Chapter 6 VHDL Code Examples

G.1 Introduction

Example VHDL code designs are presented in Chapter 6 to introduce the design and simulation of digital circuits and systems using VHDL.

This appendix presents the code examples along with commenting to support the presented code:

- **Figure 6.6** Eight-bit adder design in VHDL
- **Figure 6.7** General entity declaration
- **Figure 6.8** General architecture body
- **Figure 6.11** Two-input AND gate VHDL entity and architecture
- **Figure 6.12** Two-input AND gate VHDL entity and architecture with optional words removed
- **Figure 6.13** Two-input AND gate VHDL entity and architecture
- **Figure 6.14** Two-input AND gate VHDL entity and architecture with commenting
- **Figure 6.17** Two-to-one multiplexer dataflow description
- **Figure 6.18** Two-to-one multiplexer dataflow description test bench
- **Figure 6.19** Two-to-one multiplexer behavioral description
- **Figure 6.20** Two-to-one multiplexer behavioral description test bench
- **Figure 6.21** Basic logic gate entity and architecture
Figure 6.22 Two-to-one multiplexer structural description
Figure 6.24 Two-to-one multiplexer structural description test bench
Figure 6.26 Inertial and transport delays
Figure 6.27 Inertial and transport delays test bench
Figure 6.29 Example combinational logic circuit
Figure 6.30 Example combinational logic circuit test bench
Figure 6.31 AND gate with internal variable
Figure 6.32 AND gate with internal variable test bench
Figure 6.33 AND gate using generic time delay
Figure 6.34 AND gate using generic time delay test bench
Figure 6.35 Three-input AND gate using generic time delay
Figure 6.36 Structural design using AND gates
Figure 6.37 Structural design test bench
Figure 6.41 VHDL code for a one-bit half-adder
Figure 6.42 VHDL test bench for a one-bit half-adder
Figure 6.44 Four-to-one multiplexer using the If-then-else statement
Figure 6.45 Four-to-one multiplexer test bench
Figure 6.46 Four-to-one multiplexer using the Case-when statement
Figure 6.47 VHDL test bench for the four-to-one multiplexer using the Case-when statement
Figure 6.48 Four-to-one multiplexer using the When-else statement
Figure 6.49 Four-to-one multiplexer using the With-select-when statement
Figure 6.50 Thermometer code to three-bit binary encoder using the Case-when statement
Figure 6.51 Test bench for the thermometer code to three-bit binary encoder
Figure 6.56  *Case-when* statement example

Figure 6.57  *Case-when* statement example test bench

Figure 6.58  *If-then-else* statement example

Figure 6.60  One-bit tristate buffer

Figure 6.61  One-bit tristate buffer test bench

Figure 6.62  Eight-bit tristate buffer using the *If-then-else* statement

Figure 6.63  Eight-bit tristate buffer test bench

Figure 6.64  Eight-bit tristate buffer using the *When-else* statement

Figure 6.68  VHDL code for the D-latch

Figure 6.69  VHDL test bench for the D-latch

Figure 6.70  VHDL code for an eight-bit D-latch array

Figure 6.72  VHDL code for the D-type bistable

Figure 6.74  VHDL code for the D-type bistable register with active low asynchronous reset

Figure 6.76  VHDL code for the four-bit binary counter

Figure 6.77  VHDL test bench for the four-bit binary counter

Figure 6.81  VHDL code for the 1001 sequence detector

Figure 6.82  VHDL test bench for the 1001 sequence detector

Figure 6.85  VHDL code for a UART receiver

Figure 6.86  VHDL test bench for the UART receiver

Figure 6.88  16 × 8 RAM

Figure 6.89  VHDL test bench for the 16 × 8 RAM

Figure 6.91  16 address × 8 data bit ROM

Figure 6.92  16 × 8 ROM test bench

Figure 6.94  Unsigned addition
Figure 6.95  Signed addition
Figure 6.96  Addition test bench
Figure 6.97  Eight-bit unsigned multiplication
Figure 6.98  Eight-bit unsigned multiplication test bench
Figure 6.101 Eight-bit signed multiplication
Figure 6.103 Two-input AND gate test bench
Figure 6.104 Combinational logic circuit description
Figure 6.106 Combinational logic circuit test bench (1)
Figure 6.107 Combinational logic circuit test bench (2)

G.2  VHDL Code Examples

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Design1 IS
    PORT ( A : IN   STD_LOGIC_VECTOR (7 downto 0);
          B : IN   STD_LOGIC_VECTOR (7 downto 0);
          Z : OUT  STD_LOGIC_VECTOR (7 downto 0));
END ENTITY design1;

ARCHITECTURE Dataflow OF Design1 IS
BEGIN
    Z (7 downto 0) <= A (7 downto 0) + B (7 downto 0);
END ARCHITECTURE Dataflow;

Figure 6.6: Eight-bit adder design in VHDL
ENTITY entity_name IS

PORT(
  [SIGNAL] identifier {, identifier}: [mode] signal_type
  { ; [SIGNAL] identifier {, identifier}: [mode] signal_type});
END [ENTITY] [entity_name];

Figure 6.7: General entity declaration

ARCHITECTURE architecture_name OF entity_name IS

  type_declaration
  | signal_declaration
  | constant_declaration
  | component_declaration
  | alias_declaration
  | attribute_specification
  | subprogram_body

BEGIN

  {process_statement
   | concurrent_signal_assignment_statement
   | component_instantiation_statement
   | generate_statement}

END [ARCHITECTURE] [architecture_name];

Figure 6.8: General architecture body

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENTITY And_Gate IS</td>
</tr>
<tr>
<td>2</td>
<td>PORT( A : IN STD_LOGIC;</td>
</tr>
<tr>
<td>3</td>
<td>B : IN STD_LOGIC;</td>
</tr>
<tr>
<td>4</td>
<td>Z : OUT STD_LOGIC);</td>
</tr>
<tr>
<td>5</td>
<td>END ENTITY And_Gate;</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ARCHITECTURE Dataflow OF And_Gate IS</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>BEGIN</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Z &lt;= A AND B;</td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>END ARCHITECTURE Dataflow;</td>
</tr>
</tbody>
</table>

Figure 6.11: Two-input AND gate VHDL entity and architecture
ENTITY And_Gate IS
  PORT(  A : IN   STD_LOGIC;
          B : IN   STD_LOGIC;
          Z : OUT  STD_LOGIC);
END;

ARCHITECTURE Dataflow OF And_Gate IS
BEGIN
Z <= A AND B;
END;

Figure 6.12: Two-input AND gate VHDL entity and architecture with optional words removed

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY And_Gate IS
  PORT(  A : IN   STD_LOGIC;
          B : IN   STD_LOGIC;
          Z : OUT  STD_LOGIC);
END ENTITY And_Gate;

ARCHITECTURE Dataflow OF And_Gate IS
BEGIN
Z <= A AND B;
END ARCHITECTURE Dataflow;

Figure 6.13: Two-input AND gate VHDL entity and architecture
-- 2-input AND gate design
-- Dataflow description

-- Libraries and packages to use
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- Entity declaration
ENTITY And_Gate IS
PORT( A : IN STD_LOGIC;   -- Input A
      B : IN STD_LOGIC;   -- Input B
      Z : OUT STD_LOGIC);  -- Output Z
END ENTITY And_Gate;

-- Architecture body
ARCHITECTURE Dataflow OF And_Gate IS
BEGIN

-- Z becomes A AND B
Z <= A AND B;
END ARCHITECTURE Dataflow;

-- End of File

Figure 6.14: Two-input AND gate VHDL entity and architecture with commenting
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Two_To_One_Mux_DataFlow is
    PORT ( A    : IN   STD_LOGIC;
           B    : IN   STD_LOGIC;
           Sel0 : IN   STD_LOGIC;
           F    : OUT  STD_LOGIC);
END ENTITY Two_To_One_Mux_DataFlow;

ARCHITECTURE DataFlow OF Two_To_One_Mux_DataFlow IS
BEGIN
    F <= ((A AND NOT(Sel0)) OR (B AND Sel0));
END ARCHITECTURE DataFlow;

Figure 6.17: Two-to-one multiplexer dataflow description
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Two_To_One_Mux_DataFlow_vhd IS
END Test_Two_To_One_Mux_DataFlow_vhd;

ARCHITECTURE Behavioural OF Test_Two_To_One_Mux_DataFlow_vhd IS

COMPONENT Two_To_One_Mux_DataFlow
PORT(
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    Sel0 : IN STD_LOGIC;
    F : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A :  STD_LOGIC := '0';
SIGNAL B :  STD_LOGIC := '0';
SIGNAL Sel0 :  STD_LOGIC := '0';
SIGNAL F :  STD_LOGIC;
BEGIN

------------------------------------------------------
-- Instantiate the Unit Under Test (UUT)
------------------------------------------------------
uut: Two_To_One_Mux_DataFlow PORT MAP(
    A => A,
    B => B,
    Sel0 => Sel0,
    F => F);

Test_Bench_Process : PROCESS
BEGIN
    wait for 0 ns;  Sel0 <= '0';  A <= '0';  B <= '0';
    wait for 10 ns;  Sel0 <= '0';  A <= '0';  B <= '1';
    wait for 10 ns;  Sel0 <= '0';  A <= '1';  B <= '0';
    wait for 10 ns;  Sel0 <= '0';  A <= '1';  B <= '1';
    wait for 10 ns;  Sel0 <= '1';  A <= '0';  B <= '0';
    wait for 10 ns;  Sel0 <= '1';  A <= '0';  B <= '1';
    wait for 10 ns;  Sel0 <= '1';  A <= '1';  B <= '0';
    wait for 10 ns;  Sel0 <= '1';  A <= '1';  B <= '1';
    wait for 10 ns;
END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.18: Two-to-one multiplexer dataflow description test bench
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Two_To_One_Mux_Behavioural is
  PORT ( A : IN STD_LOGIC;
         B : IN STD_LOGIC;
         Sel0 : IN STD_LOGIC;
         F : OUT STD_LOGIC);
END ENTITY Two_To_One_Mux_Behavioural;

ARCHITECTURE Behavioural OF Two_To_One_Mux_Behavioural IS
  BEGIN
    Mux_Process: PROCESS(A, B, Sel0)
    BEGIN
      F <= ((A AND NOT(Sel0)) OR (B AND Sel0));
    END PROCESS;
  END ARCHITECTURE Behavioural;

Figure 6.19: Two-to-one multiplexer behavioral description
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Two_To_One_Mux_Behavioural_vhd IS
END Test_Two_To_One_Mux_Behavioural_vhd;

ARCHITECTURE Behavioural OF Test_Two_To_One_Mux_Behavioural_vhd IS

COMPONENT Two_To_One_Mux_Behavioural
PORT(
    A    : IN   STD_LOGIC;
    B    : IN   STD_LOGIC;
    Sel0 : IN   STD_LOGIC;
    F    : OUT  STD_LOGIC);
END COMPONENT;

SIGNAL A    :  STD_LOGIC := '0';
SIGNAL B    :  STD_LOGIC := '0';
SIGNAL Sel0 :  STD_LOGIC := '0';
SIGNAL F    :  STD_LOGIC;

