

How to use textio library functions in VHDL

Created by: David Rubio G.

Blog post: <https://soceame.wordpress.com/2025/03/04/how-to-use-textio-library-functions-in-vhdl/>

Blog: <https://soceame.wordpress.com/>

GitHub: <https://github.com/DRubioG>

Last modification date: 04/03/25

tion signals is quite complex. Well, there is a library in VHDL that allows you to do simulations with data recorded in a file and record the results in a file. This library is **std.textio**.

```
use std.textio.all;
```

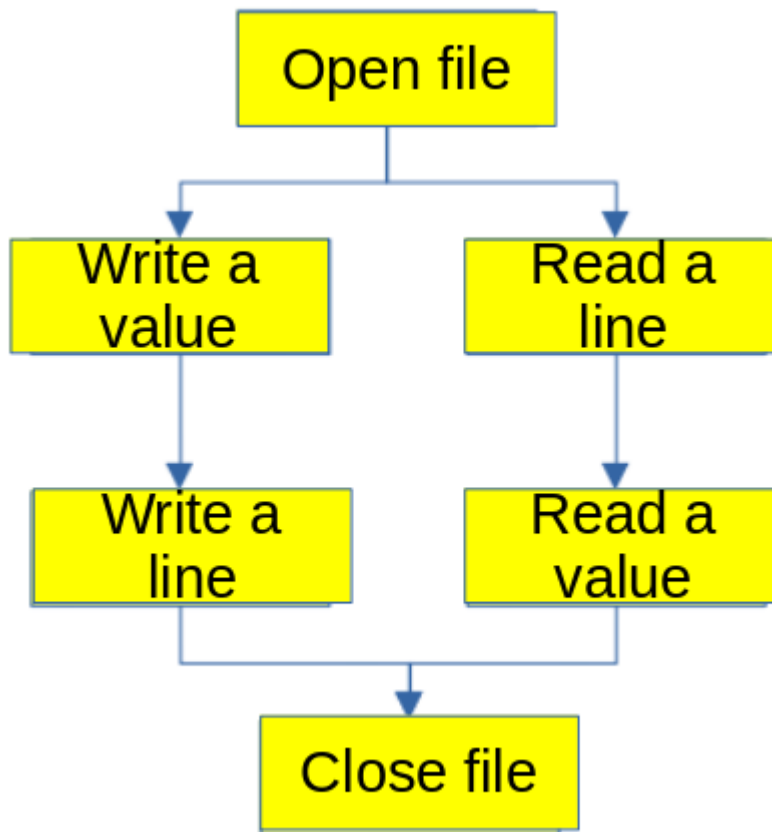
This is the base library for reading and writing files. This library has the disadvantage that it does not allow writing `std_logic_vector` data to files, they must be converted to **integer** type.

To be able to use `std_logic_vector` there is the **std_logic_textio** library.

```
use ieee.std_logic_textio.all;
```

Both libraries can be used simultaneously (this is the most recommended)

The architecture for working with this library is the following:



It is based on reading or writing value by value or reading each file line by line.

Clarification: in order to read data from a file, you must first open the file, extract the data it contains, and then read the file line by line.

NOTE: You cannot read and write the same file, so you must use two different files.

Reading

Opening a file: to open a file there are two options,

- Use the «*file_open*» function, with a *text* type «*input*» file, with the name of the file (.txt or .dat is recommended. And finally the file mode: **read_mode**, for reading, **write_mode**, for writing.

```
file input : text;
begin
...
file_open(input, "<file name>", <mode>);
```

- Declare everything as a *file* along with the constants and signals.

```
file input : text open <mode> is "<file name>";
```

Reading a line from a file: To read a line from a file, use the *readline(input, line)* function. This function has two parameters: the file type created to open the file and the process variable line type. An example:

```
file input : text open read_mode is "Input_data.dat";
begin
...
process
variable linea : line;
begin

readline(input , linea); -- We read the line from the file and dump it into
"linea"
end process;
```

