
Advanced Testing using VHDL

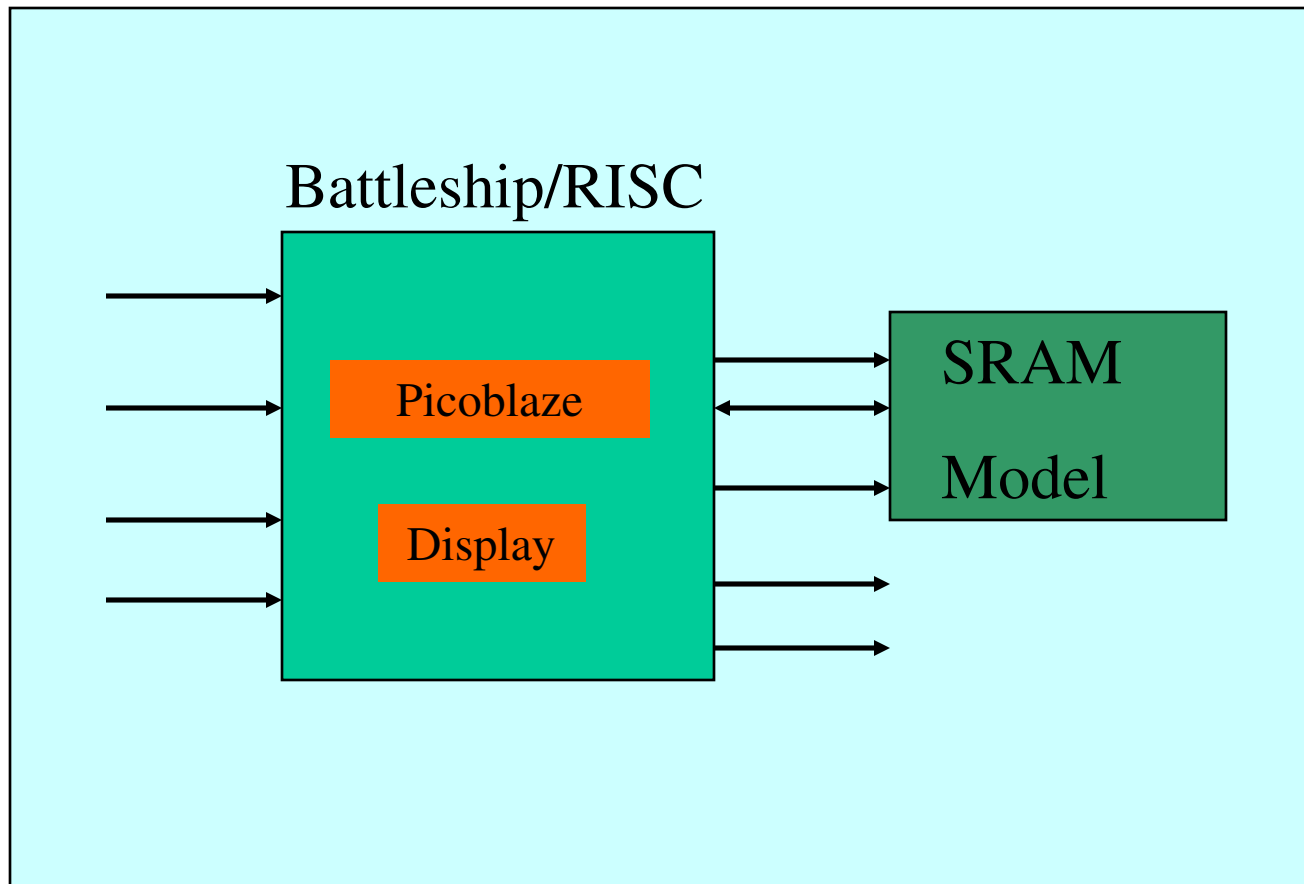
Module 9

Overview

- SRAM Model
- Attributes
- Loop Statements
- Test Bench examples using
 - TEXTIO
 - Conversion functions
 - Reading file containing test vectors
 - Writing test results to file

