1- Introduction

2- Basic Structures of VHDL

3- Combinational Circuits

4- Sequential Circuits

5- Memory

6- Writing Testbenches

7- Synthesis Issues

8- RTL Cores

5- Memory

- ROM with Select
- ROM with Constant
- TEXTIO Package
- ROM with TEXTIO
- Register File

ROM with Select

- STD logic indexing

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY rom16x8 IS
    PORT (address: IN std_logic_vector(3 DOWNTO 0);
          data: OUT std_logic_vector(7 DOWNTO 0));
END ENTITY;

ARCHITECTURE sel OF rom16x8 IS
BEGIN
    WITH address SELECT
        data <= "11111011" WHEN "0000",
                "00010010" WHEN "0001",
                "10011011" WHEN "0010",
                "10010011" WHEN "0011",
                "01011011" WHEN "0100",
                "00111010" WHEN "0101",
                "11111011" WHEN "0110",
                "00010010" WHEN "0111",
                "10100011" WHEN "1000",
                "10011010" WHEN "1001",
                "01111011" WHEN "1010",
                "00010010" WHEN "1011",
                "10101001" WHEN "1100",
                "00110110" WHEN "1101",
                "11011011" WHEN "1110",
                "01010010" WHEN "1111",
                "XXXXXXXX" WHEN OTHERS;
END ARCHITECTURE sel;
```

- Describe a ROM as a simple decoder
- Indexing is done by *std_logic*, no integer correspondence

5- Memory

ROM with Constant

- Same ROM using **constant**
- Using variable class, not a signal class
- Integer indexing
- ROM array is an array indexed by integers
- Quartus synthesizer requires **signal** to synthesize to a ROM block

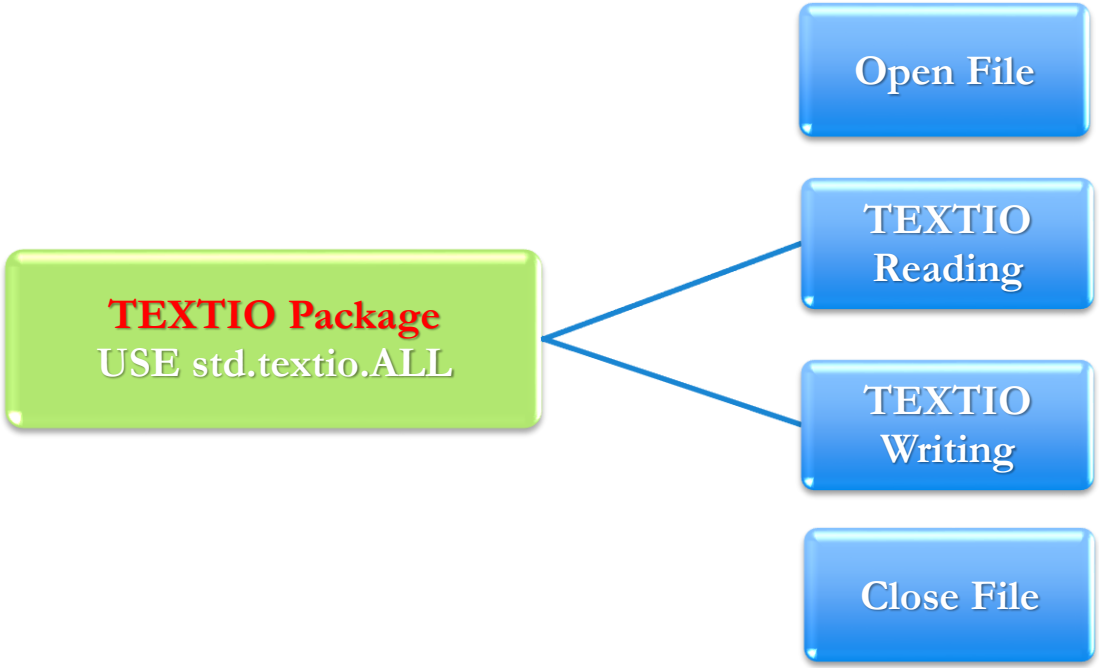
- Integer Indexing

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY rom16x8 IS
    PORT (address: IN integer RANGE 0 TO 15;
          data: OUT std_logic_vector(7 DOWNTO 0));
END ENTITY;

ARCHITECTURE const OF rom16x8 IS
    TYPE rom_array IS ARRAY (0 TO 15) OF std_logic_vector (7 DOWNTO 0);
    CONSTANT rom: rom_array := ( "11111011", "00010010", "10011011", "10010011",
                                 "01011011", "00111010", "11111011", "00010010",
                                 "10100011", "10011010", "01111011", "00010010",
                                 "10101001", "00110110", "11011011", "01010010");
BEGIN
    data <= rom(address);
END ARCHITECTURE const;
```

