APPENDIX G

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 $\times$ 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

```
 1  ENTITY And_Gate IS
 2      PORT(  A : IN   STD_LOGIC;
 3             B : IN   STD_LOGIC;
 4             Z : OUT  STD_LOGIC);
 5  END ENTITY And_Gate;
 6
 7  ARCHITECTURE Dataflow OF And_Gate IS
 8
 9  BEGIN
10
11        Z <= A AND B;
12
13  END ARCHITECTURE Dataflow;
```

Figure 6.11: Two-input AND gate VHDL entity and architecture

```
1   ENTITY And_Gate IS
2       PORT(  A : IN   STD_LOGIC;
3              B : IN   STD_LOGIC;
4              Z : OUT  STD_LOGIC);
5   END;
6
7   ARCHITECTURE Dataflow OF And_Gate IS
8
9   BEGIN
10
11         Z <= A AND B;
12
13  END;
```

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

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3
4   ENTITY And_Gate IS
5       PORT(  A : IN   STD_LOGIC;
6              B : IN   STD_LOGIC;
7              Z : OUT  STD_LOGIC);
8   END ENTITY And_Gate;
9
10  ARCHITECTURE Dataflow OF And_Gate IS
11
12  BEGIN
13
14         Z <= A AND B;
15
16  END ARCHITECTURE Dataflow;
```

Figure 6.13: Two-input AND gate VHDL entity and architecture

```
1  ----------------------------------------------------
2  -- 2-input AND gate design
3  -- Dataflow description
4  ----------------------------------------------------
5
6  ----------------------------------------------------
7  -- Libraries and packages to use
8  ----------------------------------------------------
9
10 LIBRARY IEEE;
11 USE IEEE.STD_LOGIC_1164.ALL;
12
13 ----------------------------------------------------
14 -- Entity declaration
15 ----------------------------------------------------
16
17 ENTITY And_Gate IS
18     PORT(  A : IN   STD_LOGIC;   -- Input A
19            B : IN   STD_LOGIC;   -- Input B
20            Z : OUT  STD_LOGIC);  -- Output Z
21 END ENTITY And_Gate;
22
23 ----------------------------------------------------
24 -- Architecture body
25 ----------------------------------------------------
26
27 ARCHITECTURE Dataflow OF And_Gate IS
28
29 BEGIN
30
31 ----------------------------------------
32 -- Z becomes A AND B
33 ----------------------------------------
34
35       Z <= A AND B;
36
37 END ARCHITECTURE Dataflow;
38
39 ----------------------------------------------------
40 -- End of File
41 ----------------------------------------------------
```

Figure 6.14: Two-input AND gate VHDL entity and architecture with commenting

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  USE ieee.std_logic_arith.all;
 4  USE ieee.std_logic_unsigned.all;
 5
 6  ENTITY Two_To_One_Mux_DataFlow is
 7      PORT ( A    : IN   STD_LOGIC;
 8             B    : IN   STD_LOGIC;
 9             Sel0 : IN   STD_LOGIC;
10             F    : OUT  STD_LOGIC);
11  END ENTITY Two_To_One_Mux_DataFlow;
12
13  ARCHITECTURE DataFlow OF Two_To_One_Mux_DataFlow IS
14
15  BEGIN
16
17        F <= ((A AND NOT(Sel0)) OR (B AND Sel0));
18
19  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

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  USE ieee.std_logic_arith.all;
 4  USE ieee.std_logic_unsigned.all;
 5
 6  ENTITY Two_To_One_Mux_Behavioural is
 7      PORT ( A    : IN   STD_LOGIC;
 8             B    : IN   STD_LOGIC;
 9             Sel0 : IN   STD_LOGIC;
10             F    : OUT  STD_LOGIC);
11  END ENTITY Two_To_One_Mux_Behavioural;
12
13  ARCHITECTURE Behavioural OF Two_To_One_Mux_Behavioural IS
14
15  BEGIN
16
17  Mux_Process: PROCESS(A, B, Sel0)
18
19  BEGIN
20
21                 F <= ((A AND NOT(Sel0)) OR (B AND Sel0));
22
23  END PROCESS;
24
25  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 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;
```

Inverter

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;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.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;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.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

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  USE ieee.std_logic_arith.all;
 4  USE ieee.std_logic_unsigned.all;
 5
 6  ENTITY Signals_1 IS
 7      PORT ( A    : IN   STD_LOGIC;
 8             B    : IN   STD_LOGIC;
 9             C    : IN   STD_LOGIC;
10             Out1 : OUT  STD_LOGIC;
11             Out2 : OUT  STD_LOGIC);
12  END ENTITY Signals_1;
13
14  ARCHITECTURE DataFlow OF Signals_1 IS
15
16  SIGNAL Out1_Internal : STD_LOGIC;
17
18  BEGIN
19
20        Out1_Internal <= (A AND B) OR C;
21
22        Out1 <= Out1_Internal;
23
24        Out2 <= NOT(Out1_Internal);
25
26  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';
      Wait for 10 ns;

END PROCESS;

END ARCHITECTURE behavioural;
```

Figure 6.30: Example combinational logic circuit test bench

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_arith.all;
4   USE ieee.std_logic_unsigned.all;
5
6   ENTITY And_Gate_Variables is
7       PORT ( A    : IN   STD_LOGIC;
8              B    : IN   STD_LOGIC;
9              Z    : OUT  STD_LOGIC);
10  END ENTITY And_Gate_Variables;
11
12  ARCHITECTURE Behavioural OF And_Gate_Variables IS
13
14  BEGIN
15
16  AndGate_Process: PROCESS(A, B)
17
18  VARIABLE Tmp: STD_LOGIC;
19
20  BEGIN
21
22                Tmp := (A AND B);
23                Z <= Tmp;
24
25  END PROCESS;
26
27  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

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.std_logic_arith.all;
4   USE ieee.std_logic_unsigned.all;
5
6   ENTITY Half_Adder IS
7       PORT ( A    : IN   STD_LOGIC;
8              B    : IN   STD_LOGIC;
9              Sum  : OUT  STD_LOGIC;
10             Cout : OUT  STD_LOGIC);
11  END ENTITY Half_Adder;
12
13  ARCHITECTURE Behavioural OF Half_Adder IS
14
15  BEGIN
16
17         Sum  <= (A XOR B);
18         Cout <= (A AND B);
19
20  END ARCHITECTURE Behavioural;
```