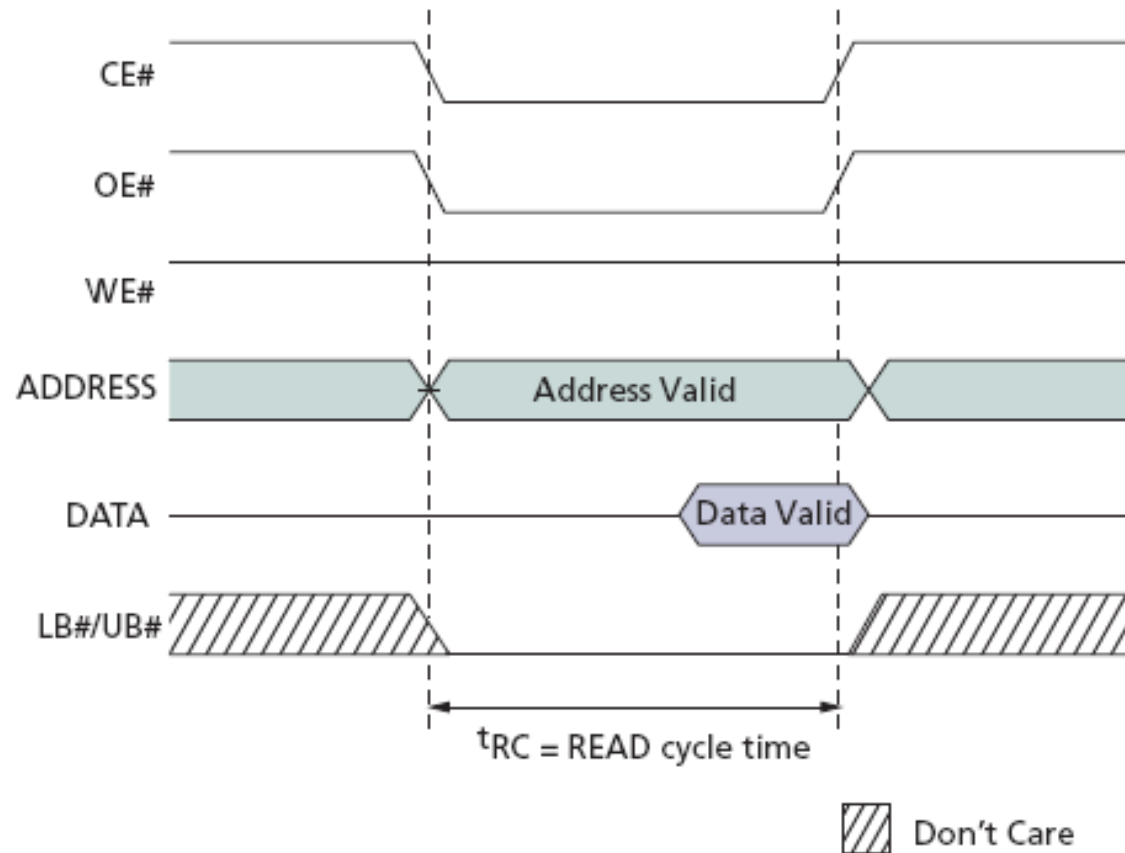
Adding the SRAM model

- New testbench



SRAM - Simplified Read Operation

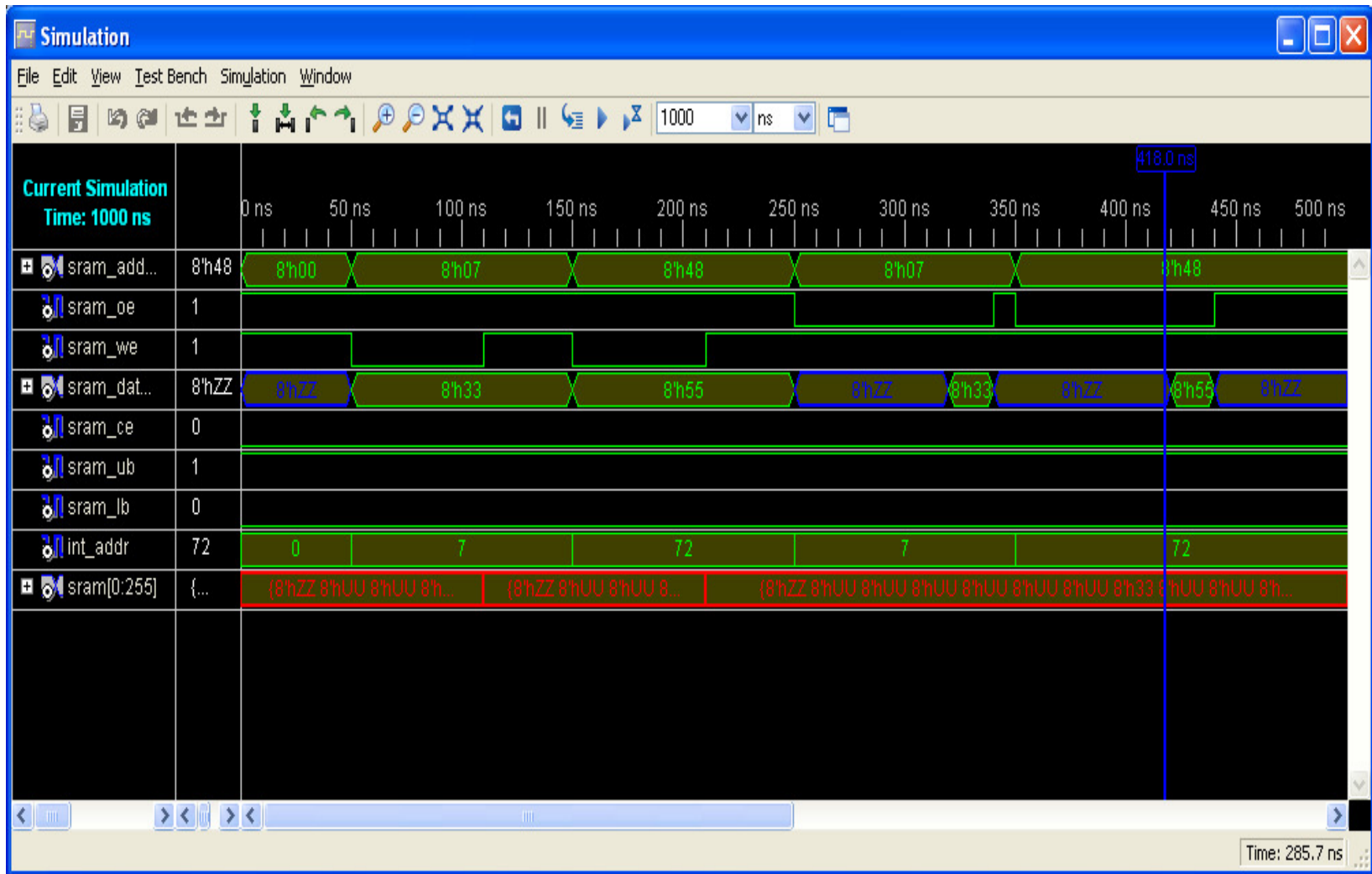
READ Operation (ADV# LOW)



SRAM Model - VHDL

```
sram_model.vhd
File Edit View Window
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 sram_model is
7 |    Port ( sram_addr : in std_logic_vector(7 downto 0);
8 |          sram_oe : in std_logic;
9 |          sram_we : in std_logic;
10 |         sram_data : inout std_logic_vector(7 downto 0);
11 |         sram_ce : in std_logic;
12 |         sram_ub : in std_logic;
13 |         sram_lb : in std_logic);
14 |end sram_model;
15 |
16 |architecture Behavioral of sram_model is
17 |    type memory is array(0 to 255) of std_logic_vector(7 downto 0);
18 |    signal sram : memory;
19 |    signal int_addr : integer range 0 to 255;
20 |begin
21 |
22 |    int_addr <= conv_integer(sram_addr);
23 |    -- read from sram
24 |    sram_data <= sram(int_addr) after 70 ns when sram_ce = '0' and sram_oe = '0' else
25 |        "ZZZZZZZZ";
26 |
27 |    -- write to sram on rising edge of WE
28 |    process(sram_we)
29 |    begin
30 |        if sram_we'event and sram_we = '1' then
31 |            if sram_ce = '0' then
32 |                sram(int_addr) <= sram_data;
33 |            end if;
34 |        end if;
35 |    end process;
36 |
37 |end Behavioral;
38 |
[CAPS] [NUM] [SCRL] [LOC] [VHDL] ...
```

SRAM Model Read and Write



VHDL Attributes

- Signals can have attributes associated with them
- Example predefined signal attributes are
 - function
 - value
 - type
 - range
- Also possible to have user-defined attributes

Function Attributes

- Predefined functions
- Returns information about the behavior of signals
 - s'EVENT
 - returns true if an event occurred
 - change in value
 - s'ACTIVE
 - returns true if signal s is active
 - new value assigned to signal s (may be same value)
 - s'LAST_EVENT
 - returns elapsed time since last event
 - s'LAST_ACTIVE
 - s'LAST_VALUE
 - returns value of s before the last event

Function Attribute example

- Check for setup time violation on d input changing

```
CONSTANT setup_time : TIME := 12 ns; -- TIME is predefined

PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        ASSERT(d'LAST_EVENT >= setup_time)
            REPORT "setup violation"
            SEVERITY error;
    END IF;
```

- Assert statement checks that the input d has not had an event during the setup time.
- If time returned is less than setup time - assertion will fail

'now' function

- NOW

- predefined function that returns simulation time

- `IF d'EVENT THEN`

- `lasteventonD := NOW;`

- `IF clk'EVENT AND clk = '1' THEN`

- `ASSERT(now - lasteventonD) >= setup_time`

- `REPORT "setup violation"`

- `SEVERITY error`

RANGE attributes

- Only for constrained array types
- Returns range of specified type
- Two range attributes
 - a'RANGE
 - a'REVERSE RANGE

```
TYPE address_bus IS ARRAY (63 DOWNT0 0) OF std_logic;
```

```
SIGNAL cpu_address : address_bus; -- declare array
```

```
FOR j IN cpu_address'RANGE LOOP -- 63 DOWNT0 0
```

```
FOR j IN cpu_address'REVERSE_RANGE LOOP -- 0 TO 63
```

VALUE and ARRAY Attributes

- These return the bounds of a type
- Four predefined attributes
 - a'LEFT
 - returns left bound of type
 - a'RIGHT
 - returns right bound of type
 - a'HIGH
 - returns upper bound of type
 - a'LOW
 - returns lower bound of type
- Also
 - a'LENGTH
 - returns total length of the array

Value and Array Attribute examples

```
TYPE address_bus IS ARRAY (63 DOWNT0 0) OF std_logic;
SIGNAL cpu_address      : address_bus; -- declare array
BEGIN

PROCESS
VARIABLE i, j, k, l, m : INTEGER;
BEGIN
    i := cpu_address'LEFT;      -- 63
    j := cpu_address'RIGHT;    -- 0
    k := cpu_address'HIGH;     -- 63
    l := cpu_address'LOW;      -- 0
    m := cpu_address'LENGTH;   -- 64
END PROCESS;
```