Reading data from a line: To read data from a line, the *read(line, data)* function is used. This function reads data sequentially, so each time it is used on a line, it reads new data. Example:

```
file input : text open read_mode is "Input_data.dat";
signal dato1, dato2, dato3 : std_logic_vector(3 downto 0);
begin
...
process
variable linea : line;
begin

readline(input , linea);
read(linea, dato1); -- every time that read, it reads a new
read(linea, dato2); -- data and assign to datoX signal
read(linea, dato3);

end process;
```

There are other ways to read data in other formats, such as hexadecimal or octal:

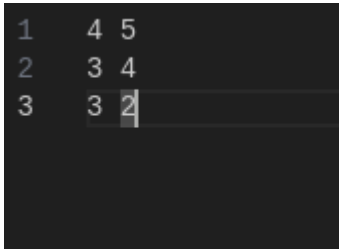
- hexadecimal -> *hread(linea, dato)*
- octal -> *oread(linea, dato)*

Closing file: to close the file, use the `file_close(<file>)` function

```
file_close(fichero);
```

Reading example

Let's imagine that we have a file called "*data.dat*" and we want to add the two columns.



1	4	5
2	3	4
3	3	2

For this we have a module (*adder.vhd*) that performs integer type sums.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use std.textio.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;

entity files_tb is
end files_tb;

architecture Behavioral of files_tb is
  component adder is
    Port (
      clk : in std_logic;
      rst_n : in std_logic;
      a, b : in integer;
      c : out integer
    );
  end component;
  signal clk : std_logic := '0';
  signal rst_n : std_logic;
  signal a, b : integer;
  signal c : integer;
  file fichero : text open read_mode is "datos.dat";

begin

DUT: adder
  Port map (
    clk => clk,
    rst_n => rst_n,
    a => a,
    b => b,
    c => c
  );

  clk <= not clk after 5ns;
  rst_n <= '0', '1' after 30 ns;
```

```

process
    variable linea : line;
    variable dato_1, dato_2 : integer;
begin
    while not endfile(fichero) loop
        readline (fichero, linea);
        read (linea, dato_1);
        a <= dato_1; -- assign the first data to a signal
        read (linea, dato_2);
        b <= dato_2; -- assign the first data to b signal
        wait for 100 ns; -- this period makes the sum
    end loop;

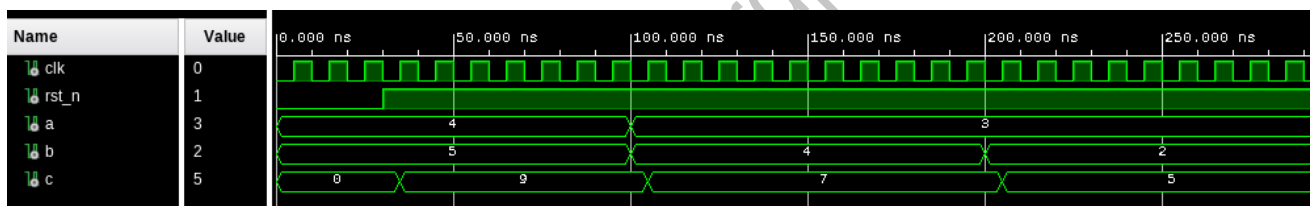
    file_close(fichero);
    report " END SIMULATION" severity failure;
end process;

end Behavioral;

```

This example makes use of the structure "*while not endfile() loop*" that extracts all the data from the file in a loop.

The simulation looks like this.



Writing

With this, reading is covered, now we move on to writing.

For writing, opening a file is the same as in the previous point, only with the «*write_mode*», and closing a file is the same, only with the writing file.

Writing value: to write there are several functions, the main one is `write(<output_file>, <value>)`, the value can be *integer* (with the `std.textio` library) or *std_logic_vector* (with the `std_logic_textio` library).

Also, there are two functions that allow writing data in hexadecimal -> *hwrite* and in octal -> *owrite*.

Writing example: In this example of use are all the functions discussed above.