Figure 6.41: VHDL code for a one-bit half-adder

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.std_logic_arith.all;
4   USE ieee.std_logic_unsigned.all;
5
6   ENTITY Test_Half_Adder_vhd IS
7   END Test_Half_Adder_vhd;
8
9   ARCHITECTURE behavioural OF Test_Half_Adder_vhd IS
10
11  COMPONENT Half_Adder
12  PORT(
13        A     : IN  STD_LOGIC;
14        B     : IN  STD_LOGIC;
15        Sum   : OUT STD_LOGIC;
16        Cout  : OUT STD_LOGIC);
17  END COMPONENT;
18
19  SIGNAL A :  STD_LOGIC:= '0';
20  SIGNAL B :  STD_LOGIC:= '0';
21
22  SIGNAL Sum  :  STD_LOGIC;
23  SIGNAL Cout :  STD_LOGIC;
24
25  BEGIN
26
27  uut: Half_Adder PORT MAP(
28        A    => A,
29        B    => B,
30        Sum  => Sum,
31        Cout => Cout);
32
33  Test_Bench_Process : PROCESS
34  BEGIN
35        Wait for 0 ns;   A <= '0';  B <= '0';
36        Wait for 10 ns;  A <= '0';  B <= '1';
37        Wait for 10 ns;  A <= '1';  B <= '0';
38        Wait for 10 ns;  A <= '1';  B <= '1';
39        Wait for 10 ns;
40  END PROCESS;
41
42  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 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;

-----------------------------------------------------------------------
-- Hex_Converter Architecture
-----------------------------------------------------------------------

ARCHITECTURE Behavioural OF Hex_Converter IS

BEGIN

---------------------------------------
-- 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 <= '0';
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 <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '0'; g <= '0';
When "1000" =>   a <= '1'; b <= '1'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
When "1001" =>   a <= '1'; b <= '1'; c <= '1'; d <= '0'; e <= '0'; f <= '1'; g <= '1';
When "1010" =>   a <= '1'; b <= '1'; c <= '1'; d <= '0'; e <= '1'; f <= '1'; g <= '1';
When "1011" =>   a <= '0'; b <= '0'; c <= '1'; d <= '1'; e <= '1'; f <= '1'; g <= '1';
When "1100" =>   a <= '1'; b <= '0'; c <= '0'; d <= '1'; e <= '1'; f <= '1'; g <= '0';
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 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 <= '0'; e <= '0'; f <= '1'; g <= '1';
ELSIF (Data_In = "1010") THEN
                  a <= '1'; b <= '1'; c <= '1'; d <= '0'; e <= '1'; 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 <= '0'; b <= '1'; c <= '1'; 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

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY D_Latch is
7       PORT ( D    : IN   STD_LOGIC;
8              CLK  : IN   STD_LOGIC;
9              Q    : OUT  STD_LOGIC);
10  END ENTITY D_Latch;
11
12  ARCHITECTURE Behavioural OF D_Latch IS
13
14  BEGIN
15
16  PROCESS(CLK, D)
17
18  BEGIN
19
20          If (CLK = '1') THEN
21                 Q <= D;
22          END IF;
23
24  END PROCESS;
25
26  END ARCHITECTURE Behavioural;
```

Figure 6.68: VHDL code for the D-latch

```
 1  LIBRARY IEEE;
 2  USE IEEE.STD_LOGIC_1164.ALL;
 3  USE IEEE.STD_LOGIC_ARITH.ALL;
 4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  ENTITY D_Latch_Array is
 7      PORT ( D    : IN   STD_LOGIC_VECTOR(7 downto 0);
 8             CLK  : IN   STD_LOGIC;
 9             Q    : OUT  STD_LOGIC_VECTOR(7 downto 0));
10  END ENTITY D_Latch_Array;
11
12  ARCHITECTURE Behavioural OF D_Latch_Array IS
13
14  BEGIN
15
16  PROCESS(CLK, D)
17
18  BEGIN
19
20                If (CLK = '1') THEN
21                      Q(7 downto 0) <= D(7 downto 0);
22                END IF;
23
24  END PROCESS;
25
26  END ARCHITECTURE Behavioural;
```

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

```
 1  LIBRARY IEEE;
 2  USE IEEE.STD_LOGIC_1164.ALL;
 3  USE IEEE.STD_LOGIC_ARITH.ALL;
 4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  ENTITY D_Type_Bistable is
 7      PORT ( D    : IN   STD_LOGIC;
 8             CLK  : IN   STD_LOGIC;
 9             Q    : OUT  STD_LOGIC);
10  END ENTITY D_Type_Bistable;
11
12  ARCHITECTURE Behavioural OF D_Type_Bistable IS
13
14  BEGIN
15
16  PROCESS(CLK, D)
17
18  BEGIN
19
20                  IF (CLK'EVENT AND CLK = '1') THEN
21                        Q <= D;
22                  END IF;
23
24  END PROCESS;
25
26  END ARCHITECTURE Behavioural;
```

Figure 6.72: VHDL code for the D-type bistable

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY D_Type_Register is
7       PORT ( D      : IN   STD_LOGIC_VECTOR(7 downto 0);
8              CLK    : IN   STD_LOGIC;
9              RESET  : IN   STD_LOGIC;
10             Q      : OUT  STD_LOGIC_VECTOR(7 downto 0));
11  END ENTITY D_Type_Register;
12
13  ARCHITECTURE Behavioural OF D_Type_Register IS
14
15  BEGIN
16
17  PROCESS(CLK, D, RESET)
18
19  BEGIN
20
21         IF (RESET = '0') THEN
22
23                Q(7 downto 0) <= "00000000";
24
25         ELSIF (CLK'EVENT AND CLK = '1') THEN
26
27                Q(7 downto 0) <= D(7 downto 0);
28
29         END IF;
30
31  END PROCESS;
32
33  END ARCHITECTURE Behavioural;
```