Enumerated type example

```
TYPE days IS (mon, tues, wed, thurs, fri, sat, sun);
SUBTYPE weekend IS days RANGE sat TO sun;
SIGNAL day1, day2, day3, day4, day5 : days;
SIGNAL count : INTEGER;

BEGIN

PROCESS
BEGIN
    day1 <= days'LEFT;      -- mon
    day2 <= days'RIGHT;    -- sun
    day3 <= days'HIGH;     -- sun
    day4 <= weekend'LOW;    -- sat

    count <= days'POS(tues); -- 1
    day5 <= days'VAL(3));  -- thurs
END PROCESS;
```

LOOP Statements

- Used to iterate through a set of sequential statements
- Three types

```
FOR identifier IN range LOOP
```

```
END LOOP;
```

```
WHILE boolean_expression LOOP
```

```
END LOOP;
```

```
LOOP
```

```
    EXIT WHEN condition_test
```

```
END LOOP;
```


FOR LOOP

- **general syntax**

```
FOR identifier IN range LOOP
```

```
END LOOP;
```

- **example:**

```
factorial := 1;
```

```
FOR number IN 2 TO n LOOP
```

```
    factorial := factorial * number;
```

```
END LOOP;
```

- number = 2, 3, 4...
- no explicit declaration for loop identifier required
- also DOWNTO

WHILE Loop

- general syntax

```
WHILE boolean_expression
```

```
END LOOP;
```

- example:

```
j := 0;
```

```
sum := 10;
```

```
wh_loop: WHILE j < 20 LOOP
```

```
    sum := sum * 2
```

```
    j := j + 3;
```

```
END LOOP wh_loop;
```

- Note: optional label for loop statement

LOOP

- No iteration scheme
 - statements in body executed repeatedly until loop exits

- General syntax

```
LOOP
    EXIT WHEN boolean_expression; -- optional EXIT statement
END LOOP;
```

- Example:

```
j := 0;
sum := 1;
12: LOOP
    sum := sum * 10
    j := j + 17;
    EXIT WHEN sum > 100;
END LOOP 12;
```

- If exit statement not present loop executes indefinitely
-

LOOP cont'd

- EXIT statement
 - only used inside a loop

- General syntax

```
EXIT [loop_label] [WHEN condition]
    EXIT WHEN boolean_expression; -- optional EXIT statement
END LOOP;
```

- Example (alternative to previous example):

```
IF sum > 100 THEN
    EXIT;
END IF;
```

LOOP cont'd

- NEXT statement
 - used to exit a loop for the current iteration of a loop
 - continue to the next iteration
- General syntax

```
NEXT [loop_label] [WHEN condition]
```

LOOP example

```
PROCESS
BEGIN
l1: LOOP
    -- statements
    --
    l2: LOOP
        -- statements
        --
        test_num := test_num + 1;
        EXIT l2 WHEN test_num = 25;
        IF solenoid_1 = '1' THEN
            drive_b := '0';
            NEXT l1; -- go to top of loop1
        END IF;
        IF trigger = '0' THEN
            -- statements
        END LOOP l2;
        EXIT l1 WHEN sim_tests = 200;
    END LOOP l1;
```

Test Bench example

- example test bench:

```
-- Test Bench to exercise and verify correctness of DECODE entity
ENTITY tb2_decode IS
END tb2_decode;

ARCHITECTURE test_bench OF tb2_decode IS
    TYPE input_array IS ARRAY (0 TO 3) OF std_logic_vector(1 DOWNTO 0);
    CONSTANT input_vectors: input_array :=
        ("00", "01", "10", "11");
    -- input_vectors array contains test vectors for input
    SIGNAL in1      : STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL out1     : STD_LOGIC_VECTOR (3 DOWNTO 0);

    COMPONENT decode
        PORT (
            sel : IN std_logic_vector(1 downto 0);
            y   : OUT std_logic_vector(3 downto 0));
    END COMPONENT;

```

Test Bench example (cont'd)

```
BEGIN
  decode_1: decode PORT MAP(sel => in1, y => out1);

  apply_inputs: PROCESS
  BEGIN
    FOR j IN input_vectors'RANGE LOOP
      in1 <= input_vectors(j);
      WAIT FOR 50 ns;
    END LOOP;
  END PROCESS apply_inputs;

  test_outputs: PROCESS
  BEGIN
    WAIT UNTIL (in1 = "01");
    WAIT FOR 25 ns;
    ASSERT (out1 = "0110")
      REPORT "Output not equal to 0110"
      SEVERITY ERROR;
  END PROCESS test_outputs;
END test_bench;
```

Assert Statements

- During testing
 - Usually want to display information about signals and variables
- Assert statement is rather limited
 - Report clause only allows a single string
 - no built-in provision for formatting data

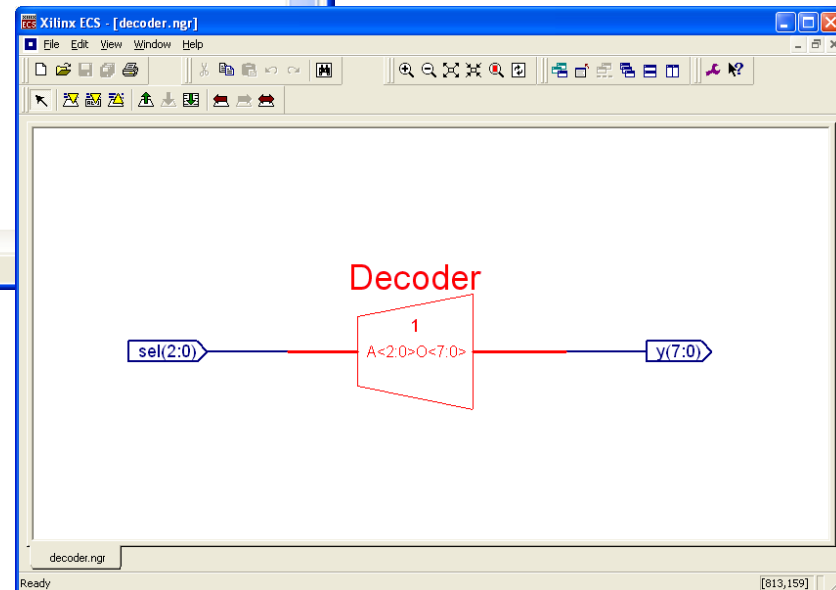
```
ASSERT FALSE  
  REPORT "first line" & CR & "second line";
```

Test Bench example

- Adapted from Pellerin and Taylor (pages 242-246)
 - demonstrates use of
 - textio to read in a test vector file
 - complex strings in Assert Statements
 - string to vector and vector to string conversion functions
 - uses a decoder component for test

VHDL for Synthesis

```
decoder.vhd * - ISE Text Editor
File Edit
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 decoder is
7     Port ( sel : in std_logic_vector(2 downto 0);
8           y : out std_logic_vector(7 downto 0));
9 end decoder;
10
11 architecture Behavioral of decoder is
12
13 begin
14     y <= "00000001" when sel = "000" else
15         "00000010" when sel = "001" else
16         "00000100" when sel = "010" else
17         "00001000" when sel = "011" else
18         "00010000" when sel = "100" else
19         "00100000" when sel = "101" else
20         "01000000" when sel = "110" else
21         "10000000";
22
23 end Behavioral;
24
For Help, press F1
```



test_vec.txt File

- Format is
 - 3 bits for SEL input
 - 8 bits for expected output

```
0000000001  
0010000010  
01000000100  
01100001000  
10000010000  
10100110000  
11001000000  
11110000000
```

