



درس: آزمایشگاه معماری کامپیوتر (جلسه نهم)

- مروری بر VHDL (ادامه)

- آزمایش نهم

نیمسال اول ۱۴۰۱

Subtype و Type

- علاوه بر type های پیش فرض VHDL امکان ایجاد فایل های دلخواه نیز وجود دارد.
- اما باید دقت شود که نمی توان از تمامی ابزارهای VHDL مانند اپراتورهای ریاضیاتی، شیفت، مقایسه و ... برای این type های جدید استفاده نمود. اما اگر subtype از type های predefined تعریف شود، این کار امکان پذیر خواهد بود.
- Subtype یعنی type ای که به صورت زیرمجموعه ای از یک type دیگر تعریف شده باشد.
- راه حل، تعریف package جدیدی از اپراتور یا function جدید یا overloading است.

TYPE my_type **is** ... (تعریف تایپ – به صورت literal یا range) ;

SUBTYPE my_subtype **IS** my_type (range) ;

مثال Type و Subtype

```
Type BYTE is array (7 downto 0) of std_logic;  
Signal b : Byte;  
b <= X"5C";  
b<=(others => '1');
```

```
subtype NATURAL is integer range 0 to integer'HIGH;  
subtype POSITIVE is integer range 1 to integer'HIGH;
```

```
subtype X01 is std_ulogic range 'X' to '1';  
subtype X01Z is std_ulogic range 'X' to 'Z';
```

```
Type v4l is ('X','0','1','Z');  
Subtype v3l is v4l range '0' to 'Z';
```

آرایه‌ها (Arrays)

- امکان تعریف آرایه در VHDL وجود دارد. می‌توان از انواع type‌های از پیش تعریف شده مانند bit, integer, std_logic_vector و ... و همچنین type‌های custom به منظور ایجاد آرایه استفاده نمود.
- همچنین تعریف آرایه‌های تو در تو (چند بعدی) نیز امکان‌پذیر است؛ می‌توان به همین شکل ماتریس ایجاد نمود.
- شماره index آرایه‌ها می‌تواند با توجه به range تعریف شده، متغیر باشد.
- برای مثال type‌های string و std_logic_vector به ترتیب آرایه ای از character و std_logic هستند.
- آمان آرایه را می‌توان به صورت منفرد و یا با استفاده از concatenation, aggregation, slice و به صورت ترکیبی مقداردهی نمود.

نحوه تعریف آرایه

TYPE array_name **is array** (range) **of** element_type;

(تعریف یک آرایه از تایپ‌های موجود یا تایپ دلخواه)

Signal signal_name : array_name;

(تعریف object از type جدید تعریف شده)

مثال روش‌های تعریف آرایه

```
Type my_array1 is (0 to 15) of INTEGER;  
Signal my_signal1 : my_array1;
```

```
Type my_type is (3 downto 0) of std_logic;  
Type my_array2 is array (31 downto 0) of my_type;  
Signal my_signal2 : my_array2;
```

```
Type my_array3 is ( integer range < > ) of std_logic_vector(1 downto 0);  
Signal my_signal3 : my_array3;
```

مثال روش‌های مقداردهی به آرایه

تعریف به صورت نامحدود



```
Type my_array3 is ( integer range < > ) of std_logic_vector(1 downto 0);
Signal my_signal3 : my_array3;

Type Nibble is array (3 downto 0) of std_logic;
Type Memory is array (7 downto 0) of Nibble;
Signal Mem8X4 : Memory ;

--
Mem8X4<=("0110","1111","1010","0000","0100","0111","1111","1110");
Mem8X4(7) <= "0110"; Mem8X4(7)(3) <= '0';

-----

Type 2D_array is array (0 to 7 , 2 downto 0) of integer range 0 to 127;
Signal singal_2D : 2D_array;

--
signal_2D <= ((5,0,10),(50,3,1),..., (14,6,6));
signal_2D(1) <= (50,3,1); signal_2D(7,2) <= 14;
```

