Using Files in VHDL

This example demonstrates the usage of files in VHDL. Files are useful to store vectors that might be used to stimulate or drive test benches. Additionally, the output results can be recorded to a file. Their behavior is similar to how files work in other programming languages such as C. Note that **none of the file operator keywords described below produce synthesizable code.**. They are only intended for use in simulation test benches.

The package file that needs to be included to make File IO work correctly is std.textio. This allows the usage of the keywords: file_open, file_close, read, readline, write, writeline, flush, endfile, and others. One issue to be aware of is that **std.textio.all** only allows for reading and writing of type bit, bit_vector_boolean, character, integer, real, string, and time. This package file does not allow for reading and writing std_logic_vector type to a file. To allow for this, include the package file: **ieee.std_logic_textio.all** at the top of your design.

This example uses a 1-bit full-adder at the lowest level. The next highest level creates what is known as a ripple-carry adder. The test bench creates stimulus for the ripple-carry adder to exercise the logic and store the results to a file. In order to compile this code correctly, all three source files need to be in the same directory and compiled into Library Work. See the Modelsim tutorial for beginners for detail. You also need the input_vectors.txt and output_results.txt files in the same directory.

Testbench Code:

```
1  ----------------------------------------------------------------------------
2  -- File Downloaded from http://www.nandland.com
3  ----------------------------------------------------------------------------
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7  use STD.textio.all;
8  use ieee.std_logic_textio.all;
9
10 entity example_file_io_tb is
11
12 end example_file_io_tb;
13
14
15 architecture behave of example_file_io_tb is
16
17   ------------------------------------------------------------------------
18   -- Declare the Component Under Test
19   ------------------------------------------------------------------------
20   component module_ripple_carry_adder is
21     generic (
22       g_WIDTH : natural);
```

```
23      port (
24        i_add_term1 : in  std_logic_vector(g_WIDTH-1 downto 0);
25        i_add_term2 : in  std_logic_vector(g_WIDTH-1 downto 0);
26        o_result    : out std_logic_vector(g_WIDTH downto 0)
27        );
28    end component module_ripple_carry_adder;
29
30
31    ----------------------------------------------------------------------
32    -- Testbench Internal Signals
33    ----------------------------------------------------------------------
34    file file_VECTORS : text;
35    file file_RESULTS : text;
36
37    constant c_WIDTH : natural := 4;
38
39    signal r_ADD_TERM1 : std_logic_vector(c_WIDTH-1 downto 0) := (others => '0');
40    signal r_ADD_TERM2 : std_logic_vector(c_WIDTH-1 downto 0) := (others => '0');
41    signal w_SUM       : std_logic_vector(c_WIDTH downto 0);
42
43  begin
44
45    ----------------------------------------------------------------------
46    -- Instantiate and Map UUT
47    ----------------------------------------------------------------------
48    MODULE_RIPPLE_CARRY_ADDER_INST : module_ripple_carry_adder
49      generic map (
50        g_WIDTH => c_WIDTH)
51      port map (
52        i_add_term1 => r_ADD_TERM1,
53        i_add_term2 => r_ADD_TERM2,
54        o_result    => w_SUM
55        );
56
57
58    ----------------------------------------------------------------------
59    -- This procedure reads the file input_vectors.txt which is located in the
60    -- simulation project area.
61    -- It will read the data in and send it to the ripple-adder component
62    -- to perform the operations.  The result is written to the
63    -- output_results.txt file, located in the same directory.
64    ----------------------------------------------------------------------
65    process
66      variable v_ILINE     : line;
67      variable v_OLINE     : line;
68      variable v_ADD_TERM1 : std_logic_vector(c_WIDTH-1 downto 0);
69      variable v_ADD_TERM2 : std_logic_vector(c_WIDTH-1 downto 0);
70      variable v_SPACE     : character;
71
72    begin
73
74      file_open(file_VECTORS, "input_vectors.txt",  read_mode);
75      file_open(file_RESULTS, "output_results.txt", write_mode);
76
77      while not endfile(file_VECTORS) loop
78        readline(file_VECTORS, v_ILINE);
79        read(v_ILINE, v_ADD_TERM1);
80        read(v_ILINE, v_SPACE);           -- read in the space character
81        read(v_ILINE, v_ADD_TERM2);
82
83        -- Pass the variable to a signal to allow the ripple-carry to use it
84        r_ADD_TERM1 <= v_ADD_TERM1;
85        r_ADD_TERM2 <= v_ADD_TERM2;
86
87        wait for 60 ns;
88
89        write(v_OLINE, w_SUM, right, c_WIDTH);
90        writeline(file_RESULTS, v_OLINE);
91      end loop;
92
93      file_close(file_VECTORS);
94      file_close(file_RESULTS);
95
```

