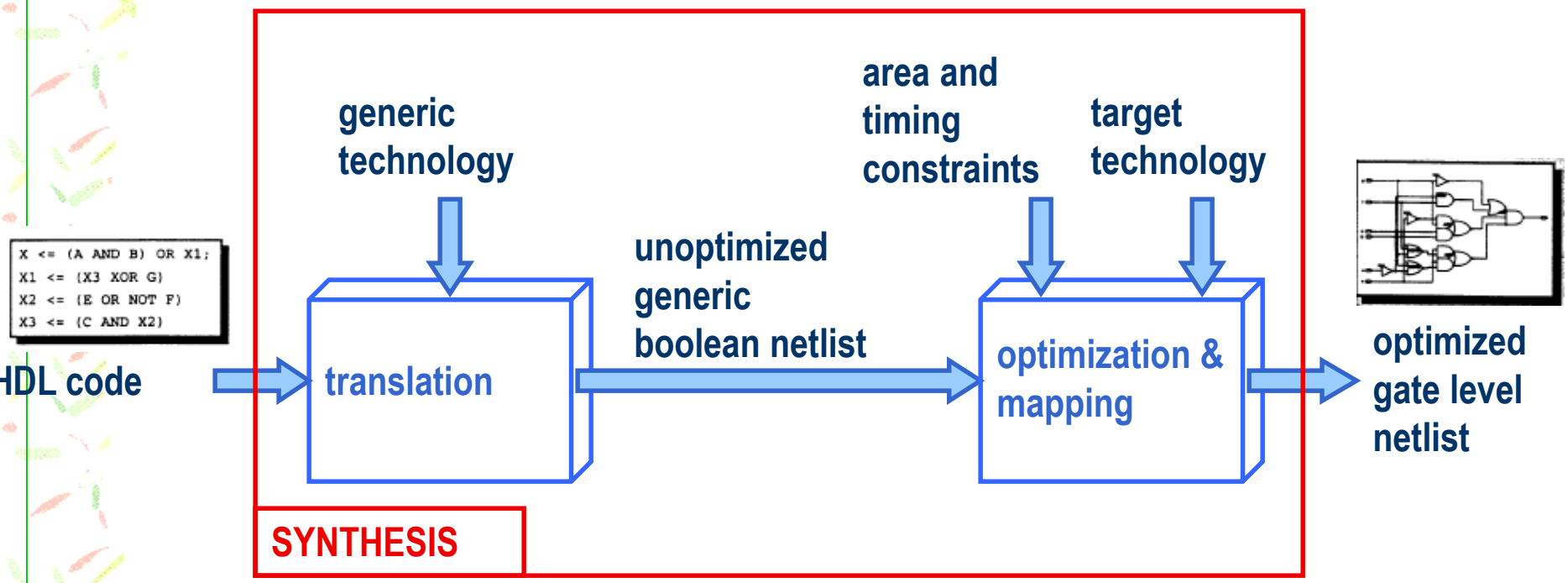
HDL Benefits

- Technology independence
  - portability
  - Reuse
- Interoperability between multiple levels of abstractions
- Cost reduction

- Higher Level of Abstraction (hiding details)
  - The design task become simpler
  - The design is less error prone
  - Productivity is increased

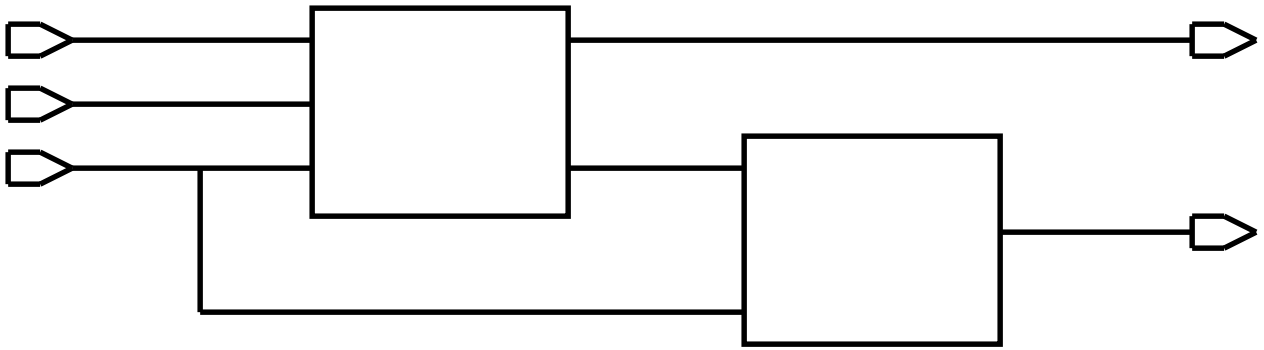
RTL

- model combinational and sequential logic components
- Only a small subset of the Language statements can be mapped in real “Silicon”.