Test Bench VHDL file

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE std.textio.ALL;          -- use text I/O features of standard library

ENTITY test_bench IS
END test_bench;

ARCHITECTURE tb1 OF test_bench IS
    COMPONENT decoder        -- component to be tested
        PORT(sel : IN std_logic_vector(2 DOWNTO 0);
              y  : OUT std_logic_vector(7 DOWNTO 0));
    END COMPONENT;

    -- FOR all: decoder USE ENTITY work.decoder;    -- configuration
```

Test Bench VHDL file (cont'd)

```
-- function to convert string of character to vector
FUNCTION str2vec(str : string) RETURN std_logic_vector IS
    VARIABLE vtmp: std_logic_vector(str'RANGE);
BEGIN
    FOR i IN str'RANGE LOOP
        IF str(i) = '1' THEN
            vtmp(i) := '1';
        ELSIF str(i) = '0' THEN
            vtmp(i) := '0';
        ELSE
            vtmp(i) := 'X';
        END IF;
    END LOOP;
    RETURN vtmp;
END str2vec;
```

Test Bench VHDL file (cont'd)

```
-- function to convert vector to string
-- for use in assert statements
FUNCTION vec2str(vec : std_logic_vector) RETURN string IS
    VARIABLE stmp : string(vec'LEFT+1 DOWNTO 1);
BEGIN
    FOR i IN vec'REVERSE_RANGE LOOP
        IF vec(i) = '1' THEN
            stmp(i+1) := '1';
        ELSIF vec(i) = '0' THEN
            stmp(i+1) := '0';
        ELSE
            stmp(i+1) := 'X';
        END IF;
    END LOOP;
    RETURN stmp;
END vec2str;
```

Test Bench VHDL file (cont'd)

```
SIGNAL clk : std_logic := '0';

-- create internal signals to connect to test component
SIGNAL sel : std_logic_vector(2 DOWNTO 0);
SIGNAL y : std_logic_vector(7 DOWNTO 0);

BEGIN

-- instantiate decoder test component
u1: decoder PORT MAP(sel => sel, y => y);

clk <= NOT clk AFTER 50 ns;    -- create clock for timing
```


Test Bench VHDL file (cont'd)

```
PROCESS
    -- declare and open file (1987 style)
    FILE vector_file: text IS in "test.vec";
    VARIABLE file_line : line; -- text line buffer
    VARIABLE str_stimulus_in: string(11 DOWNTO 1);
    VARIABLE stimulus_in : std_logic_vector(10 DOWNTO 0);
    VARIABLE y_expected : std_logic_vector(7 DOWNTO 0);
BEGIN
    -- loop through lines in test file
    WHILE NOT endfile(vector_file) LOOP

        -- read one complete line into file_line
        readline(vector_file, file_line);

        -- extract the first field from file_line
        read(file_line, str_stimulus_in);

        -- convert string to vector
        stimulus_in := str2vec(str_stimulus_in);
```

Test Bench VHDL file (cont'd)

```
WAIT UNTIL clk = '1';          -- rising edge of clock
-- now get sel input and apply to decoder
sel <= stimulus_in(10 DOWNT0 8);      -- top 3 bits for input;

WAIT UNTIL clk = '0';          -- falling edge
-- now check if decoder outputs are correct
y_expected := stimulus_in(7 DOWNT0 0); -- expected output

-- compare decoder output with expected value
IF y /= y_expected THEN
    ASSERT false
        REPORT "decoder failure" & CR &
            "Expected y to be " & vec2str(y_expected) &
            " but its value was " & vec2str(y)
        SEVERITY ERROR;
END IF;
END LOOP;