BEGIN
    uut: Two_To_One_Mux_Behavioural PORT MAP(
        A     => A,
        B     => B,
        Sel0  => Sel0,
        F     => F);

    Test_Bench_Process : PROCESS
    BEGIN
        wait for 0 ns; Sel0 <= '0'; A <= '0'; B <= '0';
        wait for 10 ns; Sel0 <= '0'; A <= '0'; B <= '1';
        wait for 10 ns; Sel0 <= '0'; A <= '1'; B <= '0';
        wait for 10 ns; Sel0 <= '0'; A <= '1'; B <= '1';
        wait for 10 ns; Sel0 <= '1'; A <= '0'; B <= '0';
        wait for 10 ns; Sel0 <= '1'; A <= '0'; B <= '1';
        wait for 10 ns; Sel0 <= '1'; A <= '1'; B <= '0';
        wait for 10 ns; Sel0 <= '1'; A <= '1'; B <= '1';
        wait for 10 ns;
    END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.20: Two-to-one multiplexer behavioral description test bench
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY And_Gate is
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Z : OUT  STD_LOGIC);
END ENTITY And_Gate;

ARCHITECTURE Behavioural OF And_Gate IS
BEGIN
    AndGate_Process: PROCESS(A, B)
    BEGIN
        Z <= (A AND B);
    END PROCESS AndGate_Process;
END ARCHITECTURE Behavioural;

2-Input AND gate

Figure 6.21: Basic logic gate entity and architecture
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Inverter is
  PORT ( A    : IN   STD_LOGIC;
        Z    : OUT  STD_LOGIC);
END ENTITY Inverter;

ARCHITECTURE Behavioural OF Inverter IS
  BEGIN
    InverterGate_Process: PROCESS(A)
    BEGIN
      Z <= NOT A;
    END PROCESS InverterGate_Process;
  END ARCHITECTURE Behavioural;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Or_Gate is
  PORT ( A    : IN   STD_LOGIC;
         B    : IN   STD_LOGIC;
         Z    : OUT  STD_LOGIC);
END ENTITY Or_Gate;

ARCHITECTURE Behavioural OF Or_Gate IS
  BEGIN
    OrGate_Process: PROCESS(A, B)
    BEGIN
      Z <= (A OR B);
    END PROCESS OrGate_Process;
  END ARCHITECTURE Behavioural;

2-Input OR gate

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Inverter is
  PORT ( A    : IN   STD_LOGIC;
        Z    : OUT  STD_LOGIC);
END ENTITY Inverter;

ARCHITECTURE Behavioural OF Inverter IS
  BEGIN
    InverterGate_Process: PROCESS(A)
    BEGIN
      Z <= NOT A;
    END PROCESS InverterGate_Process;
  END ARCHITECTURE Behavioural;

Figure 6.21: (Continued)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Two_To_One_Mux_Structural IS
  PORT ( A : IN STD_LOGIC;
          B : IN STD_LOGIC;
          Sel0 : IN STD_LOGIC;
          F : OUT STD_LOGIC);
END ENTITY Two_To_One_Mux_Structural;

ARCHITECTURE Structural OF Two_To_One_Mux_Structural IS

SIGNAL X1 : STD_LOGIC;
SIGNAL X2 : STD_LOGIC;
SIGNAL X3 : STD_LOGIC;

COMPONENT And_Gate
  PORT( A : IN STD_LOGIC;
          B : IN STD_LOGIC;
          Z : OUT STD_LOGIC);
END COMPONENT;

COMPONENT Or_Gate
  PORT( A : IN STD_LOGIC;
          B : IN STD_LOGIC;
          Z : OUT STD_LOGIC);
END COMPONENT;

COMPONENT Inverter
  PORT( A : IN STD_LOGIC;
          Z : OUT STD_LOGIC);
END COMPONENT;

BEGIN
I1: And_Gate
    PORT MAP(A => A, B => X1, Z => X2);

I2: And_Gate
    PORT MAP(A => Sel0, B => B, Z => X3);

I3: Or_Gate
    PORT MAP(A => X2, B => X3, Z => F);

I4: Inverter
    PORT MAP(A => Sel0, Z => X1);
END ARCHITECTURE Structural;

Figure 6.22: Two-to-one multiplexer structural description
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Two_To One_Mux_Structural_vhd IS
END Test_Two_To One_Mux_Structural_vhd;

ARCHITECTURE Behavioural OF Test_Two_To One_Mux_Structural_vhd IS

COMPONENT Two_To One_Mux_Structural
PORT(
    A    : IN   STD_LOGIC;
    B    : IN   STD_LOGIC;
    Sel0 : IN   STD_LOGIC;
    F    : OUT  STD_LOGIC);
END COMPONENT;

SIGNAL A    :  STD_LOGIC := '0';
SIGNAL B    :  STD_LOGIC := '0';
SIGNAL Sel0 :  STD_LOGIC := '0';
SIGNAL F :  STD_LOGIC:
BEGIN

uut: Two_To One_Mux_Structural PORT MAP(
    A => A,
    B => B,
    Sel0 => Sel0,
    F => F);

Test_Bench_Process : PROCESS
BEGIN

    wait for 0 ns;  Sel0 <= '0';  A <= '0';  B <= '0';
    wait for 10 ns;  Sel0 <= '0';  A <= '0';  B <= '1';
    wait for 10 ns;  Sel0 <= '0';  A <= '1';  B <= '0';
    wait for 10 ns;  Sel0 <= '0';  A <= '1';  B <= '1';
    wait for 10 ns;  Sel0 <= '1';  A <= '0';  B <= '0';
    wait for 10 ns;  Sel0 <= '1';  A <= '0';  B <= '1';
    wait for 10 ns;  Sel0 <= '1';  A <= '1';  B <= '0';
    wait for 10 ns;  Sel0 <= '1';  A <= '1';  B <= '1';
    wait for 10 ns;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.24: Two-to-one multiplexer structural description test bench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Example design incorporating "numeric_std" package
USE ieee.numeric_std.ALL;

ENTITY Design1 is
  PORT ( A : IN   STD_LOGIC;
         B : OUT  STD_LOGIC;
         C : OUT  STD_LOGIC;
         D : OUT  STD_LOGIC);
END ENTITY Design1;

ARCHITECTURE  DataFlow OF Design1 IS

BEGIN
  B <= A;
  C <= A AFTER 5 NS;
  D <= TRANSPORT A AFTER 5 NS;

END ARCHITECTURE DataFlow;

Figure 6.26: Inertial and transport delays
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Example incorporating "numeric_std" package
USE ieee.numeric_std.ALL;

ENTITY Test_Design1_vhd IS
END ENTITY Test_Design1_vhd;

ARCHITECTURE Behavioural OF Test_Design1_vhd IS

COMPONENT Design1
PORT(
    A :  IN   STD_LOGIC;
    B :  OUT  STD_LOGIC;
    C :  OUT  STD_LOGIC;
    D :  OUT  STD_LOGIC);
END COMPONENT;

SIGNAL A :  STD_LOGIC := '0';
SIGNAL B :  STD_LOGIC;
SIGNAL C :  STD_LOGIC;
SIGNAL D :  STD_LOGIC;

BEGIN
uut: Design1 PORT MAP(
    A => A,
    B => B,
    C => C,
    D => D);

Test_Bench_Process : PROCESS
BEGIN

    wait for  0 ns;   A <= '0';
    wait for  5 ns;   A <= '1';
    wait for  3 ns;   A <= '0';
    wait for  6 ns;   A <= '1';
    wait for  6 ns;   A <= '0';
    wait for 20 ns;

END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.27: Inertial and transport delays test bench
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Signals_1 IS
    PORT ( A : IN STD_LOGIC;
           B : IN STD_LOGIC;
           C : IN STD_LOGIC;
           Out1 : OUT STD_LOGIC;
           Out2 : OUT STD_LOGIC);
END ENTITY Signals_1;

ARCHITECTURE DataFlow OF Signals_1 IS
    SIGNAL Out1_Internal : STD_LOGIC;
BEGIN
    Out1_Internal <= (A AND B) OR C;
    Out1 <= Out1_Internal;
    Out2 <= NOT(Out1_Internal);
END ARCHITECTURE DataFlow;

Figure 6.29: Example combinational logic circuit
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Signals_1_vhd IS
END ENTITY Test_Signals_1_vhd;

ARCHITECTURE behavioural OF Test_Signals_1_vhd IS

COMPONENT Signals_1
PORT(
    A    : IN std_logic;
    B    : IN std_logic;
    C    : IN std_logic;
    Out1 : OUT std_logic;
    Out2 : OUT std_logic);
END COMPONENT;

SIGNAL A :  STD_LOGIC := '0';
SIGNAL B :  STD_LOGIC := '0';
SIGNAL C :  STD_LOGIC := '0';
SIGNAL Out1 :  STD_LOGIC;
SIGNAL Out2 :  STD_LOGIC;

BEGIN
uut: Signals_1 PORT MAP(
    A    => A,
    B    => B,
    C    => C,
    Out1 => Out1,
    Out2 => Out2);

Test_Bench_Process : PROCESS
BEGIN
    Wait for 0 ns;    A <= '0'; B <= '0'; C <= '0';
    Wait for 10 ns;   A <= '0'; B <= '0'; C <= '1';
    Wait for 10 ns;   A <= '0'; B <= '1'; C <= '0';
    Wait for 10 ns;   A <= '0'; B <= '1'; C <= '1';
    Wait for 10 ns;   A <= '1'; B <= '0'; C <= '0';
    Wait for 10 ns;   A <= '1'; B <= '0'; C <= '1';
    Wait for 10 ns;   A <= '1'; B <= '1'; C <= '0';
    Wait for 10 ns;   A <= '1'; B <= '1'; C <= '1';

    END PROCESS;
END ARCHITECTURE behavioural;

Figure 6.30: Example combinational logic circuit test bench
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY And_Gate_Variables is
    PORT ( A    : IN   STD_LOGIC;
           B    : IN   STD_LOGIC;
           Z    : OUT  STD_LOGIC);
END ENTITY And_Gate_Variables;

ARCHITECTURE Behavioural OF And_Gate_Variables IS
BEGIN
AndGate_Process: PROCESS(A, B)
    VARIABLE Tmp: STD_LOGIC;
BEGIN
    Tmp := (A AND B);
    Z <= Tmp;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.31: AND gate with internal variable
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_And_Gate_Variables_vhd IS
END ENTITY Test_And_Gate_Variables_vhd;

ARCHITECTURE Behavioural OF Test_And_Gate_Variables_vhd IS

COMPONENT And_Gate_Variables
PORT(
    A : IN  STD_LOGIC;
    B : IN  STD_LOGIC;
    Z : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A  :  STD_LOGIC := '0';
SIGNAL B  :  STD_LOGIC := '0';
SIGNAL Z  :  STD_LOGIC;

BEGIN
  uut: And_Gate_Variables PORT MAP(
    A => A,
    B => B,
    Z => Z);