Structural Level

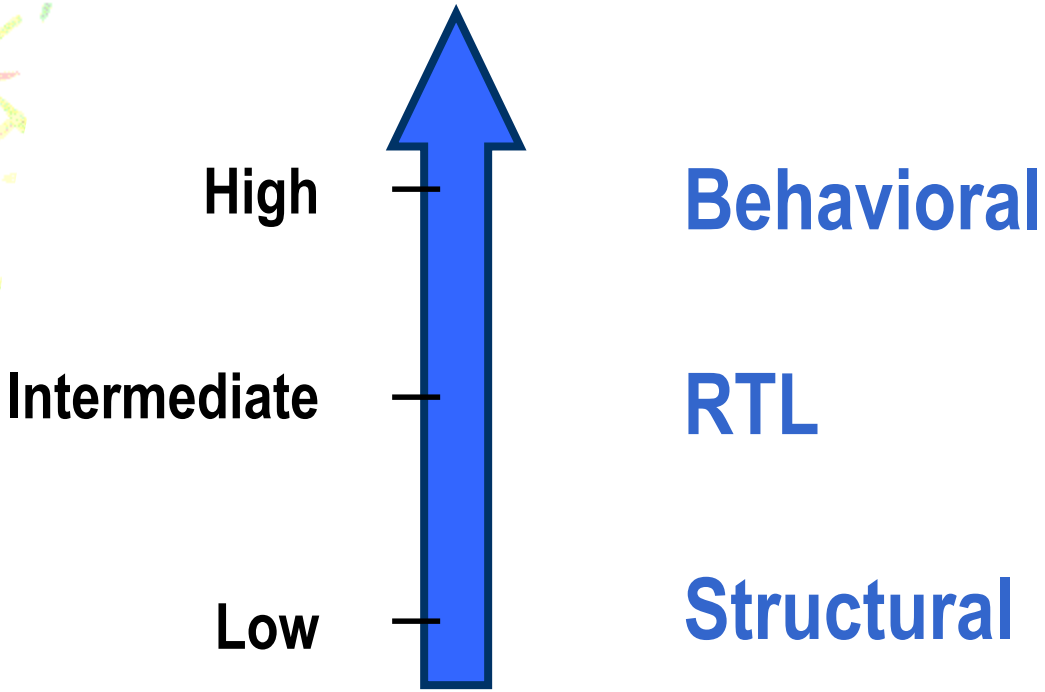
- Describe connectivity among components
- The code consists of a bunch of port mappings.



Behavioral Level

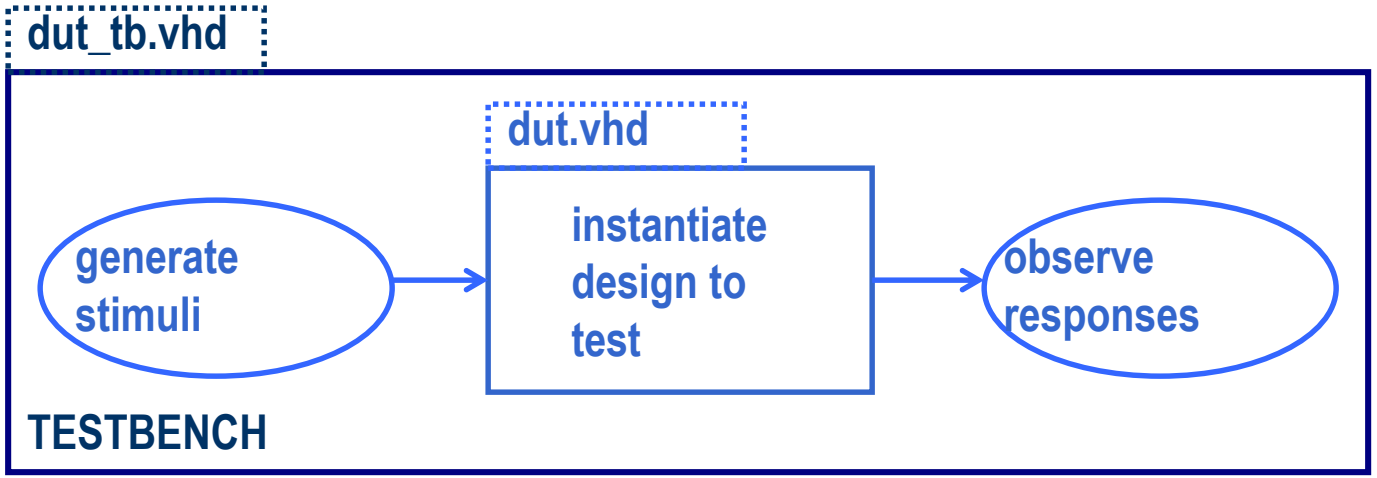
- Describe behavior (functionality and performances)
- All language features can be used

Levels of Abstraction



HDL style vs. application

- High Level Modeling (Behavioral style)
- Design Entry (Structural & RTL styles)
- Simulation (Behavioral style)
  - validation by mean of a test bench



Modeling Digital Systems

- What aspects do we need to consider to describe a digital system ?
  - Interface
  - Function
  - Performance (delay/area/costs/…)

Modeling Digital Systems

- What are the attributes necessary to describe a digital systems ?
  - events, propagation delays, concurrency
  - waveforms and timing
  - signal values
  - shared signals

Modeling Digital Systems

- Hardware description languages must provide constructs for describing the attributes of a specific design, and …
- Simulators use such descriptions for "mimicking" the physical system behavior
- Synthesis compilers use such descriptions for synthesizing manufacturable hardware that conform to a given specification

HDLs vs. Software Languages

Concurrent (parallel) Statements

vs.