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

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY Four_Bit_Counter is
7       PORT ( Clock : IN   STD_LOGIC;
8              Reset : IN   STD_LOGIC;
9              Count : OUT  STD_LOGIC_VECTOR (3 downto 0));
10  END ENTITY Four_Bit_Counter;
11
12  ARCHITECTURE Behavioural of Four_Bit_Counter is
13
14  SIGNAL Count_Int : STD_LOGIC_VECTOR(3 downto 0);
15
16  BEGIN
17
18  PROCESS(Clock, Reset)
19
20  BEGIN
21
22        IF (Reset = '0') THEN
23
24              Count_Int(3 downto 0) <= "0000";
25
26        ELSIF (Clock'Event AND Clock = '1') THEN
27
28              Count_Int(3 downto 0) <= Count_Int(3 downto 0) + 1;
29
30        END IF;
31
32  END PROCESS;
33
34  Count(3 downto 0) <= Count_Int(3 downto 0);
35
36  END ARCHITECTURE Behavioural;
```

Figure 6.76: VHDL code for the four-bit binary counter

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.ALL;
 3  USE ieee.std_logic_unsigned.all;
 4  USE ieee.numeric_std.ALL;
 5
 6  ENTITY Test_Four_Bit_Counter_vhd IS
 7  END Test_Four_Bit_Counter_vhd;
 8
 9  ARCHITECTURE Behavioural OF Test_Four_Bit_Counter_vhd IS
10
11  COMPONENT Four_Bit_Counter
12  PORT(
13               Clock : IN  std_logic;
14               Reset : IN  std_logic;
15               Count : OUT std_logic_vector(3 downto 0));
16  END COMPONENT;
17
18  SIGNAL Clock :  std_logic := '0';
19  SIGNAL Reset :  std_logic := '0';
20
21  SIGNAL Count :  std_logic_vector(3 downto 0);
22
23  BEGIN
24
25  uut: Four_Bit_Counter PORT MAP(
26               Clock => Clock,
27               Reset => Reset,
28               Count => Count);
29
30  Reset_Process: PROCESS
31
32  BEGIN
33
34        Wait For 0 ns;    Reset <= '0';
35        Wait For 160 ns;  Reset <= '1';
36        Wait;
37
38  END PROCESS;
39
40  Clock_Process: PROCESS
41
42  BEGIN
43
44        Wait For 0 ns;   Clock <= '0';
45        Wait For 20 ns;  Clock <= '1';
46        Wait For 20 ns;  Clock <= '0';
47
48  END PROCESS;
49
50  END ARCHITECTURE Behavioural;
```

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

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  USE ieee.std_logic_arith.all;
 4  USE ieee.std_logic_unsigned.all;
 5
 6  ENTITY Sequence_Detector is
 7      PORT ( Data_In  : IN   STD_LOGIC;
 8             Clock    : IN   STD_LOGIC;
 9             Reset    : IN   STD_LOGIC;
10             Q2       : OUT  STD_LOGIC;
11             Q1       : OUT  STD_LOGIC;
12             Q0       : OUT  STD_LOGIC;
13             Detected : OUT  STD_LOGIC);
14  END ENTITY Sequence_Detector;
15
16  ARCHITECTURE Behavioural OF Sequence_Detector IS
17
18  TYPE State_Type IS (State0, State1, State2, State3, State4);
19
20  SIGNAL Present_State, Next_State : State_Type;
21
22  BEGIN
23
24  PROCESS(Clock, Reset)
25  BEGIN
26
27          IF (Reset = '0') THEN
28
29                          Present_State <= State0;
30
31          ELSIF (Clock'Event AND Clock = '1') THEN
32
33                          Present_State <= Next_State;
34
35          END IF;
36
37  END PROCESS;
38
39  PROCESS(Present_State, Data_In)
30  BEGIN
41
42          CASE Present_State IS
43
44
45          WHEN State0 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '0';
46
47                          IF (Data_In = '0') THEN
48                                          Next_State <= State0;
49                          ELSE
50                                          Next_State <= State1;
51                          END IF;
```

Figure 6.81: VHDL code for the 1001 sequence detector

```
52
53
54          WHEN State1 =>  Detected <= '0';  Q2 <= '0';  Q1 <= '0';  Q0 <= '1';
55
56                          IF (Data_In = '0') THEN
57                                              Next_State <= State2;
58                          ELSE
59                                              Next_State <= State1;
60                          END IF;
61
62
63          WHEN State2 =>  Detected <= '0'; Q2 <= '0';  Q1 <= '1';  Q0 <= '0';
64
65                          IF (Data_In = '0') THEN
66                                              Next_State <= State3;
67                          ELSE
68                                              Next_State <= State1;
69                          END IF;
70
71
72          WHEN State3 =>  Detected <= '0'; Q2 <= '0';  Q1 <= '1';  Q0 <= '1';
73
74                          IF (Data_In = '0') THEN
75                                              Next_State <= State0;
76                          ELSE
77                                              Next_State <= State4;
78                          END IF;
79
80
81          WHEN State4 =>  Detected <= '1'; Q2 <= '1';  Q1 <= '0';  Q0 <= '0';
82
83                         IF (Data_In = '0') THEN
84                                          Next_State <= State0;
85                          ELSE
86                                          Next_State <= State1;
87                          END IF;
88
89
90          END CASE;
91
92      END PROCESS;
93
94      END ARCHITECTURE Behavioural;
```

Figure 6.81: (Continued)

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.std_logic_unsigned.all;
4  USE ieee.numeric_std.ALL;
5
6
7  ENTITY Test_Sequence_Detector_vhd IS
8  END Test_Sequence_Detector_vhd;
9
10
11 ARCHITECTURE Behavioural OF Test_Sequence_Detector_vhd IS
12
13 COMPONENT Sequence_Detector
14 PORT(
15              Data_In   : IN   STD_LOGIC;
16              Clock     : IN   STD_LOGIC;
17              Reset     : IN   STD_LOGIC;
18              Q2        : OUT  STD_LOGIC;
19              Q1        : OUT  STD_LOGIC;
20              Q0        : OUT  STD_LOGIC;
21              Detected  : OUT  STD_LOGIC);
22 END COMPONENT;
23
24 SIGNAL Data_In  :  STD_LOGIC := '0';
25 SIGNAL Clock    :  STD_LOGIC := '0';
26 SIGNAL Reset    :  STD_LOGIC := '0';
27
28 SIGNAL Q2       :  STD_LOGIC;
29 SIGNAL Q1       :  STD_LOGIC;
30 SIGNAL Q0       :  STD_LOGIC;
31 SIGNAL Detected :  STD_LOGIC;
32
33 BEGIN
34
35
36 uut: Sequence_Detector PORT MAP(
37              Data_In   => Data_In,
38              Clock     => Clock,
39              Reset     => Reset,
30              Q2        => Q2,
41              Q1        => Q1,
42              Q0        => Q0,
43              Detected  => Detected);
44
```