  Test_Bench_Process : PROCESS
  BEGIN
    Wait for 0 ns;   A <= '0';  B <= '0';
    Wait for 10 ns;  A <= '0';  B <= '1';
    Wait for 10 ns;  A <= '1';  B <= '0';
    Wait for 10 ns;  A <= '1';  B <= '1';
    Wait for 10 ns;
  END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.32: AND gate with internal variable test bench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY AND_Gate IS
  GENERIC( delay_time : TIME := 5 ns);
  PORT ( A  :  IN   STD_LOGIC;
         B  :  IN   STD_LOGIC;
         Z1 :  OUT  STD_LOGIC;
         Z2 :  OUT  STD_LOGIC);
END ENTITY AND_Gate;

ARCHITECTURE Behavioural OF AND_gate IS

BEGIN
  Delay1: PROCESS(A, B)
  BEGIN
    Z1 <= A AND B AFTER 5 NS;
  END PROCESS;

  Delay2: PROCESS(A, B)
  BEGIN
    Z2 <= A AND B AFTER delay_time;
  END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.33: AND gate using generic time delay
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_AND_Gate_vhd IS
END Test_AND_Gate_vhd;

ARCHITECTURE Behavioural OF Test_AND_Gate_vhd IS

COMPONENT AND_gate
PORT(
  A  : IN   STD_LOGIC;
  B  : IN   STD_LOGIC;
  Z1 : OUT  STD_LOGIC;
  Z2 : OUT  STD_LOGIC);
END COMPONENT;

SIGNAL A  :  STD_LOGIC := '0';
SIGNAL B  :  STD_LOGIC := '0';

SIGNAL Z1 :  STD_LOGIC;
SIGNAL Z2 :  STD_LOGIC;

BEGIN

uut: AND_gate PORT MAP(
  A  => A,
  B  => B,
  Z1 => Z1,
  Z2 => Z2);

Input_Process : PROCESS
BEGIN

  Wait for 0 ns;   A <= '0';  B <= '0';
  Wait for 10 ns;  A <= '0';  B <= '1';
  Wait for 10 ns;  A <= '1';  B <= '0';
  Wait for 10 ns;  A <= '1';  B <= '1';
  Wait for 10 ns;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.34: AND gate using generic time delay test bench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Three_Input_AndGate IS
  GENERIC( delay_time : TIME := 5 ns);
  PORT (  A  :  IN   STD_LOGIC;
          B  :  IN   STD_LOGIC;
          C  :  IN   STD_LOGIC;
          Z  :  OUT  STD_LOGIC);
END ENTITY Three_Input_AndGate;

ARCHITECTURE Behavioural OF Three_Input_AndGate IS
BEGIN
  Delay_Process: PROCESS(A, B, C)
  BEGIN
    Z <= (A AND B AND C) AFTER delay_time;
  END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.35: Three-input AND gate using generic time delay
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Top_Design IS
Port ( Ain : IN   STD_LOGIC;
      Bin : IN   STD_LOGIC;
      Cin : IN   STD_LOGIC;
      Z1  : OUT  STD_LOGIC;
      Z2  : OUT  STD_LOGIC;
      Z3  : OUT  STD_LOGIC);
END ENTITY Top_Design;

ARCHITECTURE Structural OF Top_Design IS

COMPONENT Three_Input_AndGate
GENERIC(delay_time : TIME);
PORT(
      A  : IN   STD_LOGIC;
      B  : IN   STD_LOGIC;
      C  : IN   STD_LOGIC;
      Z  : OUT  STD_LOGIC);
END COMPONENT;

BEGIN

I1: Three_Input_AndGate
   GENERIC MAP (delay_time => 1 ns)
   PORT MAP(A => Ain, B => Bin, C => Cin, Z => Z1);

I2: Three_Input_AndGate
   GENERIC MAP (delay_time => 5 ns)
   PORT MAP(A => Ain, B => Bin, C => Cin, Z => Z2);

I3: Three_Input_AndGate
   GENERIC MAP (delay_time => 10 ns)
   PORT MAP(A => Ain, B => Bin, C => Cin, Z => Z3);

END ARCHITECTURE Structural;

Figure 6.36: Structural design using AND gates
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Top_Design_vhd IS
END ENTITY Test_Top_Design_vhd;

ARCHITECTURE Behavioural OF Test_Top_Design_vhd IS

COMPONENT Top_Design
PORT(
    Ain : IN STD_LOGIC;
    Bin : IN STD_LOGIC;
    Cin : IN STD_LOGIC;
    Z1  : OUT STD_LOGIC;
    Z2  : OUT STD_LOGIC;
    Z3  : OUT STD_LOGIC);
END COMPONENT;

SIGNAL Ain :  STD_LOGIC := '0';
SIGNAL Bin :  STD_LOGIC := '0';
SIGNAL Cin :  STD_LOGIC := '0';
SIGNAL Z1 :  STD_LOGIC;
SIGNAL Z2 :  STD_LOGIC;
SIGNAL Z3 :  STD_LOGIC;

BEGIN

uut: Top_Design PORT MAP(
    Ain => Ain,
    Bin => Bin,
    Cin => Cin,
    Z1 => Z1,
    Z2 => Z2,
    Z3 => Z3);

Test_Bench_Process : PROCESS
BEGIN

    Wait for 0 ns;  Ain <= '0';  Bin <= '0';  Cin <= '0';
    Wait for 20 ns; Ain <= '0';  Bin <= '0';  Cin <= '1';
    Wait for 20 ns; Ain <= '0';  Bin <= '1';  Cin <= '0';
    Wait for 20 ns; Ain <= '0';  Bin <= '1';  Cin <= '1';
    Wait for 20 ns; Ain <= '1';  Bin <= '0';  Cin <= '0';
    Wait for 20 ns; Ain <= '1';  Bin <= '0';  Cin <= '1';
    Wait for 20 ns; Ain <= '1';  Bin <= '1';  Cin <= '0';
    Wait for 20 ns; Ain <= '1';  Bin <= '1';  Cin <= '1';
    Wait for 20 ns;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.37: Structural design test bench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Half_Adder IS
  PORT ( A : IN STD_LOGIC;
         B : IN STD_LOGIC;
         Sum : OUT STD_LOGIC;
         Cout : OUT STD_LOGIC);
END ENTITY Half_Adder;

ARCHITECTURE Behavioural OF Half_Adder IS
BEGIN
  Sum <= (A XOR B);
  Cout <= (A AND B);
END ARCHITECTURE Behavioural;

Figure 6.41: VHDL code for a one-bit half-adder
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Half_Adder_vhd IS
END Test_Half_Adder_vhd;

ARCHITECTURE behavioural OF Test_Half_Adder_vhd IS

COMPONENT Half_Adder
PORT(
    A     : IN  STD_LOGIC;
    B     : IN  STD_LOGIC;
    Sum   : OUT STD_LOGIC;
    Cout  : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A :  STD_LOGIC:= '0';
SIGNAL B :  STD_LOGIC:= '0';
SIGNAL Sum :  STD_LOGIC;
SIGNAL Cout :  STD_LOGIC;

BEGIN

uut: Half_Adder PORT MAP(
    A    => A,
    B    => B,
    Sum  => Sum,
    Cout => Cout);

Test_Bench_Process : PROCESS
BEGIN
    Wait for 0 ns;   A <= '0';  B <= '0';
    Wait for 10 ns;  A <= '0';  B <= '1';
    Wait for 10 ns;  A <= '1';  B <= '0';
    Wait for 10 ns;  A <= '1';  B <= '1';
    Wait for 10 ns;
END PROCESS;

END ARCHITECTURE behavioural;

Figure 6.42: VHDL test bench for a one-bit half-adder
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Four_To_One_Mux is
PORT ( A   : IN   STD_LOGIC;
       B   : IN   STD_LOGIC;
       C   : IN   STD_LOGIC;
       D   : IN   STD_LOGIC;
       C1  : IN   STD_LOGIC;
       C2  : IN   STD_LOGIC;
       Z   : OUT  STD_LOGIC);
END ENTITY Four_To_One_Mux;

ARCHITECTURE Behavioural OF Four_To_One_Mux IS
BEGIN
PROCESS (A, B, C, D, C1, C2)
BEGIN
If    (C1 = '0' AND C2 = '0') Then
    Z <= A;
ElsIf (C1 = '0' AND C2 = '1') Then
    Z <= B;
ElsIf (C1 = '1' AND C2 = '0') Then
    Z <= C;
Else
    Z <= D;
End If;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.44: Four-to-one multiplexer using the If-then-else statement
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Four_To_One_Mux_vhd IS
END ENTITY Test_Four_To_One_Mux_vhd;

ARCHITECTURE Behavioural OF Test_Four_To_One_Mux_vhd IS

COMPONENT Four_To_One_Mux
PORT(
    A  : IN  STD_LOGIC;
    B  : IN  STD_LOGIC;
    C  : IN  STD_LOGIC;
    D  : IN  STD_LOGIC;
    C1 : IN  STD_LOGIC;
    C2 : IN  STD_LOGIC;
    Z  : OUT STD_LOGIC);
END COMPONENT;

SIGNAL A  :  STD_LOGIC := '0';
SIGNAL B  :  STD_LOGIC := '0';
SIGNAL C  :  STD_LOGIC := '0';
SIGNAL D  :  STD_LOGIC := '0';
SIGNAL C1 :  STD_LOGIC := '0';
SIGNAL C2 :  STD_LOGIC := '0';
SIGNAL Z  :  STD_LOGIC;

BEGIN

uut: Four_To_One_Mux PORT MAP(
    A  =>  A,
    B  =>  B,
    C  =>  C,
    D  =>  D,
    C1 =>  C1,
    C2 =>  C2,
    Z  =>  Z);

Test_Bench_Process : PROCESS
BEGIN
Wait for 0 ns;  A <= '0'; B <= '0'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='1'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '1'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '1'; B <= '0'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '0'; C1 <= '0'; C2 <='1';
Wait for 10 ns; A <= '0'; B <= '0'; C <='1'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '1'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '1'; B <= '0'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '0'; C1 <= '1'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '1'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='1'; D <= '0'; C1 <= '0'; C2 <='1';
Wait for 10 ns; A <= '0'; B <= '1'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '1'; B <= '0'; C <='0'; D <= '0'; C1 <= '0'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '0'; C1 <= '1'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='0'; D <= '1'; C1 <= '1'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '0'; C <='1'; D <= '0'; C1 <= '1'; C2 <='0';
Wait for 10 ns; A <= '0'; B <= '1'; C <='0'; D <= '0'; C1 <= '1'; C2 <='0';
Wait for 10 ns; A <= '1'; B <= '0'; C <='0'; D <= '0'; C1 <= '1'; C2 <='0';

END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.45: Four-to-one multiplexer test bench
Figure 6.46: Four-to-one multiplexer using the *Case-when* statement
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Mux_Case_When_vhd IS
END ENTITY Test_Mux_Case_When_vhd;

ARCHITECTURE Behavioural OF Test_Mux_Case_When_vhd IS

COMPONENT Mux_Case_When
    PORT(
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        C : IN STD_LOGIC;
        D : IN STD_LOGIC;
        Control : IN STD_LOGIC_VECTOR(1 downto 0);
        Z : OUT STD_LOGIC)
    END COMPONENT;