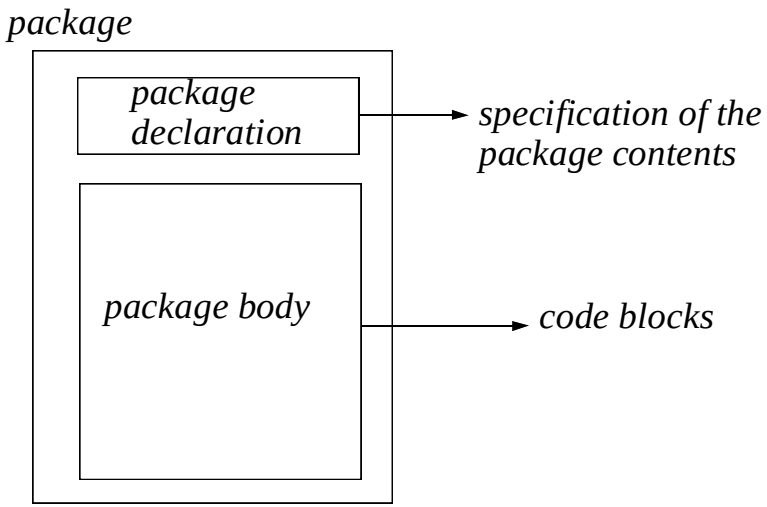
Sequential Statements

VHDL Design Organization

- **Entity**
  the "symbol" (input/output ports)
- **Architecture**
  one of the several possible implementation of the design
- **Configuration**
  binding between the symbol and one of the many possible implementation.
  Can be used to express hierarchy.

VHDL Design Organization

- **Libraries**
  logical units that are mapped to physical directories. The units of a library are called packages.
- **Packages**
  repositories for type definitions, procedures, and functions
- Libraries and packages can be system defined or user defined



Bitwise Operation

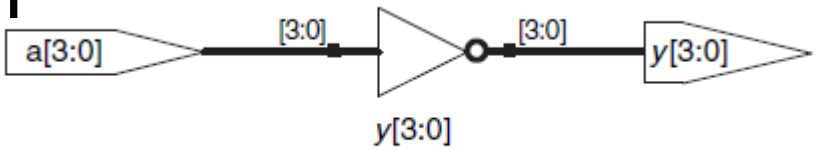
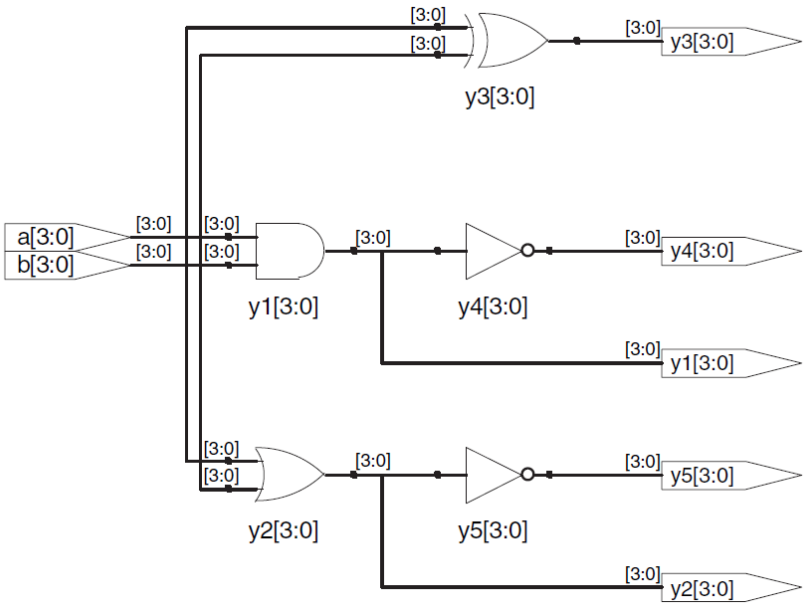