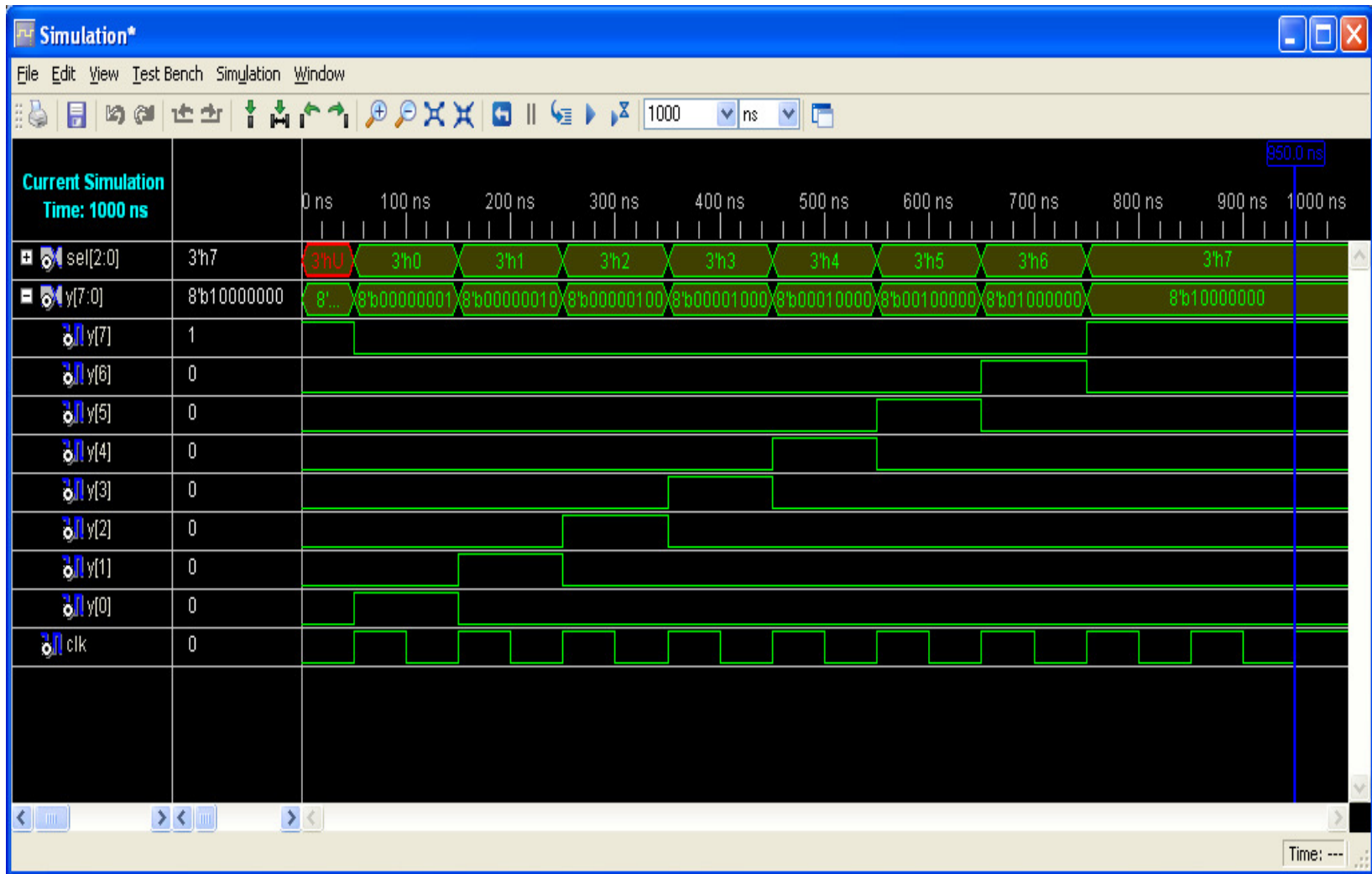
WAIT;          -- suspend the simulation
END PROCESS;
END tb1;
```

VHDL Test Bench - Results

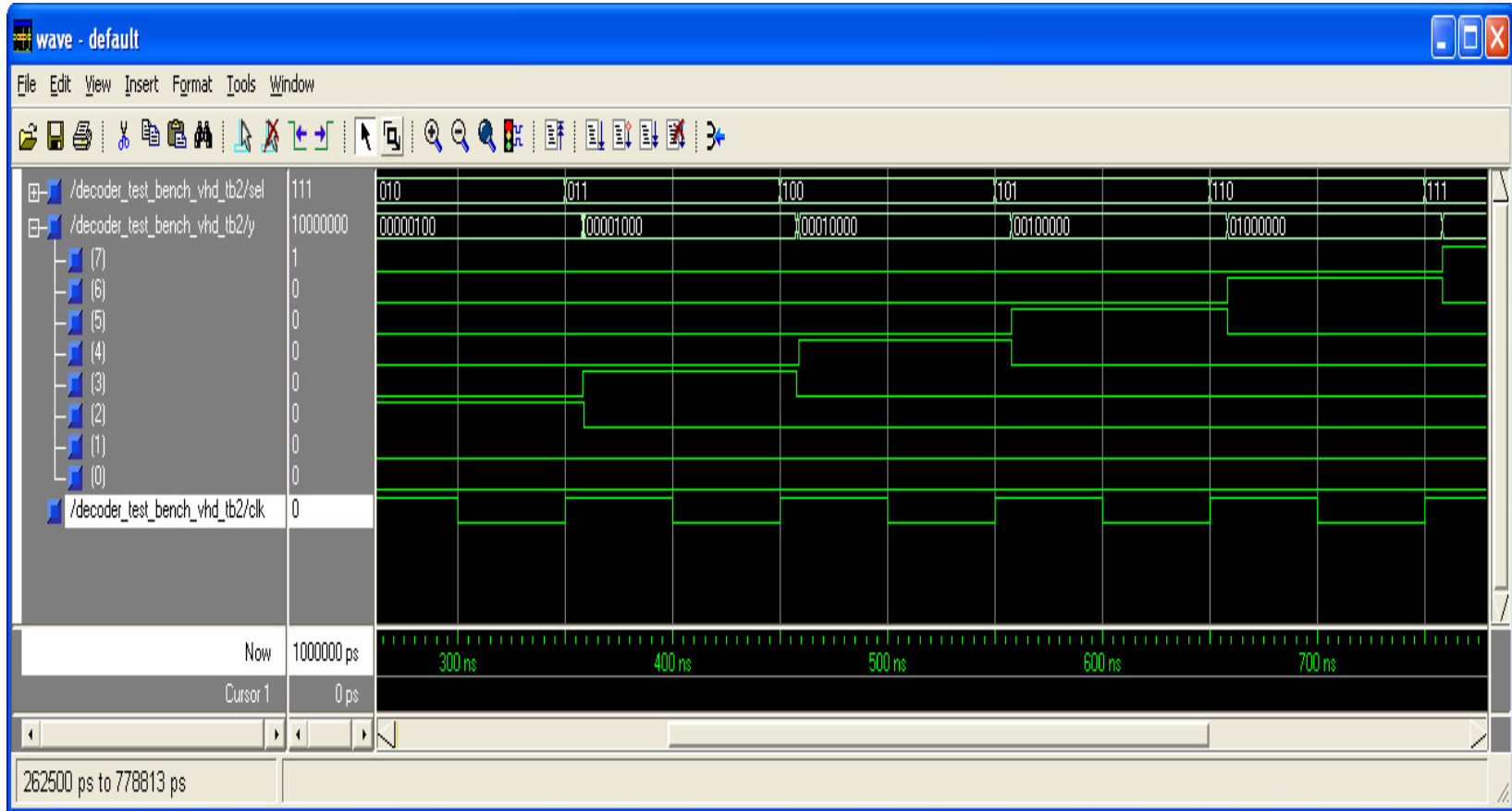
```
71  clk <= NOT clk AFTER 50 ns; -- create clock for timing
72
73
74  PROCESS
75  -- declare and open file (1993 style)
76  FILE vector_file: text OPEN read_mode IS "test_vec.txt";
77  VARIABLE file_line : line; -- text line buffer
78  VARIABLE str_stimulus_in: string(11 DOWNTO 1);
79  VARIABLE stimulus_in : std_logic_vector(10 DOWNTO 0);
80  VARIABLE y_expected : std_logic_vector(7 DOWNTO 0);
81  BEGIN
82  -- loop through lines in test file
83  WHILE NOT endfile(vector_file) LOOP
84
85
86      -- read one complete line into file_line
87      readline(vector_file, file_line);
88      -- extract the first field from file_line
89      read(file_line, str_stimulus_in);
90
91      -- convert string to vector
92      stimulus_in := str2vec(str_stimulus_in);
93
94      WAIT UNTIL clk = '1'; -- rising edge of clock
95      -- now get sel input and apply to decoder
96      sel <= stimulus_in(10 DOWNTO 8); -- top 3 bits for input;
97
98      WAIT UNTIL clk = '0'; -- falling edge
99      -- now check if decoder outputs are correct
100     y_expected := stimulus_in(7 DOWNTO 0); -- expected output
101
102     -- compare decoder output with expected value
103     IF y /= y_expected THEN
104         ASSERT false
105             REPORT "decoder failure" & CR & LF &
106                 "Expected y to be " & vec2str(y_expected) &
107                 " but its value was " & vec2str(y)
108             SEVERITY ERROR;
109     END IF;
110     END LOOP;
111
112     WAIT; -- suspend the simulation
113 END PROCESS;
```

This is a Full version of ISE Simulator (ISim).
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 600,000 ns: Error: decoder failure
Expected y to be 00110000 but its value was 00100000
%

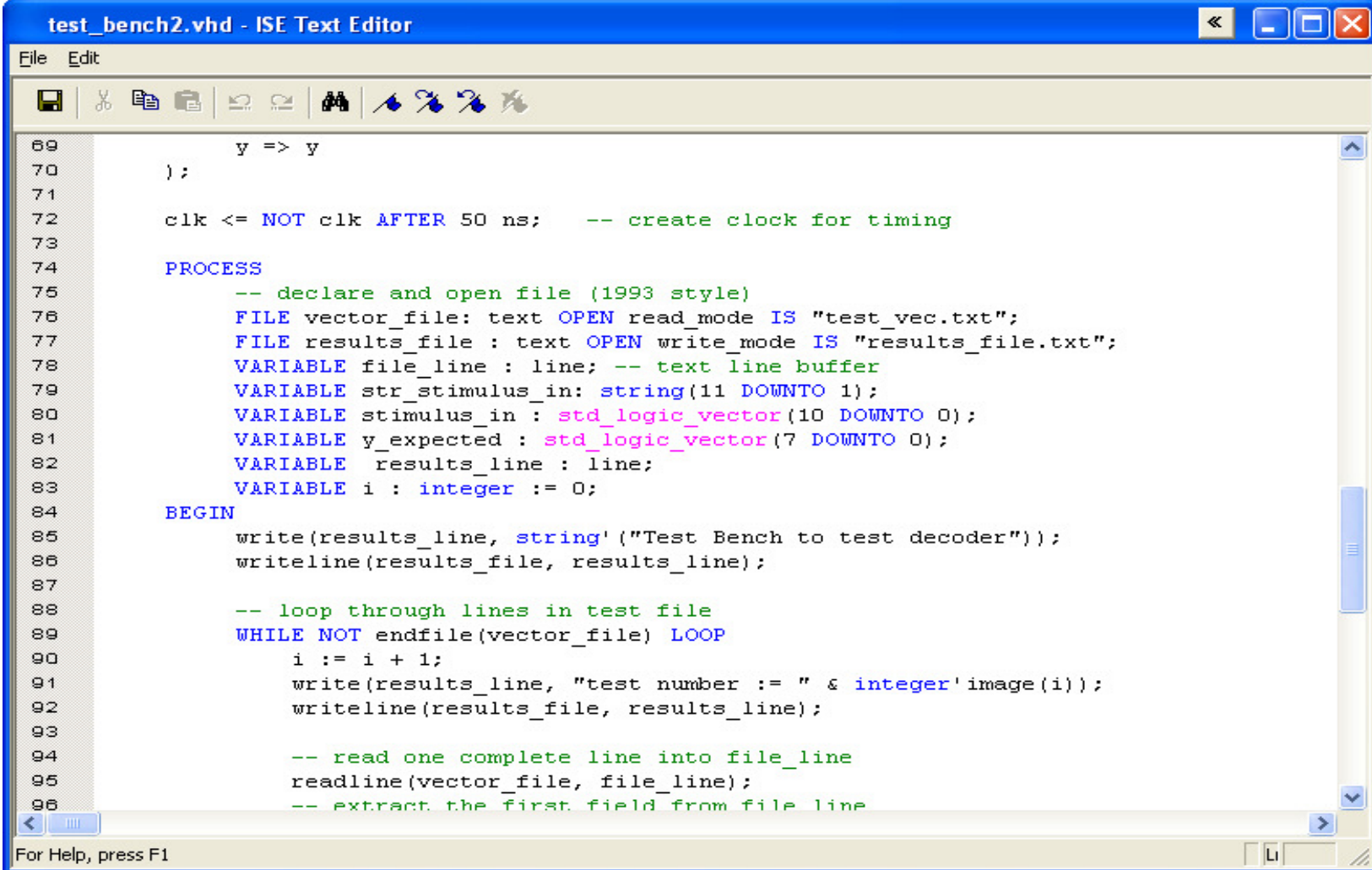
Test Bench Waveform



Simulation of post-place and route



Writing Test Results to File

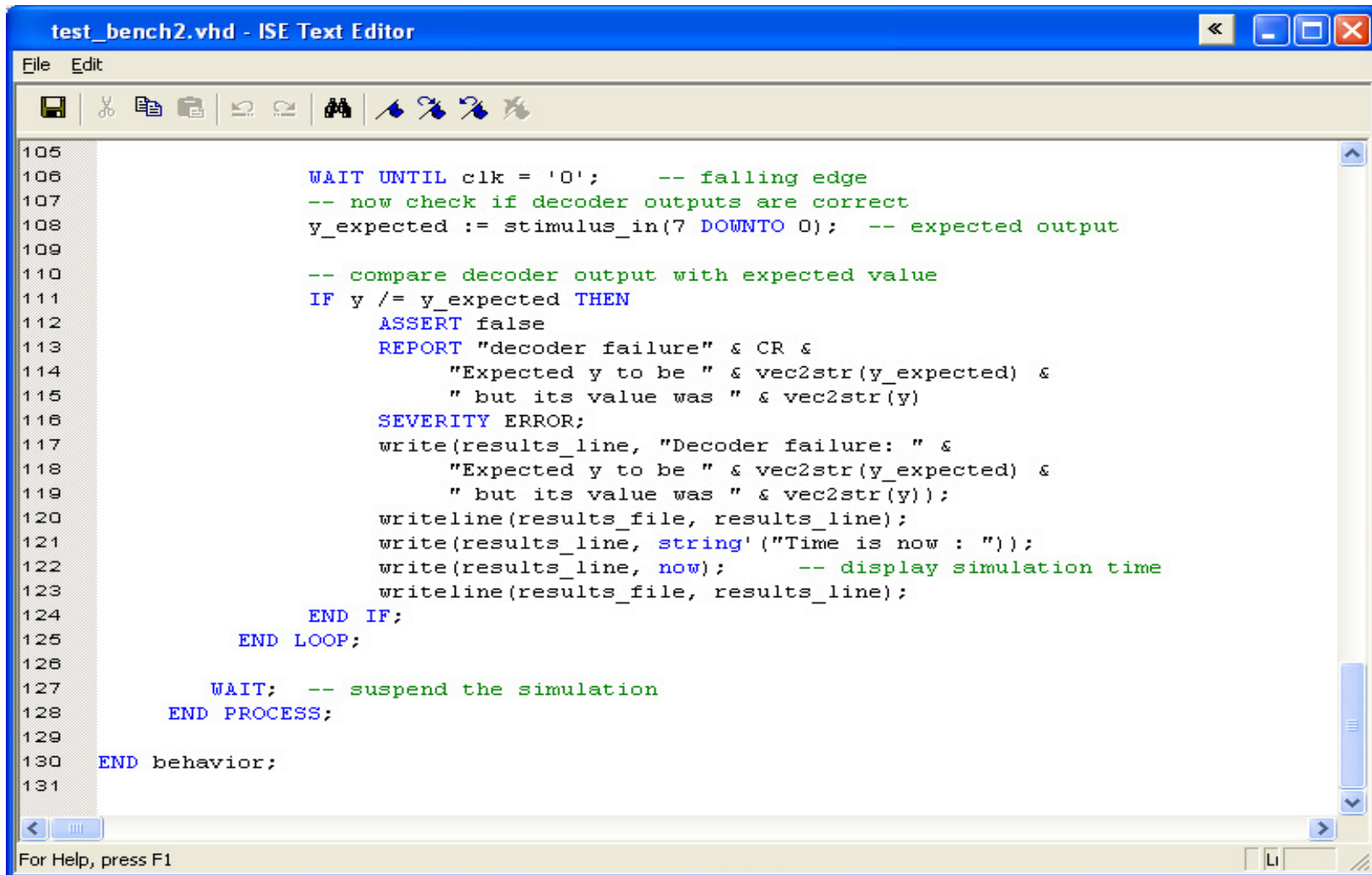