SIGNAL A       :  STD_LOGIC := '0';
SIGNAL B       :  STD_LOGIC := '0';
SIGNAL C       :  STD_LOGIC := '0';
SIGNAL D       :  STD_LOGIC := '0';
SIGNAL Control :  STD_LOGIC_VECTOR(1 downto 0) := (others=>'0');
SIGNAL Z :  STD_LOGIC;

BEGIN
    uut: Mux_Case_When PORT MAP(
        A => A,
        B => B,
        C => C,
        D => D,
        Control => Control,
        Z => Z);

    Test_Bench_Process : PROCESS
    BEGIN
        Wait for 0 ns; A <= '0'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "00";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '1'; D <= '0'; Control(1 downto 0) <= "00";
        Wait for 10 ns; A <= '0'; B <= '1'; C <= '0'; D <= '0'; Control(1 downto 0) <= "00";
        Wait for 10 ns; A <= '1'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "00";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "01";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '1'; D <= '0'; Control(1 downto 0) <= "01";
        Wait for 10 ns; A <= '0'; B <= '1'; C <= '0'; D <= '0'; Control(1 downto 0) <= "01";
        Wait for 10 ns; A <= '1'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "01";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '0'; D <= '1'; Control(1 downto 0) <= "10";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '1'; D <= '0'; Control(1 downto 0) <= "10";
        Wait for 10 ns; A <= '0'; B <= '1'; C <= '0'; D <= '0'; Control(1 downto 0) <= "10";
        Wait for 10 ns; A <= '1'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "10";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '0'; D <= '1'; Control(1 downto 0) <= "11";
        Wait for 10 ns; A <= '0'; B <= '0'; C <= '1'; D <= '0'; Control(1 downto 0) <= "11";
        Wait for 10 ns; A <= '0'; B <= '1'; C <= '0'; D <= '0'; Control(1 downto 0) <= "11";
        Wait for 10 ns; A <= '1'; B <= '0'; C <= '0'; D <= '0'; Control(1 downto 0) <= "11";
    END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.47: HDL test bench for the four-to-one multiplexer using the

Case-when statement

www.newnespress.com
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Mux_When_Else IS
PORT ( A             : IN   STD_LOGIC;
       B             : IN   STD_LOGIC;
       C             : IN   STD_LOGIC;
       D             : IN   STD_LOGIC;
       Control       : IN   STD_LOGIC_VECTOR(1 downto 0);
       Z             : OUT  STD_LOGIC);
END ENTITY Mux_When_Else;

ARCHITECTURE Dataflow OF Mux_When_Else IS
BEGIN
  Z <= A WHEN (Control(1 downto 0) = "00") ELSE
       B WHEN (Control(1 downto 0) = "01") ELSE
       C WHEN (Control(1 downto 0) = "10") ELSE
       D WHEN (Control(1 downto 0) = "11") ELSE
       A;
END ARCHITECTURE Dataflow;

Figure 6.48: Four-to-one multiplexer using the When-else statement
ENTITY Mux_With_Select_When IS
PORT ( A            : IN   STD_LOGIC;
       B            : IN   STD_LOGIC;
       C            : IN   STD_LOGIC;
       D            : IN   STD_LOGIC;
       Control      : IN   STD_LOGIC_VECTOR(1 downto 0);
       Z            : OUT  STD_LOGIC);
END ENTITY Mux_With_Select_When;

ARCHITECTURE Dataflow OF Mux_With_Select_When IS
BEGIN
  WITH Control(1 downto 0) SELECT
  Z <= A    WHEN   "00",
       B    WHEN   "01",
       C    WHEN   "10",
       D    WHEN   "11",
       '-'  WHEN   OTHERS;
END ARCHITECTURE Dataflow;

Figure 6.49: Four-to-one multiplexer using the With-select-when statement
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Thermometer_Case_When IS
PORT ( X : IN STD_LOGIC_VECTOR(6 downto 0);
      d : OUT STD_LOGIC_VECTOR(2 downto 0));
END ENTITY Thermometer_Case_When;