Figure 4.3 inv synthesized circuit

VHDL

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

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

architecture synth of inv is
begin
  y <= not a;
end;
```

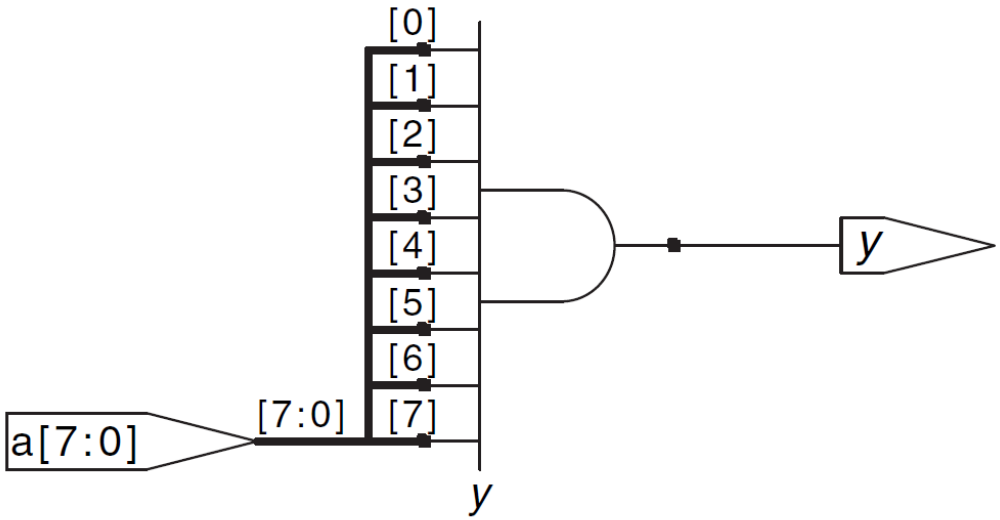
VHDL uses STD_LOGIC_VECTOR to indicate busses of STD_LOGIC. STD_LOGIC_VECTOR(3 downto 0) represents a 4-bit bus.



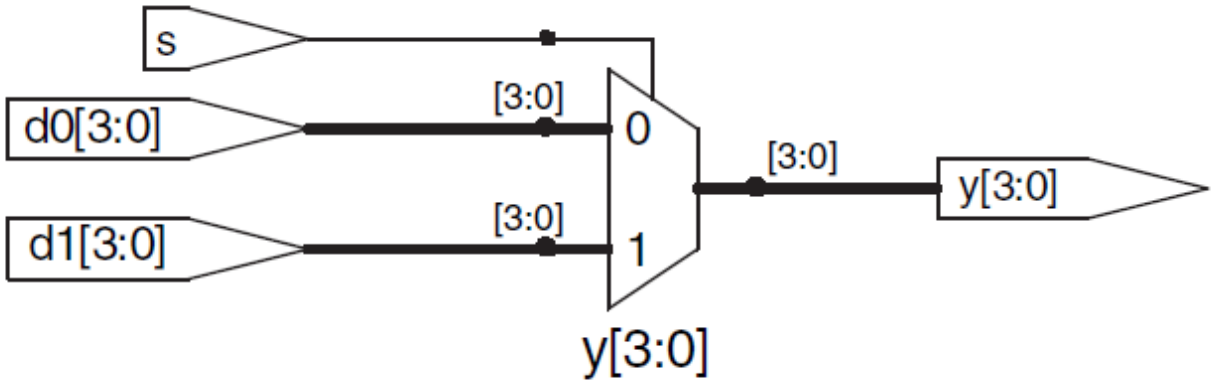

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

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;
```

VHDL

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

entity and8 is
  port(a: in  STD_LOGIC_VECTOR(7 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of and8 is
begin
  y <= and a;
  -- and a is much easier to write than
  -- y <= a(7) and a(6) and a(5) and a(4) and
  --      a(3) and a(2) and a(1) and a(0);
end;
```



Conditional Assignment

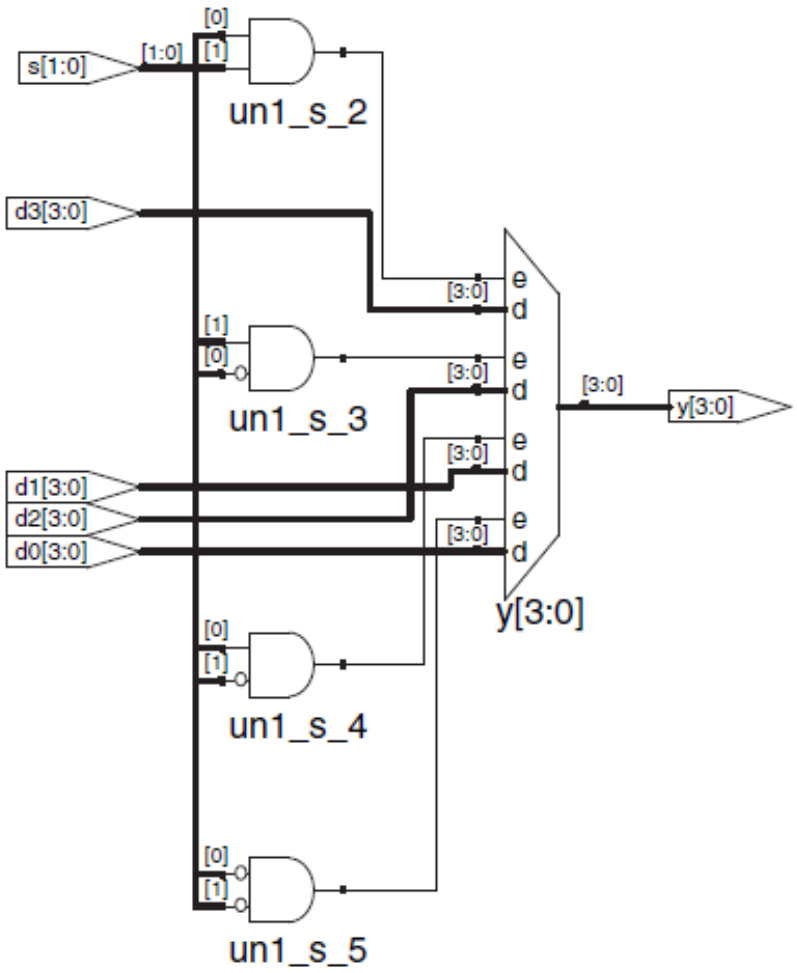