```
test_bench2.vhd - ISE Text Editor
File Edit
[Icons]
69     y => y
70   );
71
72   clk <= NOT clk AFTER 50 ns;  -- create clock for timing
73
74   PROCESS
75     -- declare and open file (1993 style)
76     FILE vector_file: text OPEN read_mode IS "test_vec.txt";
77     FILE results_file : text OPEN write_mode IS "results_file.txt";
78     VARIABLE file_line : line; -- text line buffer
79     VARIABLE str_stimulus_in: string(11 DOWNT0 1);
80     VARIABLE stimulus_in : std_logic_vector(10 DOWNT0 0);
81     VARIABLE y_expected : std_logic_vector(7 DOWNT0 0);
82     VARIABLE results_line : line;
83     VARIABLE i : integer := 0;
84   BEGIN
85     write(results_line, string'("Test Bench to test decoder"));
86     writeline(results_file, results_line);
87
88     -- loop through lines in test file
89     WHILE NOT endfile(vector_file) LOOP
90       i := i + 1;
91       write(results_line, "test number := " & integer'image(i));
92       writeline(results_file, results_line);
93
94       -- read one complete line into file_line
95       readline(vector_file, file_line);
96       -- extract the first field from file_line

```

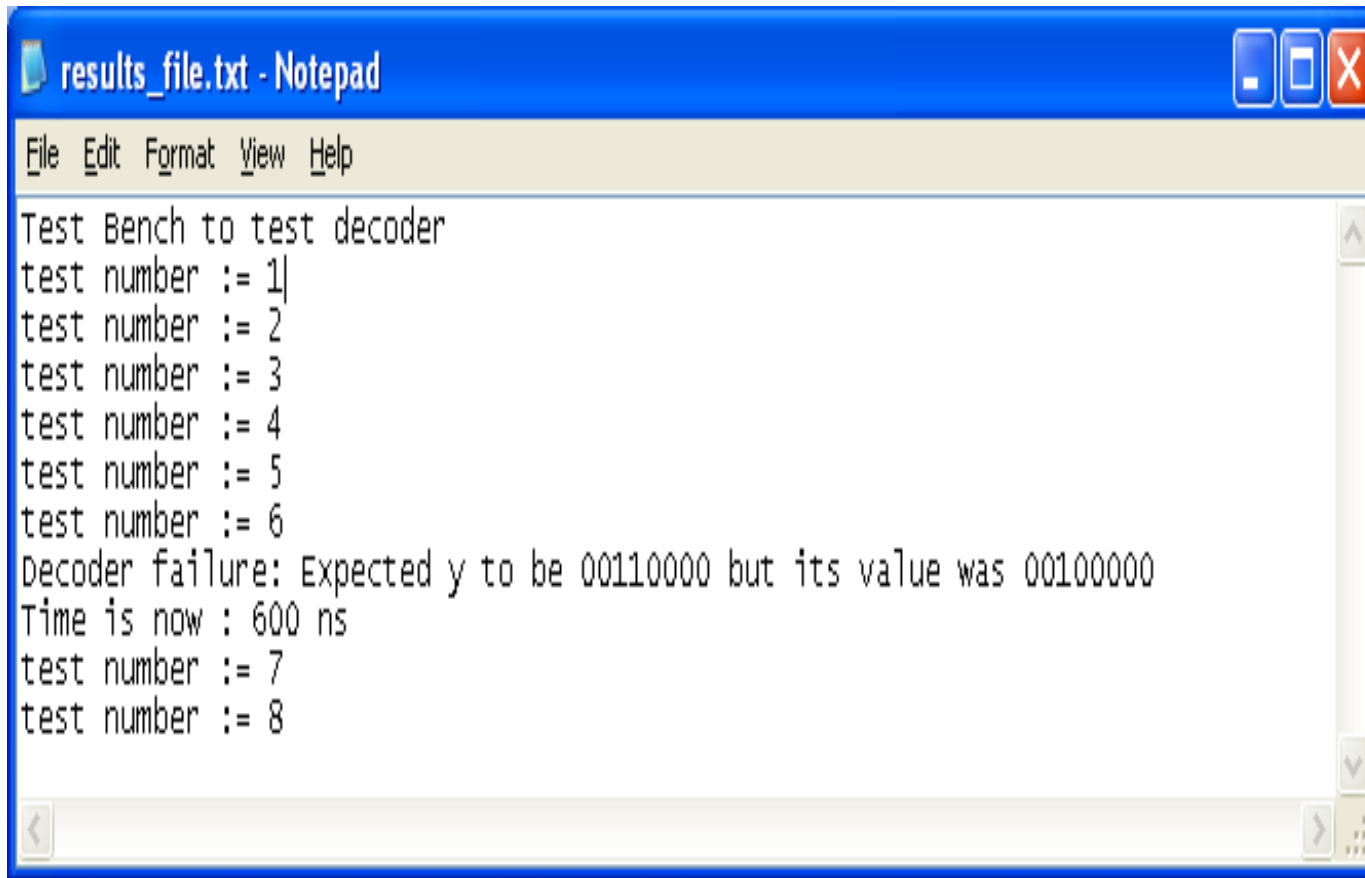
For Help, press F1

Writing Test Results to File (cont'd)