```

write(output_line, std_logic_vector(to_unsigned(c, 8)));
write(output_line, c);
hwrite(output_line, std_logic_vector(to_unsigned(c, 8)));
owrite(output_line, std_logic_vector(to_unsigned(c, 8)));

```

Note: to insert characters into the file you can do it in the following way:

```
write(output_line, string(" - "));
```

Writing a line: To write a line you have to use the function *writeline(<output file>, <output line>)*. This function writes a line with the accumulated data.

NOTE: With this function it is not necessary to add line breaks(«\n»), because it already puts them automatically.

Writing example

For this example we will use the previous example as a basis, so we will read the values from the file «**data.dat**» and write the sum value in a file called «**data_out.dat**». This value will be written in different formats (binary, hexadecimal, integer and octal) separated by a hyphen.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use std.textio.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;

entity files_tb is
end files_tb;

architecture Behavioral of files_tb is
  component adder is
    Port (
      clk : in std_logic;
      rst_n : in std_logic;
      a, b : in integer;
      c : out integer
    );
  end component;
  signal clk : std_logic := '0';
  signal rst_n : std_logic;
  signal a, b : integer;
  signal c : integer;
  file input_file : text open read_mode is "data.dat";
  file output_file : text open write_mode is "data_out.dat";
begin

  DUT: adder
    Port map (
      clk => clk,
      rst_n => rst_n,
      a => a,
      b => b,
      c => c
    );

  clk <= not clk after 5ns;
  rst_n <= '0', '1' after 30 ns;

  process
    variable input_line : line;
    variable output_line : line;
```

```

        variable dato_1, dato_2 : integer;
    begin
        while not endfile(input_file) loop
-- read a and b data from the file
            readline (input_file, input_line);
            read (input_line, dato_1);
            a <= dato_1;
            read (input_line, dato_2);
            b <= dato_2;
            wait for 100 ns;

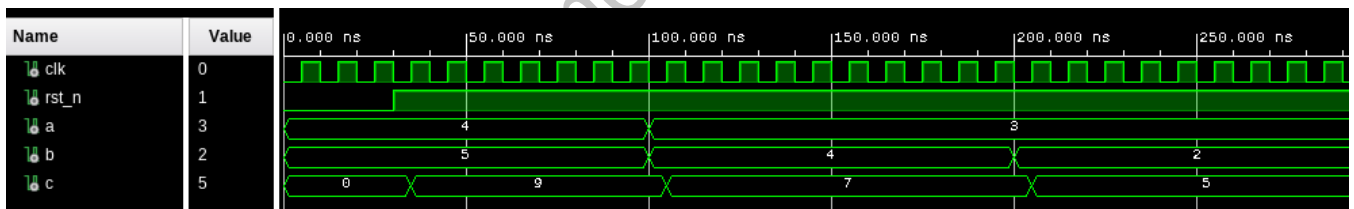
-- write the c value that came from adder module
            write(output_line, std_logic_vector(to_unsigned(c, 8)));
            write(output_line, string'(" - "));
            write(output_line, c);
            write(output_line, string'(" - "));
            hwrite(output_line, std_logic_vector(to_unsigned(c, 8)));
            write(output_line, string'(" - "));
            owrite(output_line, std_logic_vector(to_unsigned(c, 8)));
            writeline(output_file, output_line);
        end loop;

        file_close(input_file);
        file_close(output_file);
        report " END SIMULATION" severity failure;
    end process;

end Behavioral;

```

The simulation is the same as the reading simulation, only the results are saved in another file.



The generated file has the following form, the first format is in 8-bit std_logic_vector, the second in integer, the third in hexadecimal and the fourth in octal.

```

00001001 - 9 - 09 - 011
00000111 - 7 - 07 - 007
00000101 - 5 - 05 - 005

```

And with this you can read and write files in VHDL simulations.

NOTE 1:

Vivado reads and writes files by default in this path:

<proyecto>/<proyecto>.sim/sim_1/behav/xsim.

To modify it, add the path to the file read or write.

NOTE 2:

The *write* function has more parameters that have been omitted other internal parameters, such as the writing direction, to choose whether the data is joined from left to right or the other way around, options: *left* or *right*, by default, *right*. And the option of how many spaces are added to the left of the data, empty does not add any spaces.

```
write(file_line, var_data2, <writing direction>, <spaces per value>);
```

<https://soceame.wordpress.com/>