```
library IEEE; use IEEE.STD_LOGIC_1164.all;

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;
```

4:1 Multiplexer

A 4:1 multiplexer can select one of four inputs using multiple `else` clauses in the conditional signal assignment.

```
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 synth1 of mux4 is
begin
  y <= d0 when s = "00" else
       d1 when s = "01" else
       d2 when s = "10" else
       d3;
end;
```



Internal signal : Full Adder

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in} \qquad (4.1)$$

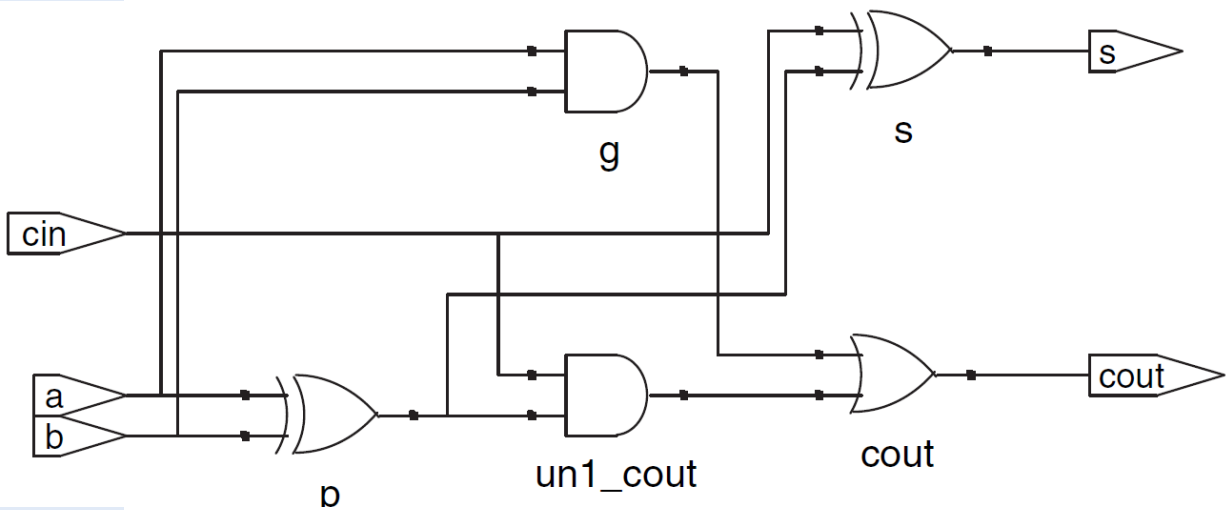
If we define intermediate signals, P and G,

$$P = A \oplus B$$
$$G = AB \qquad (4.2)$$

we can rewrite the full adder as follows:

$$S = P \oplus C_{in}$$
$$C_{out} = G + PC_{in} \qquad (4.3)$$