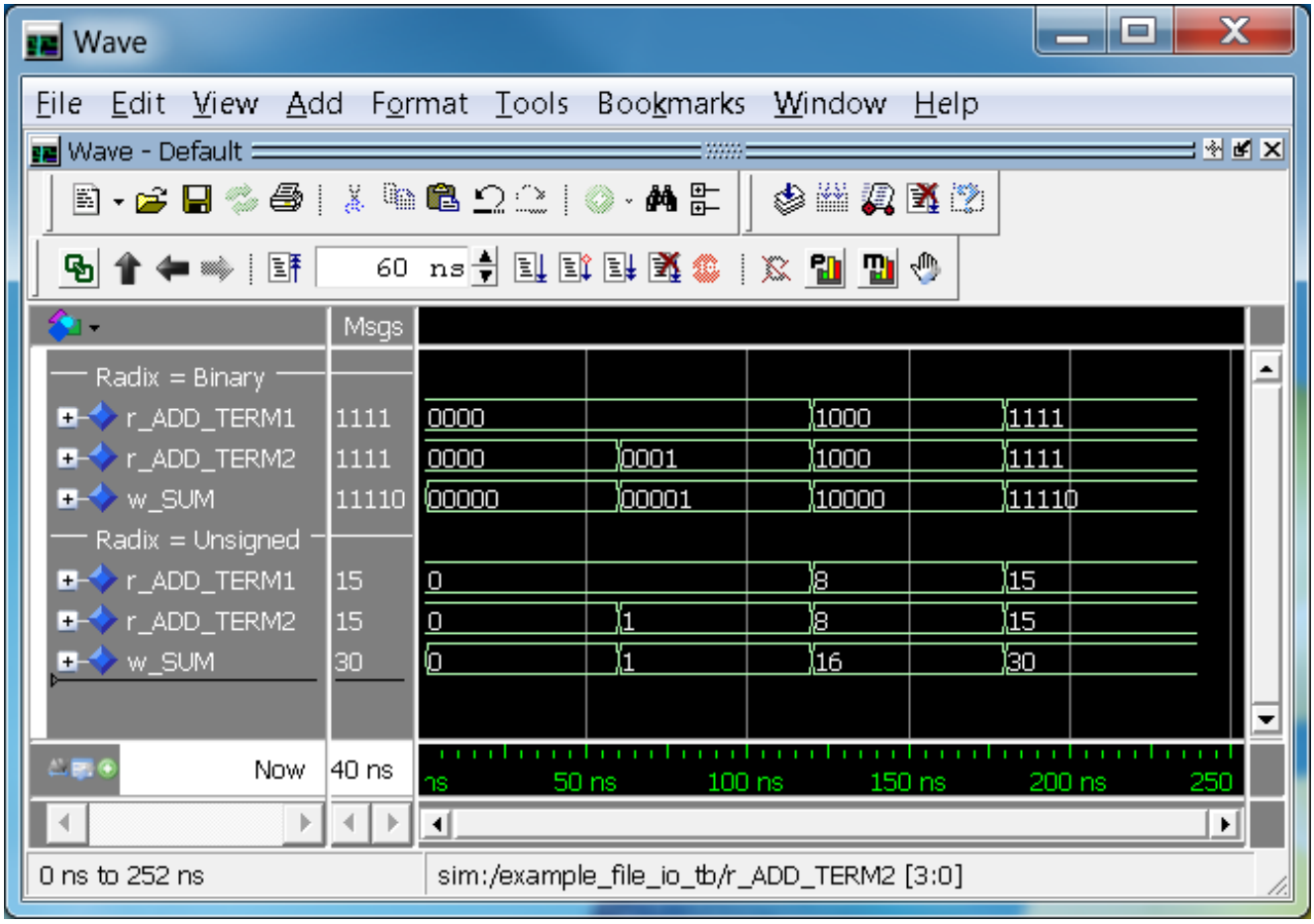
```
96      wait;
97    end process;
98
99  end behave;
```

input_vectors.txt:

```
0000 0000
0000 0001
1000 1000
1111 1111
```

output_results.txt:

```
00000
00001
10000
11110
```



Modelsim simulation wave output