ARCHITECTURE Behavioural OF Thermometer_Case_When IS
BEGIN
  PROCESS(X)
  BEGIN
    CASE (X) IS
      When "0000000"  =>  d(2 downto 0) <= "000";
      When "0000001"  =>  d(2 downto 0) <= "001";
      When "0000011"  =>  d(2 downto 0) <= "010";
      When "0000111"  =>  d(2 downto 0) <= "011";
      When "0001111"  =>  d(2 downto 0) <= "100";
      When "0011111"  =>  d(2 downto 0) <= "101";
      When "0111111"  =>  d(2 downto 0) <= "110";
      When "1111111"  =>  d(2 downto 0) <= "111";
      When OTHERS     =>  d(2 downto 0) <= "000";
    END CASE;
  END PROCESS;
END ARCHITECTURE Behavioural;
```

**Figure 6.50:** Thermometer code to three-bit binary encoder using the *Case-when* statement

[www.newnespress.com](http://www.newnespress.com)
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Thermometer_Case_When_vhd IS
END ENTITY Test_Thermometer_Case_When_vhd;

ARCHITECTURE Behavioural OF Test_Thermometer_Case_When_vhd IS

COMPONENT Thermometer_Case_When
PORT( X : IN STD_LOGIC_VECTOR(6 downto 0);
    d : OUT STD_LOGIC_VECTOR(2 downto 0));
END COMPONENT;

SIGNAL X : STD_LOGIC_VECTOR(6 downto 0) := (others=>'0');
SIGNAL d : STD_LOGIC_VECTOR(2 downto 0);

BEGIN

uut: Thermometer_Case_When PORT MAP(
    X => X,
    d => d);

Test_Bench_Process : PROCESS
BEGIN

Wait for 0 ns; X <= "0000000";
Wait for 10 ns; X <= "0000001";
Wait for 10 ns; X <= "0000011";
Wait for 10 ns; X <= "0000111";
Wait for 10 ns; X <= "0001111";
Wait for 10 ns; X <= "0011111";
Wait for 10 ns; X <= "0111111";
Wait for 10 ns; X <= "1111111";
Wait for 10 ns;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.51: Test bench for the thermometer code to three-bit binary encoder
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Hex_Converter IS
PORT(
    Data_In : IN  STD_LOGIC_VECTOR(3 downto 0);
    a       : OUT STD_LOGIC;
    b       : OUT STD_LOGIC;
    c       : OUT STD_LOGIC;
    d       : OUT STD_LOGIC;
    e       : OUT STD_LOGIC;
    f       : OUT STD_LOGIC;
    g       : OUT STD_LOGIC);
END ENTITY Hex_Converter;

--------------------------------------
-- Process to perform display encoding
PROCESS(Data_In)
BEGIN
CASE Data_In IS
    When "0000" =>   a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '0';
    When "0001" =>   a <= '0'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '0'; g <= '1';
    When "0010" =>   a <= '1'; b <= '1'; c <= '0'; d <= '1'; e <= '1'; f <= '0'; g <= '1';
    When "0011" =>   a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '0'; f <= '0'; g <= '1';
    When "0100" =>   a <= '0'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '1'; g <= '1';
    When "0101" =>   a <= '1'; b <= '0'; c <= '1'; d <= '1'; e <= '0'; f <= '1'; g <= '1';
    When "0110" =>   a <= '1'; b <= '0'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "0111" =>   a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "1000" =>   a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "1001" =>   a <= '1'; b <= '1'; c <= '0'; d <= '0'; e <= '0'; f <= '1'; g <= '1';
    When "1010" =>   a <= '1'; b <= '0'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "1011" =>   a <= '0'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "1100" =>   a <= '1'; b <= '1'; c <= '0'; d <= '0'; e <= '1'; f <= '1'; g <= '1';
    When "1101" =>   a <= '0'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '0'; g <= '1';
    When "1110" =>   a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
    When "1111" =>   a <= '1'; b <= '0'; c <= '0'; d <= '0'; e <= '1'; f <= '1'; g <= '1';
    When OTHERS =>   a <= '0'; b <= '0'; c <= '0'; d <= '0'; e <= '0'; f <= '0'; g <= '0';
END CASE;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.56: Case-when statement example
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Hex_Converter_vhd IS
END ENTITY Test_Hex_Converter_vhd;

ARCHITECTURE Behavioural OF Test_Hex_Converter_vhd IS
COMPONENT Hex_Converter
PORT(
    Data_In : IN  STD_LOGIC_VECTOR(3 downto 0);
    a        : OUT  STD_LOGIC;
    b        : OUT  STD_LOGIC;
    c        : OUT  STD_LOGIC;
    d        : OUT  STD_LOGIC;
    e        : OUT  STD_LOGIC;
    f        : OUT  STD_LOGIC;
    g        : OUT  STD_LOGIC);
END COMPONENT;
SIGNAL Data_In : std_logic_vector(3 downto 0) := (others=>'0');
SIGNAL a : std_logic;
SIGNAL b : std_logic;
SIGNAL c : std_logic;
SIGNAL d : std_logic;
SIGNAL e : std_logic;
SIGNAL f : std_logic;
SIGNAL g : std_logic;
BEGIN
    uut: Hex_Converter PORT MAP(
        Data_In  =>  Data_In,
        a        =>  a,
        b        =>  b,
        c        =>  c,
        d        =>  d,
        e        =>  e,
        f        =>  f,
        g        =>  g);
Test_Bench_Process : PROCESS
BEGIN
    wait for 0 ns;  Data_In <= "0000";
    wait for 10 ns;  Data_In <= "0001";
    wait for 10 ns;  Data_In <= "0010";
    wait for 10 ns;  Data_In <= "0011";
    wait for 10 ns;  Data_In <= "0100";
    wait for 10 ns;  Data_In <= "0101";
    wait for 10 ns;  Data_In <= "0110";
    wait for 10 ns;  Data_In <= "0111";
    wait for 10 ns;  Data_In <= "1000";
    wait for 10 ns;  Data_In <= "1001";
    wait for 10 ns;  Data_In <= "1010";
    wait for 10 ns;  Data_In <= "1011";
    wait for 10 ns;  Data_In <= "1100";
    wait for 10 ns;  Data_In <= "1101";
    wait for 10 ns;  Data_In <= "1110";
    wait for 10 ns;  Data_In <= "1111";
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.57: Case-when statement example test bench
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Hex_Converter IS
PORT(
    Data_In : IN STD_LOGIC_VECTOR(3 downto 0);
    a       : OUT STD_LOGIC;
    b       : OUT STD_LOGIC;
    c       : OUT STD_LOGIC;
    d       : OUT STD_LOGIC;
    e       : OUT STD_LOGIC;
    f       : OUT STD_LOGIC;
    g       : OUT STD_LOGIC);
END ENTITY Hex_Converter;

ARCHITECTURE Behavioural OF Hex_Converter IS
BEGIN
    PROCESS(Data_In)
    BEGIN
        IF (Data_In = "0000") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '0';
        ELSIF (Data_In = "0001") THEN
            a <= '0'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '0'; g <= '0';
        ELSIF (Data_In = "0010") THEN
            a <= '1'; b <= '1'; c <= '0'; d <= '1'; e <= '1'; f <= '0'; g <= '1';
        ELSIF (Data_In = "0011") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '0'; f <= '0'; g <= '1';
        ELSIF (Data_In = "0100") THEN
            a <= '0'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '1'; g <= '1';
        ELSIF (Data_In = "0101") THEN
            a <= '1'; b <= '0'; c <= '1'; d <= '1'; e <= '0'; f <= '1'; g <= '1';
        ELSIF (Data_In = "0110") THEN
            a <= '1'; b <= '0'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
        ELSIF (Data_In = "0111") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '0'; g <= '0';
        ELSIF (Data_In = "1000") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
        ELSIF (Data_In = "1001") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
        ELSIF (Data_In = "1010") THEN
            a <= '1'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '1'; g <= '1';
        ELSIF (Data_In = "1011") THEN
            a <= '0'; b <= '0'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
        ELSIF (Data_In = "1100") THEN
            a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '1'; g <= '0';
        ELSIF (Data_In = "1101") THEN
            a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '0'; g <= '1';
        ELSIF (Data_In = "1110") THEN
            a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
        ELSIF (Data_In = "1111") THEN
            a <= '1'; b <= '0'; c <= '0'; d <= '0'; e <= '1'; f <= '1'; g <= '1';
        ELSE
            a <= '0'; b <= '0'; c <= '0'; d <= '0'; e <= '0'; f <= '0'; g <= '0';
        END IF;
    END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.58: If-then-else statement example

[www.newnespress.com]
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY One_Bit_Buffer is
PORT ( Signal_In  : IN STD_LOGIC;
       Enable     : IN STD_LOGIC;
       Signal_Out : OUT STD_LOGIC);
END ENTITY One_Bit_Buffer;

ARCHITECTURE Behavioural OF One_Bit_Buffer IS
BEGIN
PROCESS (Signal_In, Enable)
BEGIN
If (Enable = '1') Then
   Signal_Out <= Signal_In;
Else
   Signal_Out <= 'Z';
End If;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.60: One-bit tristate buffer
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_One_Bit_Buffer_vhd IS
END ENTITY Test_One_Bit_Buffer_vhd;

ARCHITECTURE Behavioural OF Test_One_Bit_Buffer_vhd IS

COMPONENT One_Bit_Buffer
PORT(
    Signal_In   : IN   STD_LOGIC;
    Enable      : IN   STD_LOGIC;
    Signal_Out  : OUT  STD_LOGIC);
END COMPONENT;

SIGNAL Signal_In :  STD_LOGIC := '0';
SIGNAL Enable    :  STD_LOGIC := '0';
SIGNAL Signal_Out :  STD_LOGIC;

BEGIN

UUT:  One_Bit_Buffer PORT MAP(
    Signal_In   =>  Signal_In,
    Enable      =>  Enable,
    Signal_Out  =>  Signal_Out);

Test bench_Process : PROCESS
BEGIN
    wait for 0 ns;   Signal_In <= '0';  Enable <= '0';
    wait for 10 ns;  Signal_In <= '1';  Enable <= '0';
    wait for 10 ns;  Signal_In <= '0';  Enable <= '1';
    wait for 10 ns;  Signal_In <= '1';  Enable <= '1';
    wait for 10 ns;
END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.61: One-bit tristate buffer test bench
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Eight_Bit_Buffer is
PORT ( Signal_In  : IN   STD_LOGIC_VECTOR(7 downto 0);
      Enable     : IN   STD_LOGIC;
      Signal_Out : OUT  STD_LOGIC_VECTOR(7 downto 0));
END ENTITY Eight_Bit_Buffer;

ARCHITECTURE Behavioural OF Eight_Bit_Buffer IS
BEGIN

PROCESS (Signal_In, Enable)
BEGIN

If (Enable = '1') Then
    Signal_Out(7 downto 0) <= Signal_In(7 downto 0);
Else
    Signal_Out <= "ZZZZZZZZ";
End If;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.62: Eight-bit tristate buffer using the If-then-else statement
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Eight_Bit_Buffer_vhd IS
END ENTITY Test_Eight_Bit_Buffer_vhd;

ARCHITECTURE behavioural OF Test_Eight_Bit_Buffer_vhd IS

COMPONENT Eight_Bit_Buffer
PORT(
    Signal_In   : IN   STD_LOGIC_VECTOR(7 downto 0);
    Enable      : IN   STD_LOGIC;
    Signal_Out  : OUT  STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

SIGNAL Signal_In :  STD_LOGIC_VECTOR(7 downto 0) := "00000000";
SIGNAL Enable    :  STD_LOGIC := '0';
SIGNAL Signal_Out :  STD_LOGIC_VECTOR(7 downto 0);

BEGIN

UUT:  Eight_Bit_Buffer PORT MAP(
    Signal_In(7 downto 0)   =>  Signal_In(7 downto 0),
    Enable                  =>  Enable,
    Signal_Out(7 downto 0)  =>  Signal_Out(7 downto 0));

Test bench_Process : PROCESS
BEGIN

wait for 0 ns;   Signal_In <= "00000000"; Enable <= '0';
wait for 10 ns;  Signal_In <= "11111111"; Enable <= '0';
wait for 10 ns;  Signal_In <= "00000000"; Enable <= '1';
wait for 10 ns;  Signal_In <= "11111111"; Enable <= '1';

END PROCESS;

END ARCHITECTURE behavioural;

Figure 6.63: Eight-bit tristate buffer test bench
ENTITY Eight_Bit_Buffer is
PORT ( Signal_In  : IN   STD_LOGIC_VECTOR(7 downto 0);
       Enable     : IN   STD_LOGIC;
       Signal_Out : OUT  STD_LOGIC_VECTOR(7 downto 0));
END ENTITY Eight_Bit_Buffer;

ARCHITECTURE DataFlow OF Eight_Bit_Buffer IS
BEGIN
    Signal_Out(7 downto 0) <= Signal_In(7 downto 0) When Enable = '1' Else "ZZZZZZZZZZ";
END ARCHITECTURE DataFlow;

Figure 6.64: Eight-bit tristate buffer using the When-else statement
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY D_Latch is
  PORT ( D : IN STD_LOGIC;
         CLK : IN STD_LOGIC;
         Q  : OUT STD_LOGIC);
END ENTITY D_Latch;

ARCHITECTURE Behavioural OF D_Latch IS
BEGIN
  PROCESS(CLK, D)
  BEGIN
    IF (CLK = '1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.68: VHDL code for the D-latch
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_D_Latch_vhd IS
END Test_D_Latch_vhd;

ARCHITECTURE Behavioural OF Test_D_Latch_vhd IS

COMPONENT D_Latch

PORT(
  D : IN  STD_LOGIC;
  CLK : IN  STD_LOGIC;
  Q : OUT STD_LOGIC);
END COMPONENT;

SIGNAL D :  STD_LOGIC := '0';
SIGNAL CLK :  STD_LOGIC := '0';
SIGNAL Q :  STD_LOGIC;
BEGIN

uut: D_Latch PORT MAP(
  D => D,
  CLK => CLK,
  Q => Q);

CLK_Process : PROCESS
BEGIN
  Wait for 0 ns;   CLK <= '0';
  Wait for 20 ns;  CLK <= '1';
  Wait for 20 ns;  CLK <= '0';
END PROCESS;

D_Process : PROCESS
BEGIN
  Wait for 0 ns;   D <= '0';
  Wait for 60 ns;  D <= '1';
  Wait for 22 ns;  D <= '0';
  Wait for 2 ns;   D <= '1';
  Wait for 2 ns;   D <= '0';
  Wait for 16 ns;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.69: VHDL test bench for the D-latch

www.newnespress.com
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY D_Latch_Array is
    PORT ( D : IN STD_LOGIC_VECTOR(7 downto 0);
          CLK : IN STD_LOGIC;
          Q  : OUT STD_LOGIC_VECTOR(7 downto 0));
END ENTITY D_Latch_Array;

ARCHITECTURE Behavioural OF D_Latch_Array IS
BEGIN
    PROCESS(CLK, D)
    BEGIN
        IF (CLK = '1') THEN
            Q(7 downto 0) <= D(7 downto 0);
        END IF;
    END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.70: VHDL code for an eight-bit D-latch array
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY D_Type_Flip_Flop is
  PORT ( D : IN  STD_LOGIC;
         CLK : IN  STD_LOGIC;
         Q  : OUT STD_LOGIC);
END ENTITY D_Type_Flip_Flop;

ARCHITECTURE Behavioural OF D_Type_Flip_Flop IS
BEGIN
  PROCESS(CLK, D)
  BEGIN
    IF (CLK'EVENT AND CLK = '1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.72: VHDL code for the D-type.flip-flop
Figure 6.74: VHDL code for the D-type bistable register with active low asynchronous reset

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY D_Type_Register is
    PORT ( D      : IN   STD_LOGIC_VECTOR(7 downto 0);
           CLK    : IN   STD_LOGIC;
           RESET  : IN   STD_LOGIC;
           Q      : OUT  STD_LOGIC_VECTOR(7 downto 0));
END ENTITY D_Type_Register;

ARCHITECTURE Behavioural OF D_Type_Register IS
BEGIN

PROCESS(CLK, D, RESET)
BEGIN

    IF (RESET = '0') THEN

        Q(7 downto 0) <= "00000000";

    ELSIF (CLK'EVENT AND CLK = '1') THEN

        Q(7 downto 0) <= D(7 downto 0);

    END IF;

END PROCESS;

END ARCHITECTURE Behavioural;
```
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Four_Bit_Counter is
    PORT ( Clock : IN   STD_LOGIC;
            Reset : IN   STD_LOGIC;
            Count : OUT  STD_LOGIC_VECTOR (3 downto 0));
END ENTITY Four_Bit_Counter;

ARCHITECTURE Behavioural of Four_Bit_Counter is

    SIGNAL Count_Int : STD_LOGIC_VECTOR(3 downto 0);

    BEGIN

        PROCESS(Clock, Reset)
        BEGIN

            IF (Reset = '0') THEN
                Count_Int(3 downto 0) <= "0000";
            ELSIF (Clock'Event AND Clock = '1') THEN
                Count_Int(3 downto 0) <= Count_Int(3 downto 0) + 1;
            END IF;

    END PROCESS;

    Count(3 downto 0) <= Count_Int(3 downto 0);