```
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;
```



Operator

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

Multiplication has precedence over addition in VHDL, as you would expect. However, unlike SystemVerilog, all of the logical operations (`and`, `or`, etc.) have equal precedence, unlike what one might expect in Boolean algebra. Thus, parentheses are necessary; otherwise `cout <= g or p and cin` would be interpreted from left to right as `cout <= (g or p) and cin`.

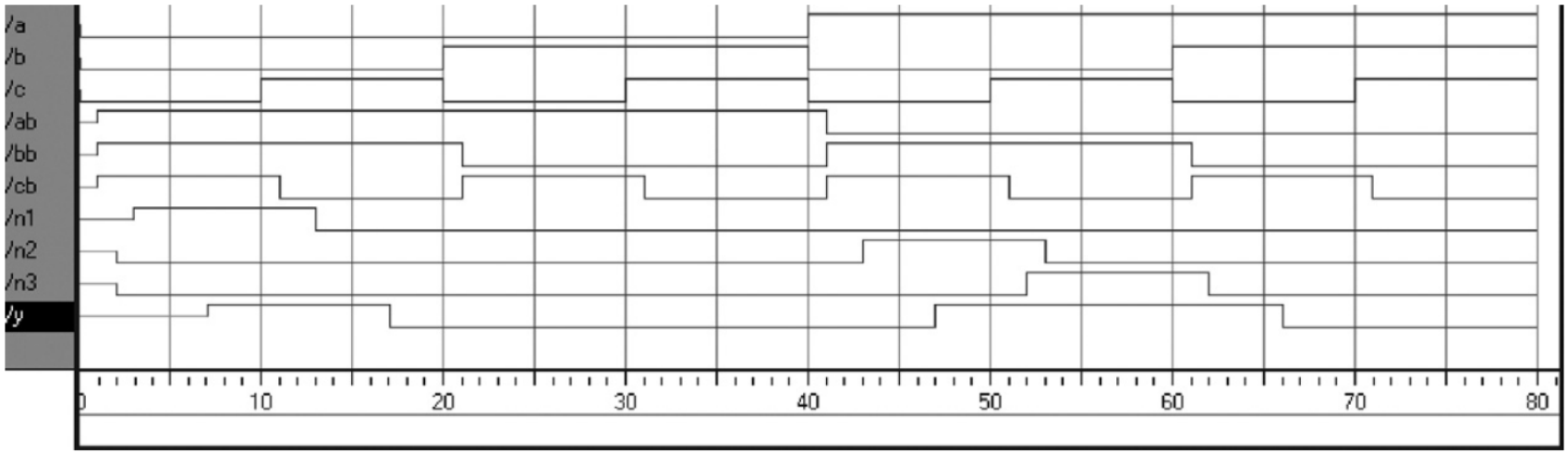
Number

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
60"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

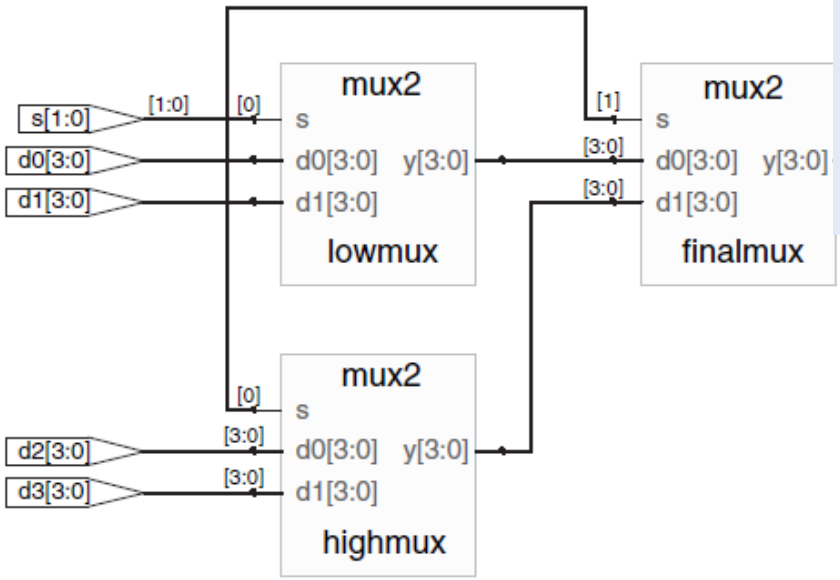
Delay

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

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```



Structural Modeling

```
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;
```



Sequential Logic

```
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;
```

A VHDL `process` is written in the form

```
process(sensitivity list) begin
  statement;
end process;
```

`rising_edge(clk)` is synonymous with `clk'event and clk='1'`.

Resettable Register

```
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;
```

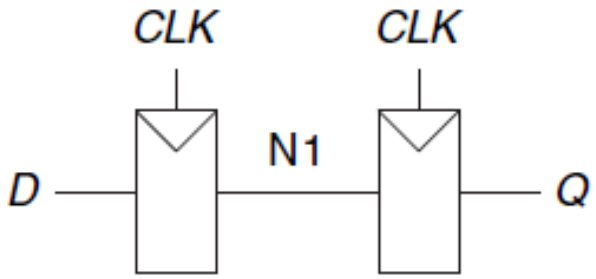
Resettable Register

```
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;
```

Resettable Register

```
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;
```

Multiple Register




```
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;
```

Process

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

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

architecture proc of inv is
begin
  process(all) begin
    y <= not a;
  end process;
end;
```

```
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;
```

Process

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;
```

Process

```
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;
```

Process

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;
```

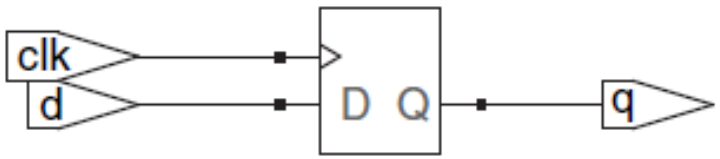
4. Do not make assignments to the same variable in more than one `process` or concurrent assignment statement.