```
test_bench2.vhd - ISE Text Editor
File Edit
105
106     WAIT UNTIL clk = '0';    -- falling edge
107     -- now check if decoder outputs are correct
108     y_expected := stimulus_in(7 DOWNTO 0); -- expected output
109
110     -- compare decoder output with expected value
111     IF y /= y_expected THEN
112         ASSERT false
113         REPORT "decoder failure" & CR &
114             "Expected y to be " & vec2str(y_expected) &
115             " but its value was " & vec2str(y)
116         SEVERITY ERROR;
117         write(results_line, "Decoder failure: " &
118             "Expected y to be " & vec2str(y_expected) &
119             " but its value was " & vec2str(y));
120         writeline(results_file, results_line);
121         write(results_line, string'("Time is now : "));
122         write(results_line, now);    -- display simulation time
123         writeline(results_file, results_line);
124     END IF;
125 END LOOP;
126
127     WAIT; -- suspend the simulation
128 END PROCESS;
129
130 END behavior;
131
For Help, press F1
```

results_file



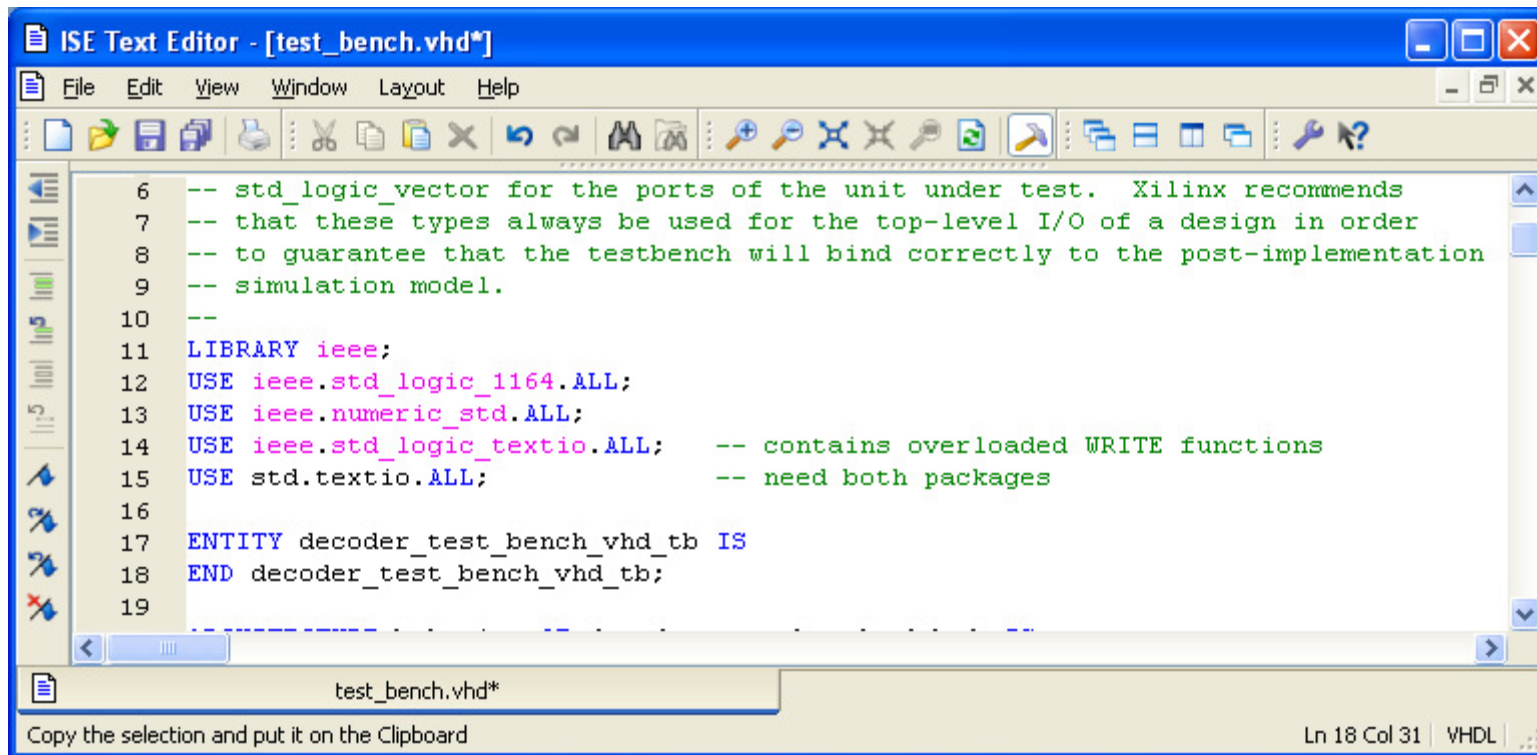
```
File Edit Format View Help
Test Bench to test decoder
test number := 1
test number := 2
test number := 3
test number := 4
test number := 5
test number := 6
Decoder failure: Expected y to be 00110000 but its value was 00100000
Time is now : 600 ns
test number := 7
test number := 8
```


Assert Statement Limitations

- Assert statements are limited to strings only
 - need conversion functions for displaying signals
- Use TEXTIO instead of Assert Statements to print messages
 - supports most data types
 - allows writing to multiple files

Improving output messages

- Use ieee.std_logic_textio.ALL package
- Removes need for conversion functions

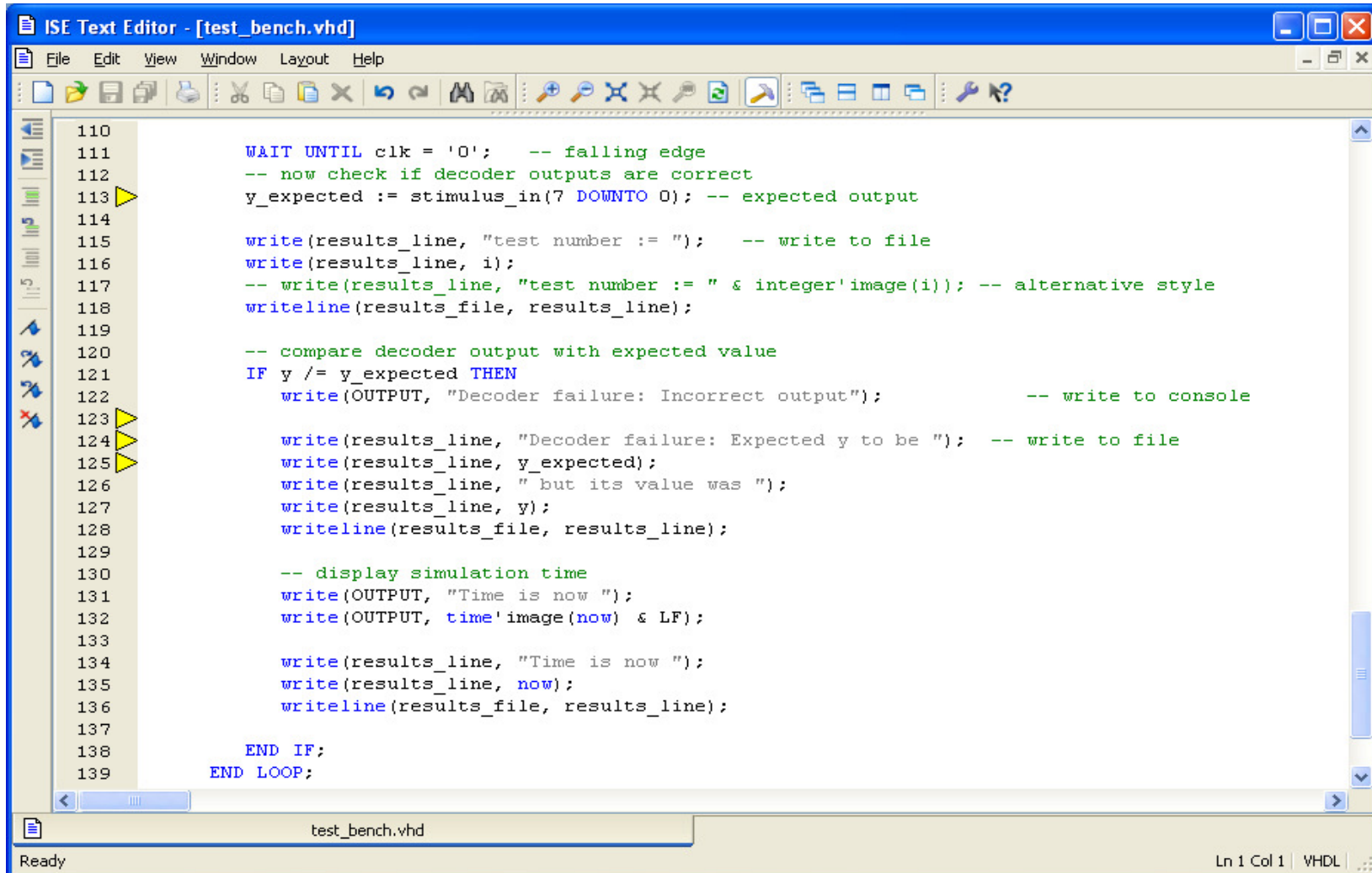


The screenshot shows the ISE Text Editor window for a file named [test_bench.vhd*]. The code is as follows:

```
6  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
7  -- that these types always be used for the top-level I/O of a design in order
8  -- to guarantee that the testbench will bind correctly to the post-implementation
9  -- simulation model.
10 --
11 LIBRARY ieee;
12 USE ieee.std_logic_1164.ALL;
13 USE ieee.numeric_std.ALL;
14 USE ieee.std_logic_textio.ALL;  -- contains overloaded WRITE functions
15 USE std.textio.ALL;             -- need both packages
16
17 ENTITY decoder_test_bench_vhd_tb IS
18 END decoder_test_bench_vhd_tb;
19
```

The status bar at the bottom indicates the current position is Ln 18 Col 31 in VHDL mode.

Modified write statements



```
110
111     WAIT UNTIL clk = '0'; -- falling edge
112     -- now check if decoder outputs are correct
113     y_expected := stimulus_in(7 DOWNT0 0); -- expected output
114
115     write(results_line, "test number := "); -- write to file
116     write(results_line, i);
117     -- write(results_line, "test number := " & integer'image(i)); -- alternative style
118     writeline(results_file, results_line);
119
120     -- compare decoder output with expected value
121     IF y /= y_expected THEN
122         write(OUTPUT, "Decoder failure: Incorrect output"); -- write to console
123
124         write(results_line, "Decoder failure: Expected y to be "); -- write to file
125         write(results_line, y_expected);
126         write(results_line, " but its value was ");
127         write(results_line, y);
128         writeline(results_file, results_line);
129
130         -- display simulation time
131         write(OUTPUT, "Time is now ");
132         write(OUTPUT, time'image(now) & LF);
133
134         write(results_line, "Time is now ");
135         write(results_line, now);
136         writeline(results_file, results_line);
137
138     END IF;
139 END LOOP;
```

test_bench.vhd

Ready Ln 1 Col 1 VHDL