5- Memory

TEXTIO Package

- Can use TEXTIO for memory content initialization and dump



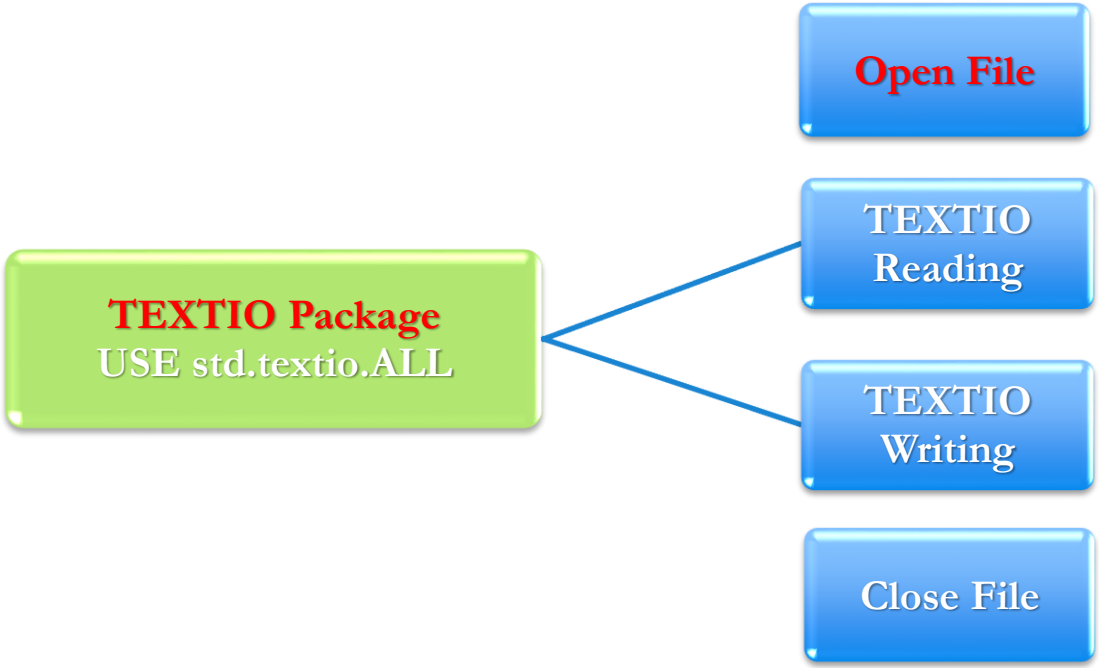
TEXTIO Package

- The **standard** package is part of VHDL **STD** library
- TEXTIO package is also part of the **STD** library
- TEXTIO package handles types in the **standard** package

TEXTIO is a package of VHDL functions that read and write text files.
To make the package visible: **USE std.textio.ALL;**

Data Types:
text => a file of character strings
line => one string from a text file

TEXTIO Package



TEXTIO Package – file open

- Opening files is done by `FILE_OPEN`

```
FILE_OPEN (fstatus, input_logic_value_file1, "input.dat", READ_MODE);
FILE_OPEN (fstatus, output_logic_value_file1, "output.dat", WRITE_MODE);
```