END ARCHITECTURE Behavioural;
```

Figure 6.76: VHDL code for the four-bit binary counter
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Four_Bit_Counter_vhd IS
END Test_Four_Bit_Counter_vhd;

ARCHITECTURE Behavioural OF Test_Four_Bit_Counter_vhd IS

COMPONENT Four_Bit_Counter
PORT(
    Clock : IN  std_logic;
    Reset : IN  std_logic;
    Count : OUT std_logic_vector(3 downto 0));
END COMPONENT;

SIGNAL Clock :  std_logic := '0';
SIGNAL Reset :  std_logic := '0';
SIGNAL Count :  std_logic_vector(3 downto 0);

BEGIN

uut: Four_Bit_Counter PORT MAP(
    Clock => Clock,
    Reset => Reset,
    Count => Count);

Reset_Process: PROCESS
BEGIN
    Wait For 0 ns;    Reset <= '0';
    Wait For 160 ns;  Reset <= '1';
    Wait;
END PROCESS;

Clock_Process: PROCESS
BEGIN
    Wait For 0 ns;   Clock <= '0';
    Wait For 20 ns;  Clock <= '1';
    Wait For 20 ns;  Clock <= '0';
END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.77: VHDL test bench for the four-bit binary counter
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Sequence_Detector is
  PORT ( Data_In  : IN   STD_LOGIC;
         Clock    : IN   STD_LOGIC;
         Reset    : IN   STD_LOGIC;
         Q2       : OUT  STD_LOGIC;
         Q1       : OUT  STD_LOGIC;
         Q0       : OUT  STD_LOGIC;
         Detected : OUT  STD_LOGIC);
END ENTITY Sequence_Detector;

ARCHITECTURE Behavioural OF Sequence_Detector IS
  TYPE State_Type IS (State0, State1, State2, State3, State4);
  SIGNAL Present_State, Next_State : State_Type;
BEGIN
  PROCESS(Clock, Reset)
  BEGIN
    IF (Reset = '0') THEN
      Present_State <= State0;
    ELSIF (Clock'Event AND Clock = '1') THEN
      Present_State <= Next_State;
    END IF;
  END PROCESS;
  PROCESS(Present_State, Data_In)
  BEGIN
    CASE Present_State IS
    WHEN State0 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
    IF (Data_In = '0') THEN
      Next_State <= State0;
    ELSE
      Next_State <= State1;
    END IF;
    WHEN State1 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
    WHEN State2 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
    WHEN State3 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
    WHEN State4 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
    END CASE;
  END PROCESS;
END Behavioral;
WHEN State1 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '1';
    IF (Data_In = '0') THEN
      Next_State <= State2;
    ELSE
      Next_State <= State1;
    END IF;

WHEN State2 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '1';  Q0 <= '0';
    IF (Data_In = '0') THEN
      Next_State <= State3;
    ELSE
      Next_State <= State1;
    END IF;

WHEN State3 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '1';  Q0 <= '1';
    IF (Data_In = '0') THEN
      Next_State <= State0;
    ELSE
      Next_State <= State4;
    END IF;

WHEN State4 =>  Detected <= '1';  Q2 <= '1';  Q1 <= '0';  Q0 <= '0';
    IF (Data_In = '0') THEN
      Next_State <= State0;
    ELSE
      Next_State <= State1;
    END IF;

END CASE;
END PROCESS;
END ARCHITECTURE Behavioural;
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Sequence_Detector_vhd IS
END Test_Sequence_Detector_vhd;

ARCHITECTURE Behavioural OF Test_Sequence_Detector_vhd IS

COMPONENT Sequence_Detector
PORT(
    Data_In   : IN   STD_LOGIC;
    Clock     : IN   STD_LOGIC;
    Reset     : IN   STD_LOGIC;
    Q2        : OUT  STD_LOGIC;
    Q1        : OUT  STD_LOGIC;
    Q0        : OUT  STD_LOGIC;
    Detected  : OUT  STD_LOGIC);
END COMPONENT;

BEGIN

uut: Sequence_Detector PORT MAP(
    Data_In   => Data_In,
    Clock     => Clock,
    Reset     => Reset,
    Q2        => Q2,
    Q1        => Q1,
    Q0        => Q0,
    Detected  => Detected);

Figure 6.82: VHDL test bench for the 1001 sequence detector
Reset_Process : PROCESS
BEGIN
  Wait for 0 ns;  Reset <= '0';
  Wait for 5 ns;  Reset <= '1';
  Wait;
END PROCESS;

Clock_Process : PROCESS
BEGIN
  Wait for 0 ns;  Clock <= '0';
  Wait for 10 ns; Clock <= '1';
  Wait for 10 ns; Clock <= '0';
END PROCESS;

Data_In_Process : PROCESS
BEGIN
  Wait for 0 ns;  Data_In <= '0';
  Wait for 80 ns; Data_In <= '1';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '1';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
  Wait for 80 ns; Data_In <= '1';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
  Wait for 20 ns; Data_In <= '0';
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.82: (Continued)
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Uart_Receiver is
  PORT ( Rx         : IN   STD_LOGIC;
        Clock      : IN   STD_LOGIC;
        Reset      : IN   STD_LOGIC;
        Data_Rx    : OUT  STD_LOGIC_VECTOR(7 downto 0);
        DR         : OUT  STD_LOGIC);
END ENTITY Uart_Receiver;

ARCHITECTURE Behavioural of Uart_Receiver is

SIGNAL Count     : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL Data_Int  : STD_LOGIC_VECTOR(7 downto 0);

BEGIN
  PROCESS (Clock, Reset)
  BEGIN
    IF (Reset='0') then
      Count <= "00000000";
    ELSIF (Clock'Event and Clock = '1') then
      IF (Rx='1' AND (Count = "00000000" or Count = "10101011")) THEN
        Count <= "00000000";
      ELSE
        Count <= Count + 1;
      END IF;
    END IF;
  END PROCESS;

  PROCESS (Clock, Reset, Count)
  BEGIN
    IF (Reset='0') THEN
      Data_Int(7 downto 0) <= "00000000";
      Data_Rx(7 downto 0)  <= "00000000";
      DR <= '0';
    ELSIF (Clock'Event and Clock = '1') THEN
      IF (COUNT = "00000000") THEN
        DR <= '0';
      END IF;
    END IF;
  END PROCESS;

END Architecture of Uart_Receiver;

Figure 6.85: VHDL code for a UART receiver
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>IF (COUNT = &quot;00000001&quot;) THEN</td>
</tr>
<tr>
<td>60</td>
<td>DR &lt;= '0';</td>
</tr>
<tr>
<td>61</td>
<td>END IF;</td>
</tr>
<tr>
<td>62</td>
<td>IF (COUNT = &quot;00010000&quot;) THEN</td>
</tr>
<tr>
<td>63</td>
<td>Data_Int(0) &lt;= Rx;</td>
</tr>
<tr>
<td>64</td>
<td>END IF;</td>
</tr>
<tr>
<td>65</td>
<td>IF (COUNT = &quot;00100000&quot;) THEN</td>
</tr>
<tr>
<td>66</td>
<td>Data_Int(1) &lt;= Rx;</td>
</tr>
<tr>
<td>67</td>
<td>END IF;</td>
</tr>
<tr>
<td>68</td>
<td>IF (COUNT = &quot;00110000&quot;) THEN</td>
</tr>
<tr>
<td>69</td>
<td>Data_Int(2) &lt;= Rx;</td>
</tr>
<tr>
<td>70</td>
<td>END IF;</td>
</tr>
<tr>
<td>71</td>
<td>IF (COUNT = &quot;01001000&quot;) THEN</td>
</tr>
<tr>
<td>72</td>
<td>Data_Int(3) &lt;= Rx;</td>
</tr>
<tr>
<td>73</td>
<td>END IF;</td>
</tr>
<tr>
<td>74</td>
<td>IF (COUNT = &quot;01010000&quot;) THEN</td>
</tr>
<tr>
<td>75</td>
<td>Data_Int(4) &lt;= Rx;</td>
</tr>
<tr>
<td>76</td>
<td>END IF;</td>
</tr>
<tr>
<td>77</td>
<td>IF (COUNT = &quot;01101000&quot;) THEN</td>
</tr>
<tr>
<td>78</td>
<td>Data_Int(5) &lt;= Rx;</td>
</tr>
<tr>
<td>79</td>
<td>END IF;</td>
</tr>
<tr>
<td>80</td>
<td>IF (COUNT = &quot;01110000&quot;) THEN</td>
</tr>
<tr>
<td>81</td>
<td>Data_Int(6) &lt;= Rx;</td>
</tr>
<tr>
<td>82</td>
<td>END IF;</td>
</tr>
<tr>
<td>83</td>
<td>IF (COUNT = &quot;10001000&quot;) THEN</td>
</tr>
<tr>
<td>84</td>
<td>Data_Int(7) &lt;= Rx;</td>
</tr>
<tr>
<td>85</td>
<td>END IF;</td>
</tr>
<tr>
<td>86</td>
<td>IF (COUNT = &quot;10011000&quot;) THEN</td>
</tr>
<tr>
<td>87</td>
<td>Data_Rx(7 downto 0) &lt;= Data_Int(7 downto 0);</td>
</tr>
<tr>
<td>88</td>
<td>END IF;</td>
</tr>
<tr>
<td>89</td>
<td>IF (COUNT = &quot;10101000&quot;) THEN</td>
</tr>
<tr>
<td>90</td>
<td>DR &lt;= '1';</td>
</tr>
<tr>
<td>91</td>
<td>END IF;</td>
</tr>
<tr>
<td>92</td>
<td>IF (COUNT = &quot;10101010&quot;) THEN</td>
</tr>
<tr>
<td>93</td>
<td>DR &lt;= '0';</td>
</tr>
<tr>
<td>94</td>
<td>END IF;</td>
</tr>
<tr>
<td>95</td>
<td>END PROCESS;</td>
</tr>
<tr>
<td>96</td>
<td>END ARCHITECTURE Behavioural;</td>
</tr>
<tr>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.85: (Continued)**
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Uart_Receiver_vhd IS
END Test_Uart_Receiver_vhd;

ARCHITECTURE Behavioural OF Test_Uart_Receiver_vhd IS

COMPONENT Uart_Receiver
PORT(
    Rx      : IN std_logic;
    Clock   : IN std_logic;
    Reset   : IN std_logic;
    Data_Rx : OUT std_logic_vector(7 downto 0);
    DR      : OUT std_logic);
END COMPONENT;

SIGNAL Rx     :  std_logic := '0';
SIGNAL Clock  :  std_logic := '0';
SIGNAL Reset  :  std_logic := '0';
SIGNAL Data_Rx  :  std_logic_vector(7 downto 0);
SIGNAL DR       :  std_logic;

BEGIN

uut: Uart_Receiver PORT MAP(
    Rx       => Rx,
    Clock    => Clock,
    Reset    => Reset,
    Data_Rx  => Data_Rx,
    DR       => DR);

Reset_Process: PROCESS
BEGIN
    Wait for 0 ns;  Reset <= '0';
    Wait for 5 ns;  Reset <= '1';
    Wait;
END PROCESS;

Clock_Process: PROCESS
BEGIN
    Wait for 10 ns;      Clock <= '1';
    Wait for 15 ns;     Clock <= '0';
    Wait;
END PROCESS;

Figure 6.86: VHDL test bench for the UART receiver
BEGIN
    Wait for 0 ns;   Clock <= '0';
    Wait for 10 ns;  Clock <= '1';
    Wait for 10 ns;  Clock <= '0';
END PROCESS;

Rx_Process: PROCESS
BEGIN
    Wait for 0 ns;    Rx <= '1';
    Wait for 100 ns;  Rx <= '0';
    Wait for 320 ns;  Rx <= '1';
    Wait for 5000 ns;
    Wait for 0 ns;    Rx <= '1';
    Wait for 100 ns;  Rx <= '0';
    Wait for 320 ns;  Rx <= '1';
    Wait for 320 ns;  Rx <= '0';
    Wait for 320 ns;  Rx <= '1';
    Wait for 5000 ns;
END PROCESS;
