Cómo usar las funciones de la librería textio en VHDL

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/07/01/como-usar-las-librerias-textio-en-vhdl/

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

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Si has programado en VHDL es probable que hayas hecho simulaciones complejas en las que después de la simulación hayas tenido que analizar los datos, y para analizar los datos utilizando señales de simulación es bastante complejo. Bien, pues existe una librería en VHDL que permite hacer simulaciones con datos grabados en un fichero y grabar los resultados en un fichero. Esta librería es la **std.textio.**

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

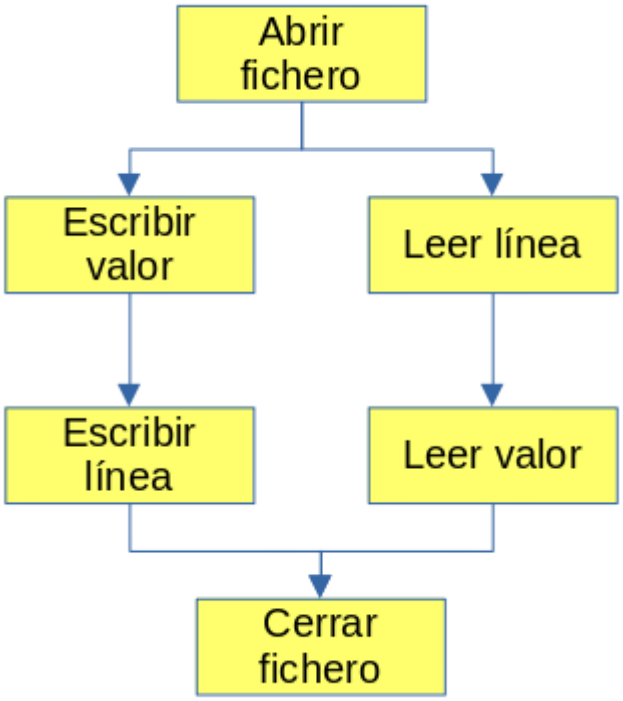
Esta es la librería base para poder leer y escribir en ficheros. Esta librería tiene el inconveniente de que no permite grabar datos *std_logic_vector* en ficheros, hay que convertirlos a tipo ***integer***.

Para poder usar *std_logic_vector* esta la librería ***std_logic_textio***.

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

Ambas librerías se pueden usar simultáneamente (esto es lo más recomendable)

La arquitectura de trabajo con esta librería es la siguiente:



Se basa en leer o escribir valor a valor o leer línea a línea cada fichero.

Aclaración: para poder leer un dato en un fichero, primero se tiene que abrir el fichero, extraer los datos que contiene, y después, se lee línea a línea el fichero.

NOTA: No se puede leer y escribir el mismo fichero, entonces, se tienen que utilizar dos ficheros distintos.

Lectura

Abrir un fichero: para abrir un fichero hay dos opciones,

- Usar la función «*file_open*«, con un file «*input*» tipo *text*, con el nombre del fichero (se recomienda .txt o .dat. Y por último el modo del fichero: **read_mode**, para lectura, **write_mode**, para escritura.

```
file input : text;
begin
...
file_open(input, "<nombre fichero>", <modo>);
```

- Declarar todo como un *file* junto con las constantes las señales.

```
file input : text open <modo> is "<nombre fichero>";
```

Leer una línea del fichero: para leer una línea del fichero se utiliza la función *readline(input, linea)*. Esta función tiene dos parámetros el tipo file creado para abrir el fichero y la variable del process tipo line. Un ejemplo:

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

readline(input , linea); -- leemos la linea del fichero y la volcamos en "linea"
end process;
```

Leer un dato de la línea: para leer un dato de la línea se utiliza la función *read(linea, dato)*. Esta función lee los datos de forma secuencial, por lo que cada vez que se usa en una línea lee un dato nuevo. Ejemplo:

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

readline(input , linea);
read(linea, dato1); -- cada vez que se lee, lee un dato nuevo
read(linea, dato2); -- y lo asigna a la señal datoX
read(linea, dato3);

end process;
```

Hay otras formas de leer datos en otro tipo de formato, como por ejemplo en hexadecimal o en octal:

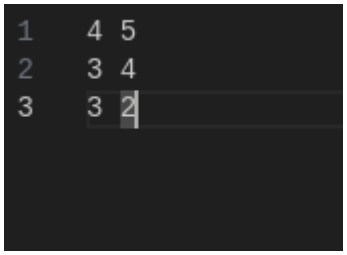
- hexadecimal -> *hread(linea, dato)*
- octal -> *oread(linea, dato)*

Cerrar fichero: para cerrar el fichero se utiliza la función *file_close(<fichero>)*

```
file_close(fichero);
```

Ejemplo de lectura

Imaginemos que tenemos un fichero llamado «*datos.dat*» y queremos hacer la suma de las dos columnas.



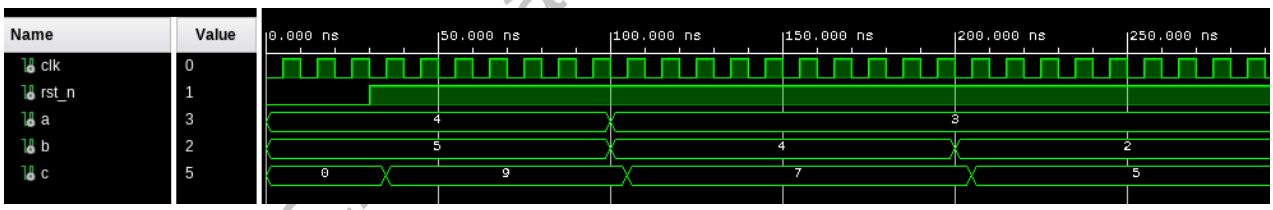
Para ello tenemos un módulo *(adder.vhd)* que hace las sumas de tipo *integer.*

```
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; -- asigno primer dato a señal a
            read (linea, dato_2);
            b <= dato_2; -- asigno segundo dato a señal b
            wait for 100 ns; -- en este periodo se hace la suma
        end loop;

        file_close(fichero);
        report " FIN DE LA SIMULACION" severity failure;
    end process;

end Behavioral;
```

Este ejemplo hace uso de la estructura «*while not endfile(<file fichero : text>) loop*» que extrae en bucle todos los datos del fichero.