Figure 6.82: VHDL test bench for the 1001 sequence detector

```
45  Reset_Process : PROCESS
46
47  BEGIN
48
49        Wait for 0 ns;  Reset <= '0';
50        Wait for 5 ns;  Reset <= '1';
51        Wait;
52
53  END PROCESS;
54
55  Clock_Process : PROCESS
56
57  BEGIN
58
59        Wait for 0 ns;   Clock <= '0';
60        Wait for 10 ns;  Clock <= '1';
61        Wait for 10 ns;  Clock <= '0';
62
63  END PROCESS;
64
65  Data_In_Process : PROCESS
66
67  BEGIN
68
69        Wait for 0 ns;    Data_In <= '0';
70
71        Wait for 80 ns;   Data_In <= '1';
72        Wait for 20 ns;   Data_In <= '0';
73        Wait for 20 ns;   Data_In <= '0';
74        Wait for 20 ns;   Data_In <= '1';
75
76        Wait for 20 ns;   Data_In <= '0';
77
78        Wait for 80 ns;   Data_In <= '1';
79        Wait for 20 ns;   Data_In <= '0';
80        Wait for 20 ns;   Data_In <= '0';
81        Wait for 20 ns;   Data_In <= '0';
82
83        Wait for 20 ns;   Data_In <= '0';
84
85  END PROCESS;
86
87  END ARCHITECTURE Behavioural;
```

Figure 6.82: (Continued)

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   ENTITY Uart_Receiver is
8       PORT ( Rx          : IN   STD_LOGIC;
9              Clock       : IN   STD_LOGIC;
10             Reset       : IN   STD_LOGIC;
11             Data_Rx     : OUT  STD_LOGIC_VECTOR(7 downto 0);
12             DR          : OUT  STD_LOGIC);
13  END ENTITY Uart_Receiver;
14
15
16  ARCHITECTURE Behavioural of Uart_Receiver is
17
18  SIGNAL Count     : STD_LOGIC_VECTOR(7 downto 0);
19  SIGNAL Data_Int  : STD_LOGIC_VECTOR(7 downto 0);
20
21  BEGIN
22
23  PROCESS (Clock, Reset)
24
25  BEGIN
26
27          IF (Reset='0') then
28
29                  Count <= "00000000";
30
31          ELSIF (Clock'Event and Clock = '1') then
32
33          IF (Rx='1' AND (Count = "00000000" or Count = "10101011")) THEN
34                          Count <= "00000000";
35          ELSE
36                          Count <= Count + 1;
37          END IF;
38
39          END IF;
40
41  END PROCESS;
42
43  PROCESS (Clock, Reset, Count)
44
45  BEGIN
46
47          IF (Reset='0') THEN
48
49                  Data_Int(7 downto 0) <= "00000000";
50                  Data_Rx(7 downto 0)  <= "00000000";
51                  DR <= '0';
52
53          ELSIF (Clock'Event and Clock = '1') THEN
54
55                  IF (COUNT = "00000000") THEN
56                          DR <= '0';
57                  END IF;
58
```

Figure 6.85: VHDL code for a UART receiver

```
59                  IF (COUNT = "00000001") THEN
60                          DR <= '0';
61                  END IF;
62
63                  IF (COUNT = "00011000") THEN
64                          Data_Int(0) <= Rx;
65                  END IF;
66
67                  IF (COUNT = "00101000") THEN
68                           Data_Int(1) <= Rx;
69                  END IF;
70
71                  IF (COUNT = "00111000") THEN
72                          Data_Int(2) <= Rx;
73                  END IF;
74
75                  IF (COUNT = "01001000") THEN
76                          Data_Int(3) <= Rx;
77                  END IF;
78
79                  IF (COUNT = "01011000") THEN
80                          Data_Int(4) <= Rx;
81                  END IF;
82
83                  IF (COUNT = "01101000") THEN
84                          Data_Int(5) <= Rx;
85                  END IF;
86
87                  IF (COUNT = "01111000") THEN
88                          Data_Int(6) <= Rx;
89                  END IF;
90
91                  IF (COUNT = "10001000") THEN
92                          Data_Int(7) <= Rx;
93                  END IF;
94
95                  IF (COUNT = "10011000") THEN
96                          Data_Rx(7 downto 0) <= Data_Int(7 downto 0);
97                  END IF;
98
99                  IF (COUNT = "10101000") THEN
100                         DR <= '1';
101                 END IF;
102
103                 IF (COUNT = "10101010") THEN
104                         DR <= '0';
105                 END IF;
106
107         END IF;
108
109 END PROCESS;
110
111 END ARCHITECTURE Behavioural;
```

Figure 6.85: (Continued)

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.ALL;
 3  USE ieee.std_logic_unsigned.all;
 4  USE ieee.numeric_std.ALL;
 5
 6  ENTITY Test_Uart_Receiver_vhd IS
 7  END Test_Uart_Receiver_vhd;
 8
 9  ARCHITECTURE Behavioural OF Test_Uart_Receiver_vhd IS
10
11  COMPONENT Uart_Receiver
12  PORT(
13                 Rx      : IN std_logic;
14                 Clock   : IN std_logic;
15                 Reset   : IN std_logic;
16                 Data_Rx : OUT std_logic_vector(7 downto 0);
17                 DR      : OUT std_logic);
18  END COMPONENT;
19
20  SIGNAL Rx     :  std_logic := '0';
21  SIGNAL Clock  :  std_logic := '0';
22  SIGNAL Reset  :  std_logic := '0';
23
24  SIGNAL Data_Rx  :  std_logic_vector(7 downto 0);
25  SIGNAL DR       :  std_logic;
26
27  BEGIN
28
29  uut: Uart_Receiver PORT MAP(
30                 Rx       => Rx,
31                 Clock    => Clock,
32                 Reset    => Reset,
33                 Data_Rx  => Data_Rx,
34                 DR       => DR);
35
36  Reset_Process: PROCESS
37
38  BEGIN
39
30         Wait for 0 ns;  Reset <= '0';
41         Wait for 5 ns;  Reset <= '1';
42         Wait;
43
44  END PROCESS;
45
46  Clock_Process: PROCESS
47
```

Figure 6.86: VHDL test bench for the UART receiver

```
48  BEGIN
49
50          Wait for 0 ns;   Clock <= '0';
51          Wait for 10 ns;  Clock <= '1';
52          Wait for 10 ns;  Clock <= '0';
53
54  END PROCESS;
55
56  Rx_Process: PROCESS
57
58  BEGIN
59
60          Wait for 0 ns;    Rx <= '1';
61          Wait for 100 ns;  Rx <= '0';
62          Wait for 320 ns;  Rx <= '1';
63
64          Wait for 5000 ns;
65
66          Wait for 0 ns;    Rx <= '1';
67          Wait for 100 ns;  Rx <= '0';
68          Wait for 320 ns;  Rx <= '1';
69          Wait for 320 ns;  Rx <= '0';
70          Wait for 320 ns;  Rx <= '1';
71
72          Wait for 5000 ns;
73
74  END PROCESS;
75
76  END ARCHITECTURE behavioural;
```