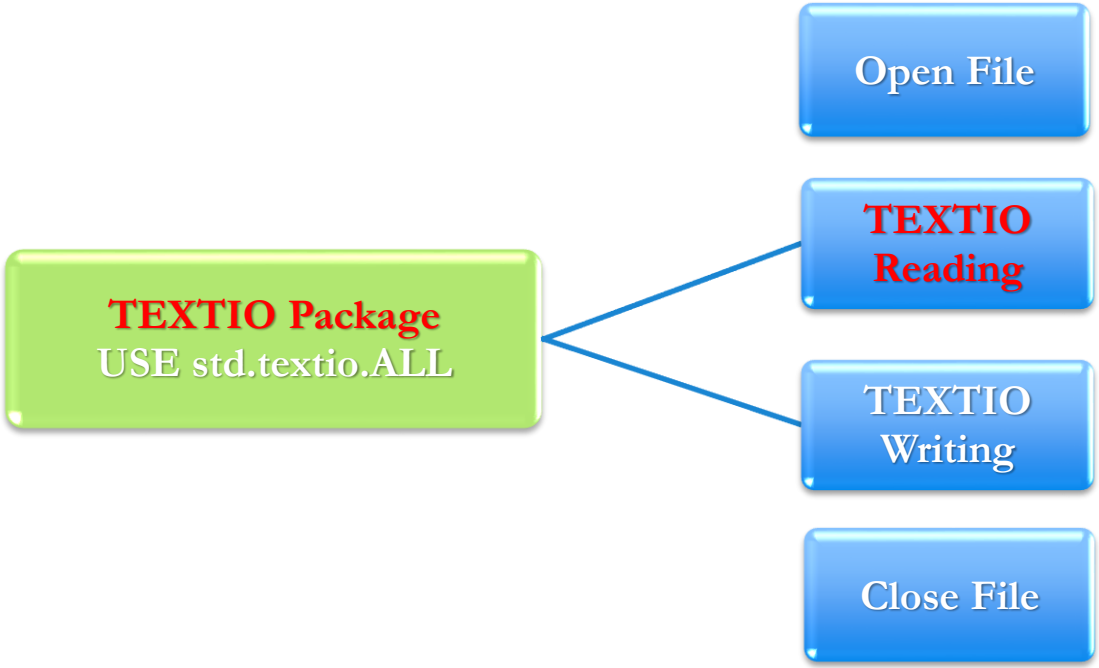
`FILE_OPEN_STATUS` type may be included as the first parameter of the FILE_OPEN statement:

```
OPEN_OK
STATUS_ERROR
NAME_ERROR
MODE_ERROR
```

```
FILE_CLOSE (input_logic_value_file1);
FILE_CLOSE (output_logic_value_file1);
```

TEXTIO Package

- Can use TEXTIO for memory content initialization and dump



TEXTIO Package - reading

- Performs a read line up to CR in a line
- Read a line up to CR
- Following a read line, READ data types of STD package from the buffer

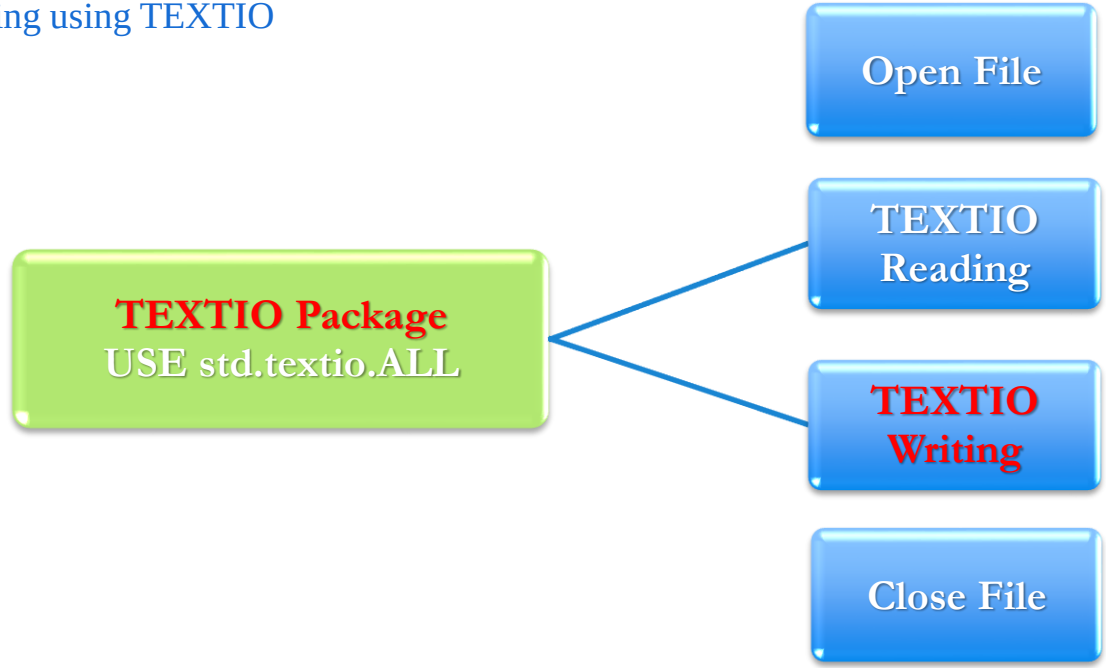
`READLINE(f, l)` -- reads a line of file *f* and places it in buffer *l* of type LINE