END ARCHITECTURE behavioural;

Figure 6.86: (Continued)
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY RAM_Model is
    PORT ( Address  : IN    Integer range 0 to 15;
          CE       : IN    STD_LOGIC;
          WE       : IN    STD_LOGIC;
          OE       : IN    STD_LOGIC;
          Data     : INOUT STD_LOGIC_VECTOR(7 downto 0));
END ENTITY RAM_Model;

ARCHITECTURE Behavioural OF RAM_Model IS
BEGIN
    PROCESS(Address, CE, WE, OE) IS
        TYPE Ram_Array IS ARRAY (0 to 15) OF STD_LOGIC_VECTOR(7 downto 0);
        VARIABLE Mem: Ram_Array;
    BEGIN
        Data(7 downto 0) <= (others => 'Z');
        IF (CE = '0') THEN
            IF (WE = '0') THEN
                Mem(Address) := Data(7 downto 0);
            ELSIF (OE = '0') THEN
                Data(7 downto 0) <= Mem(Address);
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.88: 16 × 8 RAM
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_RAM_Model_vhd IS
END Test_RAM_Model_vhd;

ARCHITECTURE behavioural OF Test_RAM_Model_vhd IS

COMPONENT RAM_Model
PORT(
    Address : IN     Integer range 0 to 15;
    CE       : IN     STD_LOGIC;
    WE       : IN     STD_LOGIC;
    OE       : IN     STD_LOGIC;
    Data     : INOUT  STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

SIGNAL CE      :  STD_LOGIC := '0';
SIGNAL WE      :  STD_LOGIC := '0';
SIGNAL OE      :  STD_LOGIC := '0';
SIGNAL Address :  Integer range 0 to 15;
SIGNAL Data :  STD_LOGIC_VECTOR(7 downto 0);
BEGIN
uut: RAM_Model PORT MAP(
    Address  =>  Address,
    CE       =>  CE,
    WE       =>  WE,
    OE       =>  OE,
    Data     =>  Data);

Test_Bench_Process : PROCESS
BEGIN
wait for 0 ns;   Address <= 0;  Data <= "ZZZZZZZZZ";
    CE <= '1';    WE <= '1';   OE <= '1';
wait for 10 ns;  Address <= 0;  Data <= "10000001";
    CE <= '0';    WE <= '1';   OE <= '1';
wait for 10 ns;  CE <= '0';    WE <= '0';   OE <= '1';
wait for 10 ns;  CE <= '1';    WE <= '1';   OE <= '1';   Data <= "ZZZZZZZZZ";
wait for 10 ns;  Address <= 0;  Data <= "ZZZZZZZZZ";
wait for 10 ns;  CE <= '0';    WE <= '1';   OE <= '1';
wait for 10 ns;  CE <= '0';    WE <= '0';   OE <= '0';
wait for 10 ns;  CE <= '1';    WE <= '1';   OE <= '1';
    wait for 10 ns;
END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.89: VHDL test bench for the $16 \times 8$ RAM
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM is
  Port ( Address : IN INTEGER Range 0 to 15;
        Data : OUT STD_LOGIC_VECTOR(7 downto 0));
END ENTITY ROM;

ARCHITECTURE Behavioural of ROM is

TYPE Rom_Array IS Array (0 to 15) of STD_LOGIC_VECTOR(7 downto 0);

CONSTANT ROM: Rom_Array := (
  "11000000",
  "00010011",
  "00100000",
  "00110000",
  "01000000",
  "01010000",
  "01100000",
  "01110000",
  "10000000",
  "10010000",
  "10100000",
  "10110000",
  "11000000",
  "11010000",
  "11100011",
  "11111111");

BEGIN
  Data <= Rom(Address);
END ARCHITECTURE Behavioural;

Figure 6.91: 16 address × 8 data bit ROM
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_ROM IS
END ENTITY Test_ROM;

ARCHITECTURE Behavioural OF Test_ROM IS

COMPONENT ROM
PORT(  Address :  IN   INTEGER Range 0 to 15;
       Data    :  OUT  STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

SIGNAL Address :  INTEGER range 0 to 15;
SIGNAL Data    :  STD_LOGIC_VECTOR (7 downto 0);

BEGIN
uut: ROM PORT MAP(
     Address => Address,
     Data    => Data);

Test_Stimulus : PROCESS
BEGIN
    wait for 0 ns;   Address <= 0;
    wait for 10 ns;  Address <= 1;
    wait for 10 ns;  Address <= 2;
    wait for 10 ns;  Address <= 3;
    wait for 10 ns;  Address <= 4;
    wait for 10 ns;  Address <= 5;
    wait for 10 ns;  Address <= 6;
    wait for 10 ns;  Address <= 7;
    wait for 10 ns;  Address <= 8;
    wait for 10 ns;  Address <= 9;
    wait for 10 ns;  Address <= 10;
    wait for 10 ns;  Address <= 11;
    wait for 10 ns;  Address <= 12;
    wait for 10 ns;  Address <= 13;
    wait for 10 ns;  Address <= 14;
    wait for 10 ns;  Address <= 15;
    wait for 10 ns;
END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.92: 16 × 8 ROM test bench
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Adder1 IS
  PORT ( A : IN   STD_LOGIC_VECTOR (7 downto 0);
         B : IN   STD_LOGIC_VECTOR (7 downto 0);
         P : OUT  STD_LOGIC_VECTOR (8 downto 0));
END ENTITY Adder1;

ARCHITECTURE DataFlow of Adder1 is

  SIGNAL A_Int : STD_LOGIC_VECTOR(8 downto 0);
  SIGNAL B_Int : STD_LOGIC_VECTOR(8 downto 0);
  BEGIN
    A_Int(8) <= '0';
    A_Int(7 downto 0) <= A(7 downto 0);
    B_Int(8) <= '0';
    B_Int(7 downto 0) <= B(7 downto 0);
    P(8 downto 0) <= A_Int(8 downto 0) + B_Int(8 downto 0);
  END ARCHITECTURE DataFlow;

Figure 6.94: Unsigned addition
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
-- USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Adder2 IS
    PORT ( A : IN STD_LOGIC_VECTOR (7 downto 0);
          B : IN STD_LOGIC_VECTOR (7 downto 0);
          P : OUT STD_LOGIC_VECTOR (8 downto 0));
END ENTITY Adder2;

ARCHITECTURE DataFlow of Adder2 is

SIGNAL Signed_A : SIGNED(8 downto 0);
SIGNAL Signed_B : SIGNED(8 downto 0);
SIGNAL Signed_P : SIGNED(8 downto 0);

BEGIN

    Signed_A(8) <= Signed_A(7);
    Signed_A(7 downto 0) <= Signed(A(7 downto 0));
    Signed_B(8) <= Signed_B(7);
    Signed_B(7 downto 0) <= Signed(B(7 downto 0));

    Signed_P(8 downto 0) <= Signed_A(8 downto 0) + Signed_B(8 downto 0)
    P <= STD_LOGIC_VECTOR(Signed_P(8 downto 0));

END ARCHITECTURE DataFlow;

Figure 6.95: Signed addition
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test_Adder1_vhd IS
END Test_Adder1_vhd;

ARCHITECTURE Behavioural OF Test_Adder1_vhd IS

COMPONENT Adder1
PORT(
    A : IN std_logic_vector(7 downto 0);
    B : IN std_logic_vector(7 downto 0);
    P : OUT std_logic_vector(7 downto 0));
END COMPONENT;

SIGNAL A :  std_logic_vector(7 downto 0) := (others=>'0');
SIGNAL B :  std_logic_vector(7 downto 0) := (others=>'0');
SIGNAL P :  std_logic_vector(7 downto 0);

BEGIN

uut: Adder1 PORT MAP(
    A => A,
    B => B,
    P => P);

Test_Bench_Process : PROCESS
BEGIN

WAIT for 0 ns;   A <= "00000000";
WAIT for 10 ns;  A <= "00000001";
WAIT for 10 ns;  A <= "11111101";
WAIT for 10 ns;  A <= "11111111";
WAIT for 10 ns;  A <= "11111101";
WAIT for 10 ns;  A <= "11111111";

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.96: Addition test bench
<table>
<thead>
<tr>
<th></th>
<th>LIBRARY IEEE;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>USE IEEE.STD_LOGIC_1164.ALL;</td>
</tr>
<tr>
<td>3</td>
<td>USE IEEE.STD_LOGIC_ARITH.ALL;</td>
</tr>
<tr>
<td>4</td>
<td>USE IEEE.STD_LOGIC_UNSIGNED.ALL;</td>
</tr>
<tr>
<td>5</td>
<td>END ENTITY Unsigned_Multiplier;</td>
</tr>
<tr>
<td>6</td>
<td>END ARCHITECTURE DataFlow;</td>
</tr>
</tbody>
</table>

**Figure 6.97: Eight-bit unsigned multiplication**

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Unsigned_Multiplier is
  PORT ( A : IN  STD_LOGIC_VECTOR (7 downto 0);
         B : IN  STD_LOGIC_VECTOR (7 downto 0);
         P : OUT STD_LOGIC_VECTOR (15 downto 0));
END ENTITY Unsigned_Multiplier;

ARCHITECTURE DataFlow of Unsigned_Multiplier is
BEGIN
  P <= A * B;
END ARCHITECTURE DataFlow;
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Test_Unsigned_Multiplier_vhd IS
END ENTITY Test_Unsigned_Multiplier_vhd;

ARCHITECTURE Behavioural OF Test_Unsigned_Multiplier_vhd IS

COMPONENT Unsigned_Multiplier
PORT(
    A : IN  STD_LOGIC_VECTOR(7 downto 0);
    B : IN  STD_LOGIC_VECTOR(7 downto 0);
    P : OUT STD_LOGIC_VECTOR(15 downto 0));
END COMPONENT;

SIGNAL A :  STD_LOGIC_VECTOR(7 downto 0) := (others=>'0');
SIGNAL B :  STD_LOGIC_VECTOR(7 downto 0) := (others=>'0');
SIGNAL P :  STD_LOGIC_VECTOR(15 downto 0);

BEGIN

uut: Unsigned_Multiplier PORT MAP(
    A => A,
    B => B,
    P => P);

Test_Bench_Process : PROCESS
BEGIN

    wait for  0 ns;  A <= "00000000";  B <= "00000000";
    wait for 10 ns;  A <= "00000001";  B <= "00000001";
    wait for 10 ns;  A <= "10000000";  B <= "10000000";
    wait for 10 ns;  A <= "00000010";  B <= "00000010";
    wait for 10 ns;  A <= "11111111";  B <= "00000001";
    wait for 10 ns;  A <= "11111111";  B <= "11111111";
    wait for 10 ns;

END PROCESS;

END ARCHITECTURE Behavioural;

Figure 6.98: Eight-bit unsigned multiplication test bench
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
-- USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Signed_Multiplier is
  PORT ( A : IN  STD_LOGIC_VECTOR (7 downto 0);
        B : IN  STD_LOGIC_VECTOR (7 downto 0);
        P : OUT STD_LOGIC_VECTOR (15 downto 0));
END ENTITY Signed_Multiplier;

ARCHITECTURE DataFlow of Signed_Multiplier is

  SIGNAL  A_Signed: SIGNED(7 downto 0);
  SIGNAL  B_Signed: SIGNED(7 downto 0);
  SIGNAL  P_Signed: SIGNED(15 downto 0);

BEGIN

  A_Signed(7 downto 0)  <=  SIGNED(A(7 downto 0));
  B_Signed(7 downto 0)  <=  SIGNED(B(7 downto 0));
  P_Signed(15 downto 0) <= A_Signed(7 downto 0) * B_Signed(7 downto 0);
  P(15 downto 0) <= STD_LOGIC_VECTOR(P_Signed(15 downto 0));

END ARCHITECTURE DataFlow;

Figure 6.101: Eight-bit signed multiplication
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY test_And_Gate_vhd IS
END ENTITY test_And_Gate_vhd;

ARCHITECTURE Behavioural OF test_And_Gate_vhd IS

COMPONENT And_Gate
  PORT(
         A : IN std_logic;