Figure 6.86: (Continued)

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.std_logic_unsigned.all;
4   USE ieee.numeric_std.ALL;
5
6   ENTITY RAM_Model is
7       PORT ( Address  : IN    Integer range 0 to 15;
8              CE       : IN    STD_LOGIC;
9              WE       : IN    STD_LOGIC;
10             OE       : IN    STD_LOGIC;
11             Data     : INOUT STD_LOGIC_VECTOR(7 downto 0));
12  END ENTITY RAM_Model;
13
14  ARCHITECTURE Behavioural OF RAM_Model IS
15
16  BEGIN
17
18  PROCESS(Address, CE, WE, OE) IS
19
20  TYPE Ram_Array IS ARRAY (0 to 15) OF STD_LOGIC_VECTOR(7 downto 0);
21
22  VARIABLE Mem: Ram_Array;
23
24  BEGIN
25
26  Data(7 downto 0) <= (others => 'Z');
27
28          IF (CE = '0') THEN
29                  IF (WE = '0') THEN
30                          Mem(Address) := Data(7 downto 0);
31                  ELSIF (OE = '0') THEN
32                          Data(7 downto 0) <= Mem(Address);
33                  END IF;
34          END IF;
35
36  END PROCESS;
37
38  END ARCHITECTURE Behavioural;
```

Figure 6.88: 16 × 8 RAM

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.std_logic_unsigned.all;
4   USE ieee.numeric_std.ALL;
5
6   ENTITY Test_RAM_Model_vhd IS
7   END Test_RAM_Model_vhd;
8
9   ARCHITECTURE behavioural OF Test_RAM_Model_vhd IS
10
11  COMPONENT RAM_Model
12  PORT(
13         Address  : IN     Integer range 0 to 15;
14         CE       : IN     STD_LOGIC;
15         WE       : IN     STD_LOGIC;
16         OE       : IN     STD_LOGIC;
17         Data     : INOUT  STD_LOGIC_VECTOR(7 downto 0));
18  END COMPONENT;
19
20  SIGNAL CE      :  STD_LOGIC := '0';
21  SIGNAL WE      :  STD_LOGIC := '0';
22  SIGNAL OE      :  STD_LOGIC := '0';
23  SIGNAL Address :  Integer range 0 to 15;
24
25  SIGNAL Data :  STD_LOGIC_VECTOR(7 downto 0);
26
27  BEGIN
28
29  uut: RAM_Model PORT MAP(
30                 Address  =>  Address,
31                 CE       =>  CE,
32                 WE       =>  WE,
33                 OE       =>  OE,
34                 Data     =>  Data);
35
36  Test_Bench_Process : PROCESS
37
38  BEGIN
39
40  wait for 0 ns;   Address <= 0;  Data <= "ZZZZZZZZ";
41                   CE <= '1';     WE <= '1';   OE <= '1';
42
43  wait for 10 ns;  Address <= 0;  Data <= "10000001";
44  wait for 10 ns;  CE <= '0';     WE <= '1';   OE <= '1';
45  wait for 10 ns;  CE <= '0';     WE <= '0';   OE <= '1';
46  wait for 10 ns;  CE <= '1';     WE <= '1';   OE <= '1';   Data <= "ZZZZZZZZ";
47
48  wait for 10 ns;  Address <= 0;  Data <= "ZZZZZZZZ";
49  wait for 10 ns;  CE <= '0';     WE <= '1';   OE <= '1';
50  wait for 10 ns;  CE <= '0';     WE <= '1';   OE <= '0';
51  wait for 10 ns;  CE <= '1';     WE <= '1';   OE <= '1';
52
53  wait for 10 ns;
54
55  END PROCESS;
56
57  END ARCHITECTURE Behavioural;
```

Figure 6.89: VHDL test bench for the 16 × 8 RAM

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY ROM is
7       Port ( Address :   IN    INTEGER Range 0 to 15;
8              Data    :   OUT   STD_LOGIC_VECTOR(7 downto 0));
9   END ENTITY ROM;
10
11  ARCHITECTURE Behavioural of ROM is
12
13  TYPE Rom_Array IS Array (0 to 15) of STD_LOGIC_VECTOR(7 downto 0);
14
15  CONSTANT ROM: Rom_Array := (
16                                  "11000000",
17                                  "00010011",
18                                  "00100000",
19                                  "00110000",
20                                  "01000000",
21                                  "01010000",
22                                  "01100000",
23                                  "01110000",
24                                  "10000000",
25                                  "10011000",
26                                  "10100000",
27                                  "10110000",
28                                  "11000000",
29                                  "11010000",
30                                  "11100011",
31                                  "11111111");
32
33  BEGIN
34
35         Data <= Rom(Address);
36
37  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;

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

```
 1  LIBRARY IEEE;
 2  USE IEEE.STD_LOGIC_1164.ALL;
 3  USE IEEE.STD_LOGIC_ARITH.ALL;
 4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  ENTITY Adder1 IS
 7      PORT ( A : IN   STD_LOGIC_VECTOR (7 downto 0);
 8             B : IN   STD_LOGIC_VECTOR (7 downto 0);
 9             P : OUT  STD_LOGIC_VECTOR (8 downto 0));
10  END ENTITY Adder1;
11
12  ARCHITECTURE DataFlow of Adder1 is
13
14  SIGNAL A_Int : STD_LOGIC_VECTOR(8 downto 0);
15  SIGNAL B_Int : STD_LOGIC_VECTOR(8 downto 0);
16
17  BEGIN
18
19          A_Int(8) <= '0';
20          A_Int(7 downto 0) <= A(7 downto 0);
21          B_Int(8) <= '0';
22          B_Int(7 downto 0) <= B(7 downto 0);
23          P(8 downto 0) <= A_Int(8 downto 0) + B_Int(8 downto 0);
24
25  END ARCHITECTURE DataFlow;
```