`READ(l, v, …)` -- reads a value *v* of its type form *l*

`ENDFILE(f)` -- returns TRUE if the end of file f is reached

TEXTIO Package

- Writing using TEXTIO



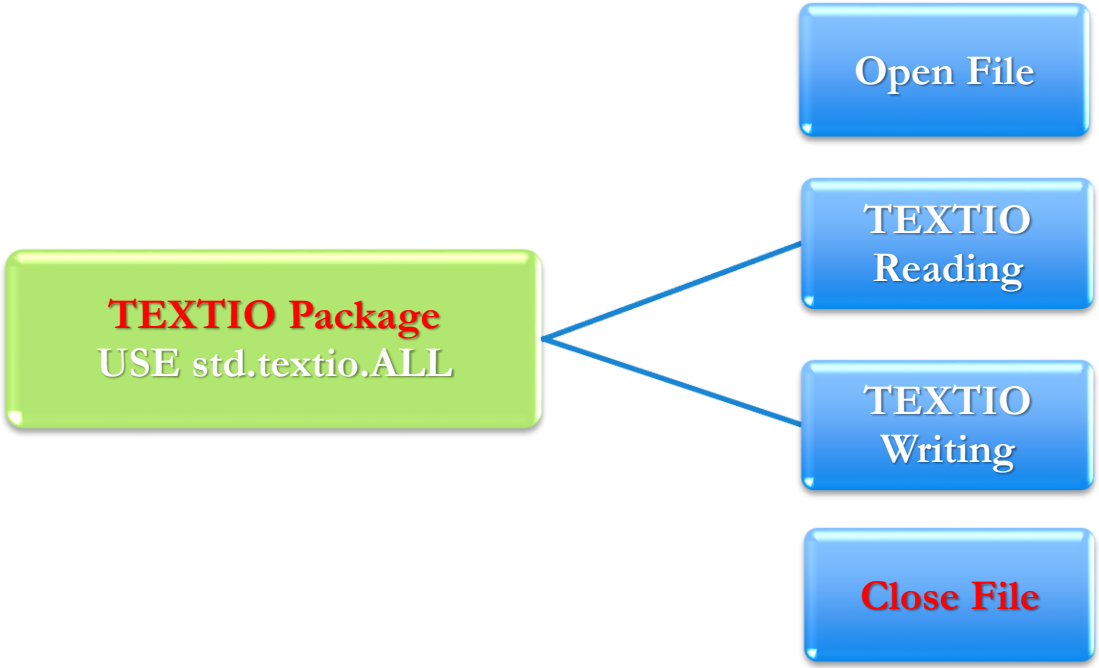
TEXTIO Package - writing

- WRITE(), writes STD types in buffer *l*
- WRITELINE writes buffer *l* in a file, including CR and LF

`WRITE(l, v, …)` -- writes the value *v* to LINE *l*

`WRIELINE(f, l)` -- writes *l* to file *f*. Function

TEXTIO Package



TEXTIO Package - close file

- Closing files is done by `FILE_CLOSE`

```
FILE_OPEN (fstatus, input_logic_value_file1, "input.dat", READ_MODE);
FILE_OPEN (fstatus, output_logic_value_file1, "output.dat", WRITE_MODE);
.
.
.
FILE_CLOSE (input_logic_value_file1);
FILE_CLOSE (output_logic_value_file1);
```

Std_logic TEXTIO

- The `IEEE.std_logic_textio.ALL.` library overloads the definitions in TEXTIO for `std_logic` and `std_logic_vector` types.

- Using `STD_LOGIC_TEXTIO` enables handling STD logic package types

ROM with TEXTIO

- A ROM model that initializes at time zero
- Initializing all memory locations to zero allows only locations of interest to be read from the file
- Use file contents to initialize memory
- Data can be in any order
- The initialization process suspends forever after the initialization file is read
- Line 44 reads from *std_logic_vector* type address location