La simulación queda de la siguiente forma.



Escritura

Con esto la lectura queda cubierta, ahora vamos a la escritura.

Para la escritura el abrir un fichero es igual que en el punto anterior solo que con el «***write_mode***«, y el cierre de un fichero es igual solo que con el fichero de escritura.

Escribir valor: para escribir hay varias funciones, la principal es write(<fichero_salida>, <valor>), el valor puede ser *integer* (con la librería std.textio) o *std_logic_vector* (con la librería std_logic_textio).

También, hay dos funciones que permiten escribir los datos en hexadecimal -> *hwrite* y en octal -> *owrite*.

Ejemplo de escritura: En este ejemplo de uso están todas las funciones comentadas anteriormente.

```
write(linea_salida, std_logic_vector(to_unsigned(c, 8)));
write(linea_salida, c);
hwrite(linea_salida, std_logic_vector(to_unsigned(c, 8)));
owrite(linea_salida, std_logic_vector(to_unsigned(c, 8)));
```

Nota: para insertar caracteres en el fichero se puede hacer de la siguiente forma:

```
write(linea_salida, string'(" - "));
```

Escribir una línea: para escribir una línea se tiene que utilizar la función *writeline(<fichero salida>, <línea de salida>)*. Esta función escribe una línea con los datos acumulados.

NOTA: Con esta función no es necesario añadir saltos de línea(«\n»), porque ya los pone automáticamente.

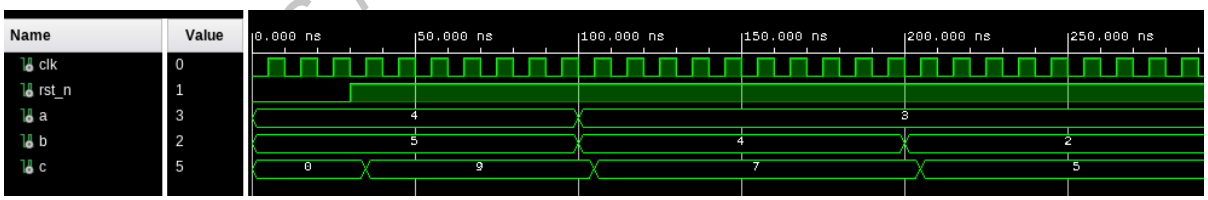
Ejemplo de escritura

Para este ejemplo nos vamos a basar en el ejemplo anterior, por lo que vamos a leer del fichero «**datos.dat**» los valores y vamos a escribir en un fichero llamado «**data_out.dat**» el valor de la suma. Este valor se va a escribir en diferentes formatos(binario, hexadecimal, entero y octal) separados por un guión.

```
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_entrada : text open read_mode is "datos.dat";
file fichero_salida : text open write_mode is "datos_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 linea_entrada : line;
        variable linea_salida : line;
        variable dato_1, dato_2 : integer;
    begin
        while not endfile(fichero_entrada) loop
-- leo los datos a y b del fichero
            readline (fichero_entrada, linea_entrada);
            read (linea_entrada, dato_1);
            a <= dato_1;
            read (linea_entrada, dato_2);
            b <= dato_2;
            wait for 100 ns;

-- escribo el valor de c que viene del sumador (adder)
            write(linea_salida, std_logic_vector(to_unsigned(c, 8)));
            write(linea_salida, string'(" - "));
            write(linea_salida, c);
            write(linea_salida, string'(" - "));
            hwrite(linea_salida, std_logic_vector(to_unsigned(c, 8)));
            write(linea_salida, string'(" - "));
            owrite(linea_salida, std_logic_vector(to_unsigned(c, 8)));
            writeline(fichero_salida, linea_salida);
        end loop;

        file_close(fichero_entrada);
        file_close(fichero_salida);
        report " FIN DE LA SIMULACION" severity failure;
    end process;

end Behavioral;
```

La simulación es igual a la de lectura solo que guardando los resultados en otro fichero.



El fichero generado tiene la siguiente forma, el primer formato es en std_logic_vector de 8 bits, el segundo en integer, el tercero en hexadecimal y el cuarto en octal.

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

Y con esto puedes leer escribir ficheros en simulaciones de VHDL.

NOTA 1:

Vivado lee y escribe ficheros por defecto en esta ruta:

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

Para modificarlo, añadir la ruta en la lectura o escritura del fichero.

NOTA 2:

La función *write* tiene más parámetros que se han omitido otros parámetros internos, como la dirección de escritura, para elegir si los datos se unen de izquierda a derecha o al revés, opciones: *left* o *right*, por defecto, *right*. Y la opción de cuantos espacios se añaden a la izquierda del dato, vacío no añade ningún espacio.

```
write(file_line, var_data2, <dirección de escritura>, <espacios por cada valor>);
```

Utilidad avanzada

https://soceame.wordpress.com/2025/02/09/que-son-las-metodologias-de-verificacion-y-como-crear-la-tuya-propia/