Process

```
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 nonblocking of fulladder is
  signal p, g: STD_LOGIC;
begin
  process(all) begin
    p <= a xor b; -- nonblocking
    g <= a and b; -- nonblocking
    s <= p xor cin;
    cout <= g or (p and cin);
  end process;
end;
```

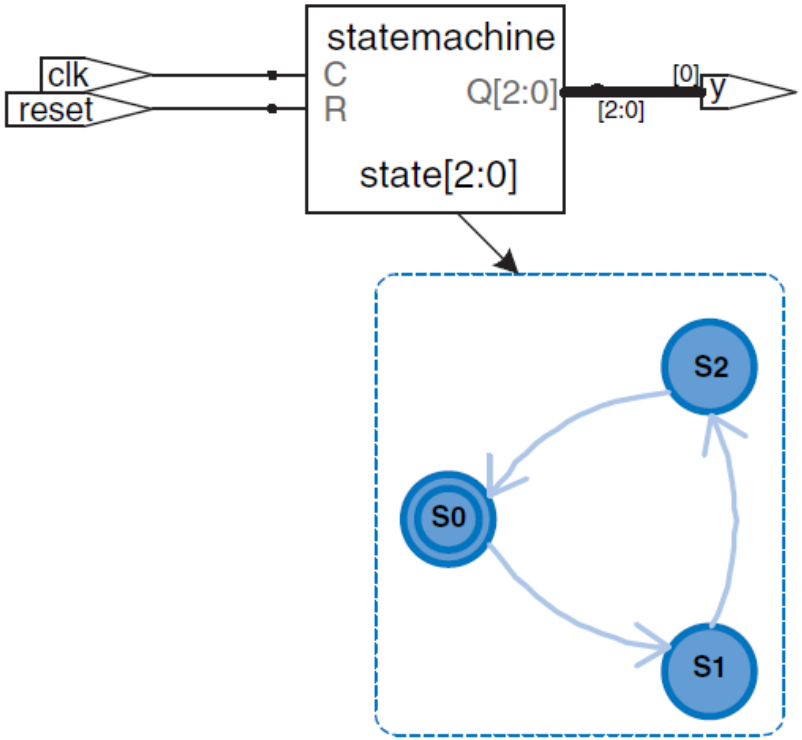
Process

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

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

architecture bad of syncbad is
begin
  process(clk)
    variable n1: STD_LOGIC;
  begin
    if rising_edge(clk) then
      n1 := d; -- blocking
      q <= n1;
    end if;
  end process;
end;
```



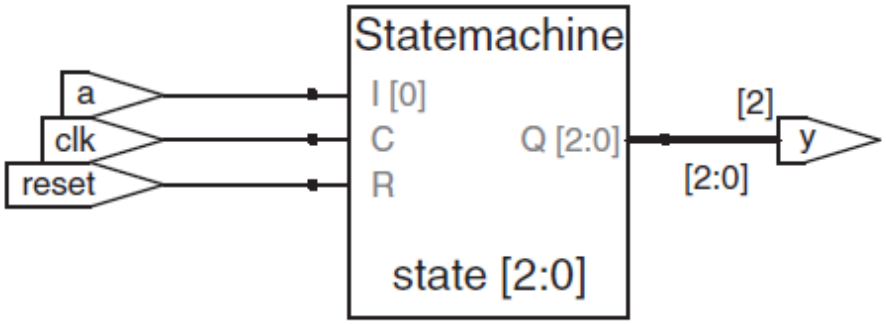
q

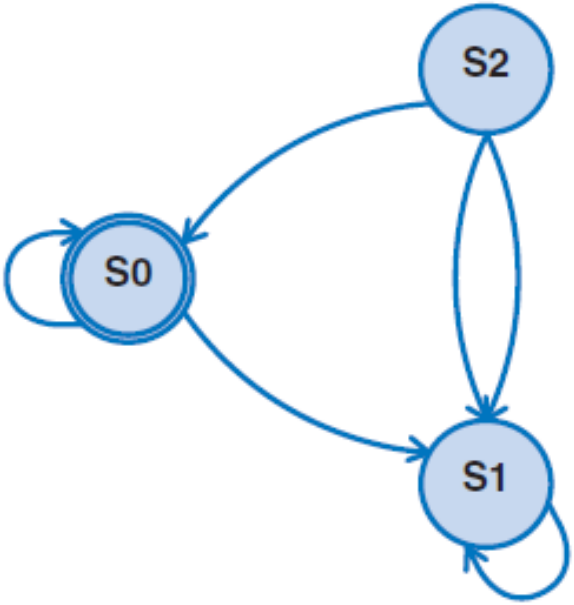
State Machine




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

entity divideby3FSM is
  port(clk, reset: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of divideby3FSM 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
  nextstate <= S1 when state = S0 else
               S2 when state = S1 else
               S0;

  -- output logic
  y <= '1' when state = S0 else '0';
end;
```

State Machine






```
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

```
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;
```

The `generic` statement includes a default value (8) of `width`. The value is an integer.