Figure 6.94: Unsigned addition

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY Adder2 IS
7       PORT ( A : IN   STD_LOGIC_VECTOR (7 downto 0);
8              B : IN   STD_LOGIC_VECTOR (7 downto 0);
9              P : OUT  STD_LOGIC_VECTOR (8 downto 0));
10  END ENTITY Adder2;
11
12  ARCHITECTURE DataFlow of Adder2 is
13
14  SIGNAL Signed_A : SIGNED(8 downto 0);
15  SIGNAL Signed_B : SIGNED(8 downto 0);
16  SIGNAL Signed_P : SIGNED(8 downto 0);
17
18  BEGIN
19
20          Signed_A(8) <= Signed_A(7);
21          Signed_A(7 downto 0)   <= Signed(A(7 downto 0));
22          Signed_B(8) <= Signed_B(7);
23          Signed_B(7 downto 0)   <= Signed(B(7 downto 0));
25
26          Signed_P(8 downto 0) <= Signed_A(8 downto 0) + Signed_B(8 downto 0)
27          P <= STD_LOGIC_VECTOR(Signed_P(8 downto 0));
28
29  END ARCHITECTURE DataFlow;
```

Figure 6.95: Signed addition

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.ALL;
 3  USE ieee.std_logic_unsigned.all;
 4  USE ieee.numeric_std.ALL;
 5
 6  ENTITY Test_Adder1_vhd IS
 7  END Test_Adder1_vhd;
 8
 9  ARCHITECTURE Behavioural OF Test_Adder1_vhd IS
10
11  COMPONENT Adder1
12  PORT(
13                A : IN std_logic_vector(7 downto 0);
14                B : IN std_logic_vector(7 downto 0);
15                P : OUT std_logic_vector(7 downto 0));
16  END COMPONENT;
17
18  SIGNAL A :  std_logic_vector(7 downto 0) := (others=>'0');
19  SIGNAL B :  std_logic_vector(7 downto 0) := (others=>'0');
20
21  SIGNAL P :  std_logic_vector(7 downto 0);
22
23  BEGIN
24
25  uut: Adder1 PORT MAP(
26                A => A,
27                B => B,
28                P => P);
29
30  Test_Bench_Process : PROCESS
31
32  BEGIN
33
34  WAIT for 0 ns;   A <= "00000000";    B <= "00000000";
35  WAIT for 10 ns;  A <= "00000001";    B <= "00000000";
36  WAIT for 10 ns;  A <= "00000001";    B <= "00000001";
37  WAIT for 10 ns;  A <= "11111111";    B <= "00000000";
38  WAIT for 10 ns;  A <= "11111111";    B <= "00000001";
39  WAIT for 10 ns;  A <= "11111101";    B <= "11111101";
40  WAIT for 10 ns;  A <= "11111111";    B <= "11111111";
41  WAIT for 10 ns;
42
43  END PROCESS;
44
45  END ARCHITECTURE Behavioural;
```

Figure 6.96: Addition test bench

```
 1  LIBRARY IEEE;
 2  USE IEEE.STD_LOGIC_1164.ALL;
 3  USE IEEE.STD_LOGIC_ARITH.ALL;
 4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6
 7  ENTITY Unsigned_Multiplier is
 8      PORT ( A : IN  STD_LOGIC_VECTOR (7 downto 0);
 9             B : IN  STD_LOGIC_VECTOR (7 downto 0);
10             P : OUT STD_LOGIC_VECTOR (15 downto 0));
11  END ENTITY Unsigned_Multiplier;
12
13
14  ARCHITECTURE DataFlow of Unsigned_Multiplier is
15
16  BEGIN
17
18         P <= A * B;
19
20  END ARCHITECTURE DataFlow;
```

Figure 6.97: Eight-bit unsigned multiplication

```
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

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_ARITH.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   ENTITY Signed_Multiplier is
8       PORT ( A : IN  STD_LOGIC_VECTOR (7 downto 0);
9              B : IN  STD_LOGIC_VECTOR (7 downto 0);
10             P : OUT STD_LOGIC_VECTOR (15 downto 0));
11  END ENTITY Signed_Multiplier;
12
13
14
15  ARCHITECTURE DataFlow of Signed_Multiplier is
16
17  SIGNAL  A_Signed: SIGNED(7 downto 0);
18  SIGNAL  B_Signed: SIGNED(7 downto 0);
19  SIGNAL  P_Signed: SIGNED(15 downto 0);
20
21  BEGIN
22
23          A_Signed(7 downto 0)  <=  SIGNED(A(7 downto 0));
24          B_Signed(7 downto 0)  <=  SIGNED(B(7 downto 0));
25
26          P_Signed(15 downto 0) <= A_Signed(7 downto 0) * B_Signed(7 downto 0);
27
28          P(15 downto 0) <= STD_LOGIC_VECTOR(P_Signed(15 downto 0));
29
30  END ARCHITECTURE DataFlow;
```

Figure 6.101: Eight-bit signed multiplication

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.ALL;
 3
 4
 5  ENTITY test_And_Gate_vhd IS
 6  END ENTITY test_And_Gate_vhd;
 7
 8
 9  ARCHITECTURE Behavioural OF test_And_Gate_vhd IS
10
11  COMPONENT And_Gate
12          PORT(
13                  A : IN std_logic;
14                  B : IN std_logic;
15                  Z : OUT std_logic
16                  );
17  END COMPONENT;
18
19  SIGNAL A :  std_logic := '0';
20  SIGNAL B :  std_logic := '0';
21
22  SIGNAL Z :  std_logic;
23
24  BEGIN
25
26  uut: And_Gate PORT MAP(
27                  A => A,
28                  B => B,
29                  Z => Z);
30
31  Input_Process : PROCESS
32
33  BEGIN
34
35          WAIT FOR 0 ns;   A <= '0'; B <= '0';
36          WAIT FOR 10 ns;  A <= '1'; B <= '0';
37          WAIT FOR 10 ns;  A <= '0'; B <= '1';
38          WAIT FOR 10 ns;  A <= '1'; B <= '1';
39          WAIT FOR 10 ns;
40
41  END PROCESS;
42
43  END ARCHITECTURE Behavioural;
```

Figure 6.103: Two-input AND gate test bench