         B : IN std_logic;
         Z : OUT std_logic
      );
END COMPONENT;

SIGNAL A :  std_logic := '0';
SIGNAL B :  std_logic := '0';
SIGNAL Z :  std_logic;

BEGIN
  uut: And_Gate PORT MAP(
    A => A,
    B => B,
    Z => Z);

  Input_Process : PROCESS
  BEGIN
    WAIT FOR 0 ns;   A <= '0'; B <= '0';
    WAIT FOR 10 ns;  A <= '1'; B <= '0';
    WAIT FOR 10 ns;  A <= '0'; B <= '1';
    WAIT FOR 10 ns;  A <= '1'; B <= '1';
    WAIT FOR 10 ns;
  END PROCESS;
END ARCHITECTURE Behavioural;

Figure 6.103: Two-input AND gate test bench
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

-- Top Entity

ENTITY Circuit1 IS
    PORT ( A : IN   STD_LOGIC;
            B : IN   STD_LOGIC;
            C : IN   STD_LOGIC;
            Z : OUT  STD_LOGIC);
END ENTITY Circuit1;

-- Top Architecture

ARCHITECTURE Behavioural OF Circuit1 IS
BEGIN
    Z <= (A AND B) OR NOT((NOT(A OR B)) AND (A OR C));
END ARCHITECTURE Behavioural;

-- End of File

Figure 6.104: Combinational logic circuit description
1 | -- Test bench for Circuit1
2 |-------------------------------------------------------------
3 |-------------------------------------------------------------
4 |-------------------------------------------------------------
5 |-------------------------------------------------------------
6 | -- Libraries and packages to use
7 |-------------------------------------------------------------
8 | LIBRARY ieee;
9 | USE ieee.std_logic_1164.ALL;
10 | USE ieee.std_logic_unsigned.ALL;
11 | USE ieee.numeric_std.ALL;
12 | USE std.textio.ALL;
13 |-------------------------------------------------------------
14 |-------------------------------------------------------------
15 | -- Test bench Entity
16 |-------------------------------------------------------------
17 | ENTITY Test_Circuit1_vhd IS
18 | END Test_Circuit1_vhd;
19 |-------------------------------------------------------------
20 | ARCHITECTURE Behavioural OF Test_Circuit1_vhd IS
21 |-------------------------------------------------------------
22 |-------------------------------------------------------------
23 |-------------------------------------------------------------
24 |COMPONENT Circuit1
25 |PORT( A : IN STD_LOGIC;
26 | B : IN STD_LOGIC;
27 | C : IN STD_LOGIC;
28 | Z : OUT STD_LOGIC);
29 |END COMPONENT;
30 |-------------------------------------------------------------
31 |-- Inputs
32 |-------------------------------------------------------------
33 |-------------------------------------------------------------

Figure 6.106: Combinational logic circuit test bench (1)
SIGNAL A :  STD_LOGIC := '0';
SIGNAL B :  STD_LOGIC := '0';
SIGNAL C :  STD_LOGIC := '0';

-- Outputs

SIGNAL Z :  STD_LOGIC;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: Circuit1 PORT MAP(
    A  =>  A,
    B  =>  B,
    C  =>  C,
    Z  =>  Z);

-- Read from stimulus file and apply to circuit process

Process_1 : PROCESS

FILE      Stimulus_File     : TEXT;
FILE      Results_File      : TEXT;

VARIABLE  Input_Pattern     :  LINE;
VARIABLE  Results_Pattern   :  LINE;
VARIABLE  Read_OK           :  BOOLEAN;
VARIABLE  Char              :  CHARACTER;
VARIABLE  A_In, B_In, C_In  :  STD_LOGIC;

Figure 6.106: (Continued)
BEGIN
-- Open files for READ and WRITE

FILE_OPEN(Stimulus_File, "C:\Circuit1_Stimulus.txt", READ_MODE);
FILE_OPEN(Results_File, "C:\Circuit1_Results.txt", WRITE_MODE);

-- Write header text to results file

WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "ABC Z");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);

-- Loop read from file and apply
-- stimulus until end of file

WHILE (NOT ENDFILE(Stimulus_File)) LOOP

-- Read line from 'Stimulus_File' into
-- variable 'Input_Pattern'

READLINE(Stimulus_File, Input_Pattern);

-- Read first character from
-- 'Input_Pattern'

READ(Input_Pattern, CHAR, Read_OK);

Figure 6.106: (Continued)
\begin{verbatim}
124  -- If line is not good or the first
125  -- character is not a TAB, then
126  -- skip remainder of loop is not good
127  --------------------------------------
128
129  IF((NOT Read_OK) OR (CHAR /= HT)) THEN NEXT;
130  END IF;
131
132  --------------------------------------
133  -- Read first stimulus bit
134  -- Read second stimulus bit
135  -- Read third stimulus bit
136  --------------------------------------
137
138  READ(Input_Pattern, A_In);
139  READ(Input_Pattern, CHAR);
140  READ(Input_Pattern, B_In);
141  READ(Input_Pattern, CHAR);
142  READ(Input_Pattern, C_In);
143
144  A  <=  A_In;
145  B  <=  B_In;
146  C  <=  C_In;
147
148  --------------------------------------
149  -- Wait for time before applying next
150  -- test stimulus
151  --------------------------------------
152
153  WAIT FOR 10 ns;
154
155  --------------------------------------
156  -- Write stimulus and output to output
157  -- file
158  --------------------------------------
159
160
\end{verbatim}

Figure 6.106: (Continued)
WRITE(Results_Pattern, A);
WRITE(Results_Pattern, B);
WRITE(Results_Pattern, C);
WRITE(Results_Pattern, " ");
WRITE(Results_Pattern, Z)
WRITELINE(Results_File, Results_Pattern);
-- End of Loop
END LOOP;
-- Write footer text to results file
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-- Test completed");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);
-- Close the OPENed files
FILE_CLOSE(Stimulus_File);
FILE_CLOSE(Results_File);
-- Stop process or it will repeat if
-- simulation time longer than a
-- single pass of the input and will
-- overwrite results file
-----------------------------
Figure 6.106: (Continued)
| 199 | WAIT;                      |
| 200 | END PROCESS;              |
| 201 |                           |
| 202 | END ARCHITECTURE Behavioural; |
| 203 |                           |
| 204 | -- End of File            |
| 205 |                           |

Figure 6.106: (Continued)
-- Test bench for Circuit1

-- Libraries and packages to use

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE std.textio.ALL;

-- Test bench Entity

ENTITY Test_Circuit1_vhd IS
END Test_Circuit1_vhd;

ARCHITECTURE Behavioural OF Test_Circuit1_vhd IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT Circuit1
PORT(
    A : IN STD_LOGIC;
    B : IN STD_LOGIC;
    C : IN STD_LOGIC;
    Z : OUT STD_LOGIC);
END COMPONENT;

Figure 6.107: Combinational logic circuit test bench (2)
SIGNAL A :  STD_LOGIC := '0';
SIGNAL B :  STD_LOGIC := '0';
SIGNAL C :  STD_LOGIC := '0';

-- Outputs
SIGNAL Z :  STD_LOGIC;

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: Circuit1 PORT MAP(
     A  =>  A,
     B  =>  B,
     C  =>  C,
     Z  =>  Z);

-- Read from stimulus file and apply to circuit process

Process_1 : PROCESS
FILE  Stimulus_File  : TEXT;
FILE  Results_File   : TEXT;

VARIABLE Input_Pattern   :  LINE;
VARIABLE Results_Pattern :  LINE;
VARIABLE Read_OK         :  BOOLEAN;
VARIABLE Char            :  CHARACTER;

VARIABLE A_In, B_In, C_In  :  BIT;
Variable Z_Out             :  BIT;

--------------------------------------
BEGIN
--------------------------------------
-- Open files for READ and WRITE
--------------------------------------
FILE_OPEN(Stimulus_File, "C:\Circuit1_Stimulus.txt", READ_MODE);
FILE_OPEN(Results_File,"C:\Circuit1_Results.txt", WRITE_MODE);

--------------------------------------
-- Write header text to results file
--------------------------------------
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "ABC Z");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);

Figure 6.107: (Continued)
-- Loop read from file and apply stimulus until end of file

WHILE (NOT ENDFILE(Stimulus_File)) LOOP

-- Read line from 'Stimulus_File' into variable 'Input_Pattern'
READLINE(Stimulus_File, Input_Pattern);

-- Read first character from 'Input_Pattern'
READ(Input_Pattern, CHAR, Read_OK);

-- If line is not good or the first character is not a TAB, then skip remainder of loop is not good
IF((NOT Read_OK) OR (CHAR /=HT)) THEN NEXT;
END IF;

-- Read first stimulus bit
-- Read second stimulus bit
-- Read third stimulus bit

Figure 6.107: (Continued)
READ(Input_Pattern, A_In);
READ(Input_Pattern, CHAR);
READ(Input_Pattern, B_In);
READ(Input_Pattern, CHAR);
READ(Input_Pattern, C_In);

-- Apply test stimulus
-- Initially convert inputs (A, B, C)
-- as BIT to STD_LOGIC - only consider
-- logic '0' or logic '1'

IF (A_In = '1') THEN A <= '1'; ELSE A <= '0'; END IF;
IF (B_In = '1') THEN B <= '1'; ELSE B <= '0'; END IF;
IF (C_In = '1') THEN C <= '1'; ELSE C <= '0'; END IF;

-- Wait for time before applying next test stimulus
WAIT FOR 10 ns;

-- Convert 'Z' STD_LOGIC to 'Z_Out'
-- BIT - only consider logic '0' and
-- logic '1' and unknown 'X'

Figure 6.107: (Continued)
IF (Z = '1') THEN  Z_Out := '1';
ELSE  Z_Out := '0';
END IF;

-- Write stimulus and output to output file
WRITE(Results_Pattern, A_In);
WRITE(Results_Pattern, B_In);
WRITE(Results_Pattern, C_In);
WRITE(Results_Pattern, " ");
WRITE(Results_Pattern, Z_Out);
WRITELINE(Results_File, Results_Pattern);

END LOOP;

-- Write footer text to results file
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-- Test completed");
WRITELINE(Results_File, Results_Pattern);
WRITE(Results_Pattern, "-----------------------------");
WRITELINE(Results_File, Results_Pattern);

-- Close the OPENed files

Figure 6.107: (Continued)
Figure 6.107: (Continued)