```
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);
```

Parameterized

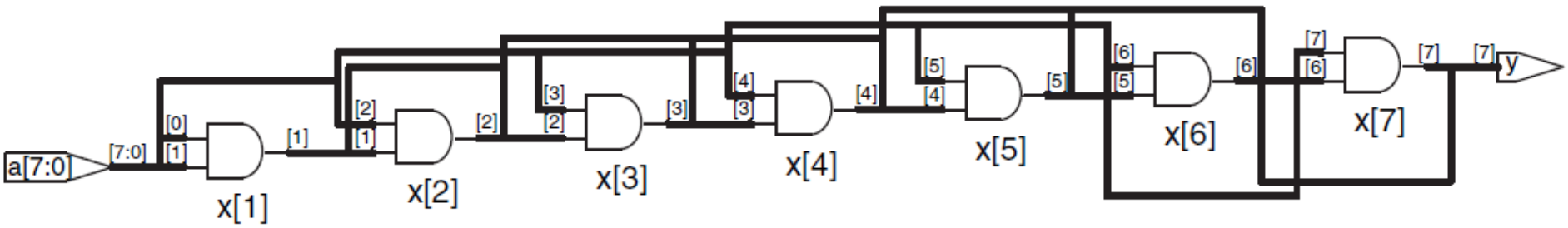
PARAMETERIZED N:2^N DECODER

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

entity decoder is
  generic(N: integer := 3);
  port(a: in  STD_LOGIC_VECTOR(N-1 downto 0);
       y: out STD_LOGIC_VECTOR(2**N-1 downto 0));
end;

architecture synth of decoder is
begin
  process(all)
  begin
    y <= (OTHERS => '0');
    y(TO_INTEGER(a)) <= '1';
  end process;
end;
```

2**N indicates 2^N.

Parameterized

PARAMETERIZED N:2^N DECODER




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

entity andN is
  generic(width: integer := 8);
  port(a: in  STD_LOGIC_VECTOR(width-1 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of andN is
  signal x: STD_LOGIC_VECTOR(width-1 downto 0);
begin
  x(0) <= a(0);
  gen: for i in 1 to width-1 generate
    x(i) <= a(i) and x(i-1);
  end generate;
  y <= x(width-1);
end;
```

Testbenches

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

entity testbench1 is -- no inputs or outputs
end;

architecture sim of testbench1 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- instantiate device under test
  dut: sillyfunction port map(a, b, c, y);

  -- apply inputs one at a time
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    wait; -- wait forever
  end process;
end;
```

Consider testing the `sillyfunction` module from Section 4.1.1 that computes $y = \overline{a}\,\overline{b}\,\overline{c} + a\overline{b}\,\overline{c} + a\overline{b}c$. This is a simple module, so we can perform exhaustive testing by applying all eight possible test vectors.

Testbenches

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

entity testbench2 is -- no inputs or outputs
end;

architecture sim of testbench2 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- instantiate device under test
  dut: sillyfunction port map(a, b, c, y);

  -- apply inputs one at a time
  -- checking results
```

```
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
      assert y='1' report "000 failed.";
    c <= '1';                     wait for 10 ns;
      assert y='0' report "001 failed.";
    b <= '1'; c <= '0';           wait for 10 ns;
      assert y='0' report "010 failed.";
    c <= '1';                     wait for 10 ns;
      assert y='0' report "011 failed.";
    a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
      assert y='1' report "100 failed.";
    c <= '1';                     wait for 10 ns;
      assert y='1' report "101 failed.";
    b <= '1'; c <= '0';           wait for 10 ns;
      assert y='0' report "110 failed.";
    c <= '1';                     wait for 10 ns;
      assert y='0' report "111 failed.";
    wait; -- wait forever
  end process;
end;
```

Testbenches

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_TEXTIO.ALL; use STD.TEXTIO.all;

entity testbench3 is -- no inputs or outputs
end;

architecture sim of testbench3 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y:  STD_LOGIC;
  signal y_expected: STD_LOGIC;
  signal clk, reset: STD_LOGIC;
begin
  -- instantiate device under test
  dut: sillyfunction port map(a, b, c, y);

  -- generate clock
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- at start of test, pulse reset
  process begin
    reset <= '1'; wait for 27 ns; reset <= '0';
    wait;
  end process;
```

```
process is
  file tv: text;
  variable L: line;
  variable vector_in: std_logic_vector(2 downto 0);
  variable dummy: character;
  variable vector_out: std_logic;
  variable vectornum: integer := 0;
  variable errors: integer := 0;
begin
  FILE_OPEN(tv, "example.tv", READ_MODE);
  while not endfile(tv) loop

    -- change vectors on rising edge
    wait until rising_edge(clk);

    -- read the next line of testvectors and split into
    readline(tv, L);
    read(L, vector_in);
    read(L, dummy); -- skip over underscore
```

Testbenches

```
      read(L, vector_out);
      (a, b, c) <= vector_in(2 downto 0) after 1 ns;
      y_expected <= vector_out after 1 ns;

      -- check results on falling edge
      wait until falling_edge(clk);

      if y /= y_expected then
        report "Error: y = " & std_logic'image(y);
        errors := errors + 1;
      end if;

      vectornum := vectornum + 1;
    end loop;

    -- summarize results at end of simulation
    if (errors = 0) then
      report "NO ERRORS -- " &
             integer'image(vectornum) &
             " tests completed successfully."
             severity failure;
    else
      report integer'image(vectornum) &
             " tests completed, errors = " &
             integer'image(errors)
             severity failure;
    end if;
  end process;
end;
```