```
1  ----------------------------------------------------------------
2  -- Combinational Logic Circuit Design: Dataflow Description
3  ----------------------------------------------------------------
4
5  ----------------------------------------------------------------
6  -- Libraries and packages to use
7  ----------------------------------------------------------------
8
9  LIBRARY ieee;
10 USE ieee.std_logic_1164.all;
11 USE ieee.std_logic_arith.all;
12 USE ieee.std_logic_unsigned.all;
13
14 ----------------------------------------------------------------
15 -- Top Entity
16 ----------------------------------------------------------------
17
18 ENTITY Circuit1 IS
19
20     PORT (    A : IN   STD_LOGIC;
21               B : IN   STD_LOGIC;
22               C : IN   STD_LOGIC;
23               Z : OUT  STD_LOGIC);
24
25 END ENTITY Circuit1;
26
27 ----------------------------------------------------------------
28 -- Top Architecture
29 ----------------------------------------------------------------
30
31 ARCHITECTURE Behavioural OF Circuit1 IS
32
33 BEGIN
34
35        Z <= (A AND B) OR NOT((NOT(A OR B)) AND (A OR C));
36
37 END ARCHITECTURE Behavioural;
38
39 ----------------------------------------------------------------
40 -- End of File
41 ----------------------------------------------------------------
```

Figure 6.104: Combinational logic circuit description

```
1  ----------------------------------------------------------
2  -- Test bench for Circuit1
3  ----------------------------------------------------------
4
5  ----------------------------------------------------------
6  -- Libraries and packages to use
7  ----------------------------------------------------------
8
9  LIBRARY ieee;
10 USE ieee.std_logic_1164.ALL;
11 USE ieee.std_logic_unsigned.ALL;
12 USE ieee.numeric_std.ALL;
13
14 USE std.textio.ALL;
15
16 ----------------------------------------------------------
17 -- Test bench Entity
18 ----------------------------------------------------------
19
20 ENTITY Test_Circuit1_vhd IS
21 END Test_Circuit1_vhd;
22
23 ARCHITECTURE Behavioural OF Test_Circuit1_vhd IS
24
25 ----------------------------------------------------------
26 -- Component Declaration for the Unit Under Test (UUT)
27 ----------------------------------------------------------
28
29 COMPONENT Circuit1
30 PORT(
31                A : IN   STD_LOGIC;
32                B : IN   STD_LOGIC;
33                C : IN   STD_LOGIC;
34                Z : OUT  STD_LOGIC);
35 END COMPONENT;
36
37 ----------------------------------------------------------
38 -- Inputs
39 ----------------------------------------------------------
40
```

Figure 6.106: Combinational logic circuit test bench (1)

```
41  SIGNAL A :  STD_LOGIC := '0';
42  SIGNAL B :  STD_LOGIC := '0';
43  SIGNAL C :  STD_LOGIC := '0';
44
45  ------------------------------------------------------------
46  -- Outputs
47  ------------------------------------------------------------
48
49  SIGNAL Z :  STD_LOGIC;
50
51  ------------------------------------------------------------
52
53  BEGIN
54
55  ------------------------------------------------------------
56  -- Instantiate the Unit Under Test (UUT)
57  ------------------------------------------------------------
58
59  uut: Circuit1 PORT MAP(
60                A  =>  A,
61                B  =>  B,
62                C  =>  C,
63                Z  =>  Z);
64
65  ------------------------------------------------------------
66  -- Read from stimulus file and apply to circuit process
67  ------------------------------------------------------------
68
69  Process_1 : PROCESS
70
71  FILE      Stimulus_File     : TEXT;
72  FILE      Results_File      : TEXT;
73
74  VARIABLE  Input_Pattern     :  LINE;
75  VARIABLE  Results_Pattern   :  LINE;
76  VARIABLE  Read_OK           :  BOOLEAN;
77  VARIABLE  Char              :  CHARACTER;
78
79  VARIABLE  A_In, B_In, C_In  :  STD_LOGIC;
80
81  ------------------------------------
82
```

Figure 6.106: (Continued)

```
 83  BEGIN
 84
 85  ------------------------------------
 86  -- Open files for READ and WRITE
 87  ------------------------------------
 88
 89        FILE_OPEN(Stimulus_File,"C:\Circuit1_Stimulus.txt", READ_MODE);
 90        FILE_OPEN(Results_File, "C:\Circuit1_Results.txt", WRITE_MODE);
 91
 92  ------------------------------------
 93  -- Write header text to results file
 94  ------------------------------------
 95
 96        WRITE(Results_Pattern, "-----------------------------");
 97        WRITELINE(Results_File, Results_Pattern);
 98        WRITE(Results_Pattern, "ABC Z");
 99        WRITELINE(Results_File, Results_Pattern);
100        WRITE(Results_Pattern, "-----------------------------");
101        WRITELINE(Results_File, Results_Pattern);
102
103 ------------------------------------
104 -- Loop read from file and apply
105 -- stimulus until end of file
106 ------------------------------------
107
108       WHILE (NOT ENDFILE(Stimulus_File)) LOOP
109
110 ------------------------------------
111 -- Read line from 'Stimulus_File' into
112 -- variable 'Input_Pattern'
113 ------------------------------------
114
115       READLINE(Stimulus_File, Input_Pattern);
116
117 ------------------------------------
118 -- Read first character from
119 -- 'Input_Pattern'
120 ------------------------------------
121
122       READ(Input_Pattern, CHAR, Read_OK);
123
```

Figure 6.106: (Continued)

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

Figure 6.106: (Continued)

```
161         WRITE(Results_Pattern, A);
162         WRITE(Results_Pattern, B);
163         WRITE(Results_Pattern, C);
164         WRITE(Results_Pattern, " ");
165         WRITE(Results_Pattern, Z)
166
167         WRITELINE(Results_File, Results_Pattern);
168
169     -------------------------------------
170     -- End of Loop
171     -------------------------------------
172
173          END LOOP;
174
175     -------------------------------------
176     -- Write footer text to results file
177     -------------------------------------
178
179         WRITE(Results_Pattern, "-----------------------------");
180         WRITELINE(Results_File, Results_Pattern);
181         WRITE(Results_Pattern, "-- Test completed");
182         WRITELINE(Results_File, Results_Pattern);
183         WRITE(Results_Pattern, "-----------------------------");
185         WRITELINE(Results_File, Results_Pattern);
185
186     -------------------------------------
187     -- Close the OPENed files
188     -------------------------------------
189
190         FILE_CLOSE(Stimulus_File);
191         FILE_CLOSE(Results_File);
192
193     -------------------------------------
194     -- Stop process or it will repeat if
195     -- simulation time longer than a
196     -- single pass of the input and will
197     -- overwrite results file
198     -------------------------------------
```

Figure 6.106: (Continued)

```
199
200         WAIT;
201
202     ------------------------------------
203     END PROCESS;
204
205     ------------------------------------------------------------
206
207     END ARCHITECTURE Behavioural;
208
209     ------------------------------------------------------------------
210     -- End of File
211     ------------------------------------------------------------------
```

Figure 6.106: (Continued)

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

Figure 6.107: Combinational logic circuit test bench (2)