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3   USE std.textio.ALL;
4   USE IEEE.std_logic_textio.ALL;
5   USE IEEE.std_logic_unsigned.ALL;
6
7   ENTITY rom16x8 IS
8       PORT(
9           address  : IN  std_logic_vector(3 DOWNTO 0);
10          dataout  : OUT std_logic_vector(7 DOWNTO 0));
11  END  ;
12
13  ARCHITECTURE txt OF rom16x8 IS
14      TYPE memory IS ARRAY (0 TO 15) OF std_logic_vector(7 DOWNTO 0);
15      SIGNAL data : memory;
16  BEGIN
```

ROM with TEXTIO

- A ROM model that initializes at time zero
- Initializing all memory locations to zero allows only locations of interest to be read from the file
- Use file contents to initialize memory
- Data can be in any order
- The initialization process suspends forever after the initialization file is read
- Line 44 reads from *std_logic_vector* type address location

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE std
4  USE IEE
5  USE IEE
6
7  ENTITY
8      POR
9
10
11 END   ;
12
13 ARCHITE
14     TYP
15     SIG
16 BEGIN
```

```
16 BEGIN
17
18     InitiateROM_proc: PROCESS
19         FILE     fptr          : text;
20         VARIABLE fstatus       : file_open_status;
21         VARIABLE file_line     : line;
22         VARIABLE var_location  : integer;
23         VARIABLE var_space     : character;
24         VARIABLE var_data      : std_logic_vector (7 DOWNTO 0);
25     BEGIN
26         FILE_OPEN(fstatus, fptr, "myfile.txt", read_mode);
27
28         FOR i IN 0 TO 15 LOOP
29             data(i) <= (OTHERS => '0');
30         END LOOP;
31
```

myfile.txt

```
1  0 10100010
2  1 10001010
3  2 10111010
4  3 10111010
5  4 10001010
6  5 10000010
7  6 00100011
8  7 00001011
9  8 00111010
10 9 01010010
11 10 10001010
12 11 10000011
13 12 00100011
14 13 00111011
15 14 10101101
16 15 01111011
```

Location Data

5- Memory

ROM with TEXTIO

- A ROM model that initializes at time zero
- Initializing all memory locations to zero allows only locations of interest to be read from the file
- Use file contents to initialize memory
- Data can be in any order
- The initialization process suspends forever after the initialization file is read
- Line 44 reads from *std_logic_vector* type address location

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE std
4  USE IEE
5  USE IEE
6
7  ENTITY
8      POR
9
10
11 END ;
12
13 ARCHITE
14     TYP
15     SIG
16 BEGIN
```

```
16 BEGIN
17
18     Ini
19
20
21
22
23
24
25     BEG
26
27
28
29
30
31
```

```
31
32         WHILE NOT ENDFILE(fptr) LOOP
33             READLINE(fptr, file_line);
34             READ(file_line, var_location);
35             READ(file_line, var_space);
36             READ(file_line, var_data);
37             data(var_location) <= var_data;
38         END LOOP;
39         FILE_CLOSE(fptr);
40
41         WAIT;
42     END PROCESS;
43
44     dataout<=data(conv_integer(address));
45
46 END txt;
```

myfile.txt

```
1  0 10100010
2  1 10001010
3  2 10111010
4  3 10111010
5  4 10001010
6  5 10000010
7  6 00100011
8  7 00001011
9  8 00111010
10 9 01010010
11 10 10001010
12 11 10000011
13 12 00100011
14 13 00111011
15 14 10101101
16 15 01111011
```

Location Data

5- Memory

Register File

- A register file with *std_logic* logic data type
- *IEEE.numeric_std* is needed for conversion

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY regFile IS
    PORT (
            clk : IN std_logic;
            rst : IN std_logic;
            we : IN std_logic;
            address : IN std_logic_vector (2 DOWNTO 0);
            inData : IN std_logic_vector (15 DOWNTO 0);
            outData : OUT std_logic_vector (15 DOWNTO 0)
        );
END regFile ;

ARCHITECTURE behavioral OF regFile IS
    TYPE reg_arr IS ARRAY (0 TO 7) OF std_logic_vector (15 DOWNTO 0);
    SIGNAL regArray : reg_arr ;
BEGIN
```

Register File

- A register file with *std_logic* logic data type
- *IEEE.numeric_std* is needed for conversion

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY regFile IS
END
ARC
BEG
```

```
BEGIN
    outData <= regArray(to_integer(unsigned(address)));

    wrProc: PROCESS (clk)
    BEGIN
        IF (clk = '1' AND clk'EVENT) THEN
            IF (rst = '1') THEN
                regArray <= (OTHERS => (OTHERS => '0'));
            ELSE
                IF (we = '1') THEN
                    regArray(to_integer(unsigned(address))) <= inData;
                END IF;
            END IF;
        END IF;
    END PROCESS;
END behavioral;
```

5- Memory