```
36
37    ------------------------------------------------------
38    -- Inputs
39    ------------------------------------------------------
40
41    SIGNAL A :  STD_LOGIC := '0';
42    SIGNAL B :  STD_LOGIC := '0';
43    SIGNAL C :  STD_LOGIC := '0';
44
45    ------------------------------------------------------
46    -- Outputs
47    ------------------------------------------------------
48
49    SIGNAL Z :  STD_LOGIC;
50
51    ------------------------------------------------------
52
53    BEGIN
54
55    ------------------------------------------------------
56    -- Instantiate the Unit Under Test (UUT)
57    ------------------------------------------------------
58
59    uut: Circuit1 PORT MAP(
60                  A  =>  A,
61                  B  =>  B,
62                  C  =>  C,
63                  Z  =>  Z);
64
65    ------------------------------------------------------
66    -- Read from stimulus file and apply to circuit process
67    ------------------------------------------------------
68
69    Process_1 : PROCESS
```

Figure 6.107: (Continued)

```
70
71    FILE      Stimulus_File     : TEXT;
72    FILE      Results_File      : TEXT;
73
74    VARIABLE  Input_Pattern     :  LINE;
75    VARIABLE  Results_Pattern   :  LINE;
76    VARIABLE  Read_OK           :  BOOLEAN;
77    VARIABLE  Char              :  CHARACTER;
78
79    VARIABLE  A_In, B_In, C_In  :  BIT;
80    Variable  Z_Out             :  BIT;
81
82    -------------------------------------
83
84    BEGIN
85
86    -------------------------------------
87    -- Open files for READ and WRITE
88    -------------------------------------
89
90          FILE_OPEN(Stimulus_File,"C:\Circuit1_Stimulus.txt",READ_MODE);
91          FILE_OPEN(Results_File,"C:\Circuit1_Results.txt", WRITE_MODE);
92
93    -------------------------------------
94    -- Write header text to results file
95    -------------------------------------
96
97          WRITE(Results_Pattern, "-----------------------------");
98          WRITELINE(Results_File, Results_Pattern);
99          WRITE(Results_Pattern, "ABC Z");
100         WRITELINE(Results_File, Results_Pattern);
101         WRITE(Results_Pattern, "-----------------------------");
102         WRITELINE(Results_File, Results_Pattern);
```

Figure 6.107: (Continued)

```
103
104     -------------------------------------
105     -- Loop read from file and apply
106     -- stimulus until end of file
107     -------------------------------------
108
109           WHILE (NOT ENDFILE(Stimulus_File)) LOOP
110
111     -------------------------------------
112     -- Read line from 'Stimulus_File' into
113     -- variable 'Input_Pattern'
114     -------------------------------------
115
116           READLINE(Stimulus_File, Input_Pattern);
117
118     -------------------------------------
119     -- Read first character from
120     -- 'Input_Pattern'
121     -------------------------------------
122
123           READ(Input_Pattern, CHAR, Read_OK);
124
125     -------------------------------------
126     -- If line is not good or the first
127     -- character is not a TAB, then
128     -- skip remainder of loop is not good
129     -------------------------------------
130
131           IF((NOT Read_OK) OR (CHAR /=HT)) THEN NEXT;
132           END IF;
133
134     -------------------------------------
135     -- Read first stimulus bit
136     -- Read second stimulus bit
137     -- Read third stimulus bit
138     -------------------------------------
```

Figure 6.107: (Continued)

```
139
140         READ(Input_Pattern, A_In);
141         READ(Input_Pattern, CHAR);
142         READ(Input_Pattern, B_In);
143         READ(Input_Pattern, CHAR);
144         READ(Input_Pattern, C_In);
145
146     -------------------------------------
147     -- Apply test stimulus
148     -- Initially convert inputs (A, B, C)
149     -- as BIT to STD_LOGIC - only consider
150     -- logic '0' or logic '1'
151     -------------------------------------
152
153         IF    (A_In = '1') THEN  A <= '1';
154         ELSE  A <= '0';
155         END IF;
156
157         IF    (B_In = '1') THEN  B <= '1';
158         ELSE  B <= '0';
159         END IF;
160
161         IF    (C_In = '1') THEN  C <= '1';
162         ELSE  C <= '0';
163         END IF;
164
165     -------------------------------------
166     -- Wait for time before applying next
167     -- test stimulus
168     -------------------------------------
169
170          WAIT FOR 10 ns;
171
172     -------------------------------------
173     -- Convert 'Z' STD_LOGIC to 'Z_Out'
174     -- BIT - only consider logic '0' and
175     -- logic '1' and unknown 'X'
176     -------------------------------------
```

Figure 6.107: (Continued)

```
177
178         IF    (Z = '1') THEN  Z_Out := '1';
179         ELSE  Z_Out := '0';
180         END IF;
181
182     -------------------------------------
183     -- Write stimulus and output to output
185     -- file
185     -------------------------------------
186
187         WRITE(Results_Pattern, A_In);
188         WRITE(Results_Pattern, B_In);
189         WRITE(Results_Pattern, C_In);
190         WRITE(Results_Pattern, " ");
191         WRITE(Results_Pattern, Z_Out);
192
193         WRITELINE(Results_File, Results_Pattern);
194
195     -------------------------------------
196     -- End of Loop
197     -------------------------------------
198
199         END LOOP;
200
201     -------------------------------------
202     -- Write footer text to results file
203     -------------------------------------
204
205         WRITE(Results_Pattern, "----------------------------");
206         WRITELINE(Results_File, Results_Pattern);
207         WRITE(Results_Pattern, " -- Test completed");
208         WRITELINE(Results_File, Results_Pattern);
209         WRITE(Results_Pattern, "----------------------------");
210         WRITELINE(Results_File, Results_Pattern);
211
212     -------------------------------------
213     -- Close the OPENed files
214     -------------------------------------
```

Figure 6.107: (Continued)

```
215
216         FILE_CLOSE(Stimulus_File);
217         FILE_CLOSE(Results_File);
218
219     -------------------------------------
220     -- Stop process or it will repeat if
221     -- simulation time longer than a
222     -- single pass of the input and will
223     -- overwrite results file
224     -------------------------------------
225
226         WAIT;
227
228     -------------------------------------
229
230     END PROCESS;
231
232     -----------------------------------------------------
233
234     END ARCHITECTURE Behavioural;
235
236     ---------------------------------------------------------------
237     -- End of File
238     ---------------------------------------------------------------
```

Figure 6.107: (Continued)