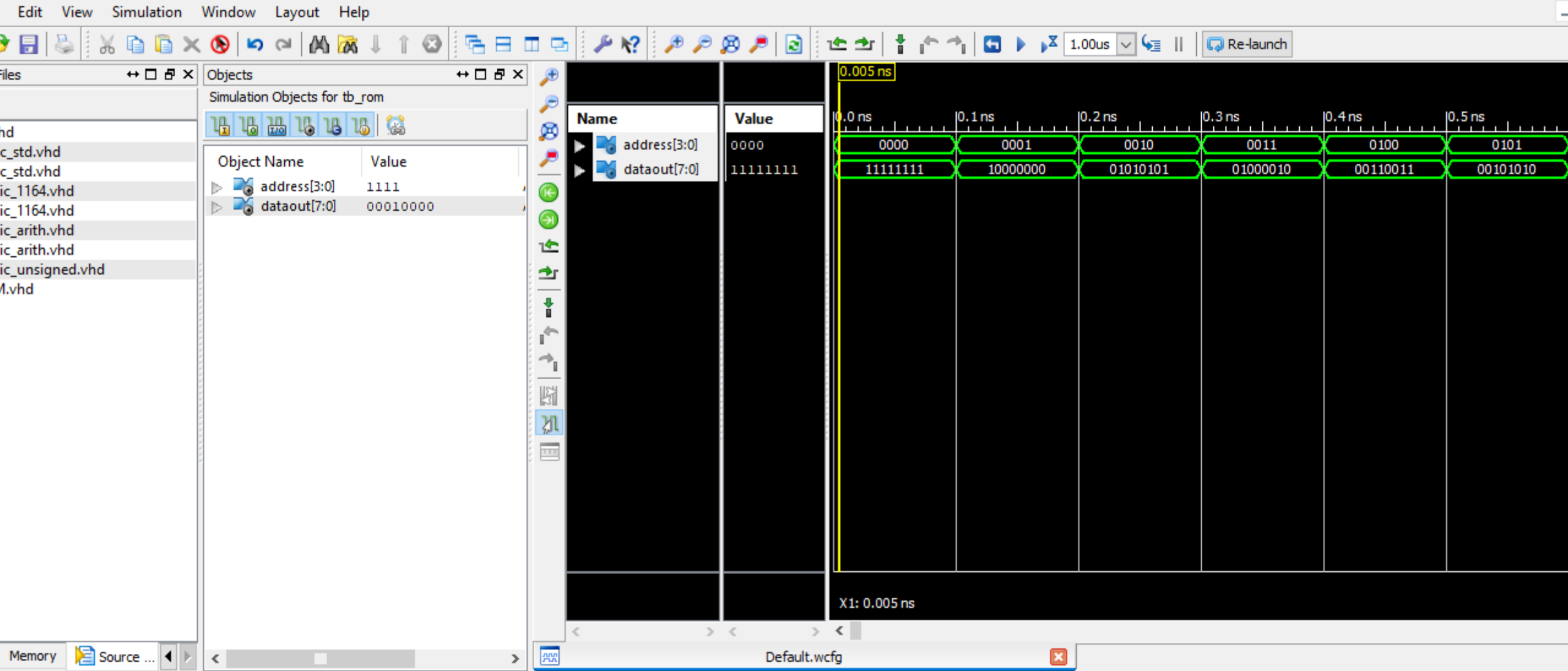

ROM

حافظه ای که یکبار مقداردهی می شود و پس از آن فقط قابلیت خواندن دارد.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_unsigned.ALL;
4 use IEEE.numeric_std.ALL;
5
6 entity ROM16X4 is
7     port( --clk: in std_logic;
8           address : in std_logic_vector(3 downto 0) := "0000";
9           dataout : out std_logic_vector(7 downto 0));
10 end ROM16X4;
11
12 architecture Behavioral of ROM16X4 is
13
14     TYPE romdata is ARRAY (0 to 15) of std_logic_vector(7 downto 0);
15
16     constant data : romdata := ("11111111","10000000","01010101",
17     "01000010","00110011","00101010","00100100","00100000",
18     "00011100","00011001","00010111","00010101","00010011",
19     "00010010", -- data in address 1101
20     "00010001", -- data in address 1110
21     "00010000"); -- data in address 1111
22 begin
23     --process (clk)
24     --begin
25         --if rising_edge(clk) then
26             dataout <= data(to_integer(unsigned(address)));
27         --end if;
28     --end process;
29 end Behavioral;
```


Test bench

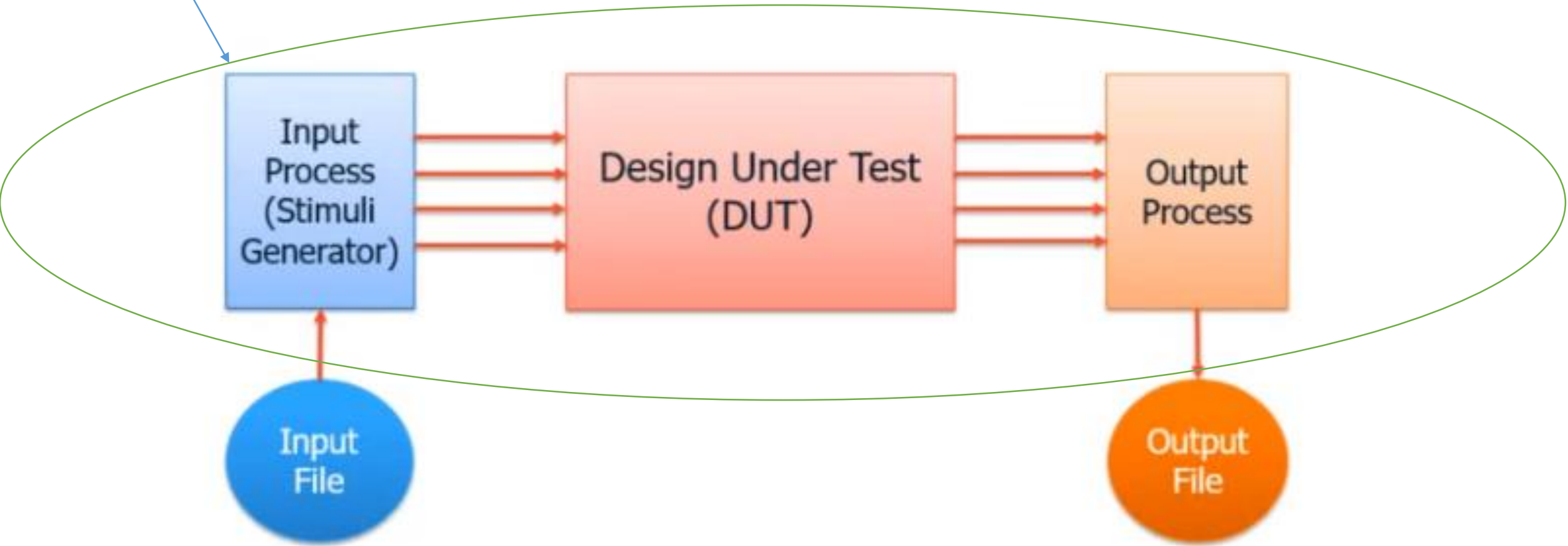
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY tb_ROM IS
6  END tb_ROM;
7
8  ARCHITECTURE behavior OF tb_ROM IS
9      COMPONENT ROM16X4
10         PORT (
11             address : IN  std_logic_vector(3 downto 0);
12             dataout  : OUT std_logic_vector(7 downto 0)
13         );
14     END COMPONENT;
15     signal address : std_logic_vector(3 downto 0) := (others => '0');
16     signal dataout : std_logic_vector(7 downto 0);
17 BEGIN
18     uut: ROM16X4 PORT MAP (address,dataout);
19
20     process begin
21         for i in 0 to 15 loop
22             address <= std_logic_vector(to_unsigned(i,address'length));
23             wait for 100 ps;
24         end loop;
25         wait;
26     end process;
27
28 END;
```



0131013 (signature 0x7708f090)
Full version of ISim.
Resolution is 1 ps
or is doing circuit initialization process.
circuit initialization process.

test bench محیط

File Read & Write



File Read & Write

- در VHDL امکان خواندن و نوشتن در فایل ها به صورت محدود وجود دارد. گستره ی دسترسی به فایل ها در VHDL قابل مقایسه با زبان های نرم افزاری نیست. فایل ها در VHDL معمولاً از نوع text و با فرمت ASCII هستند اما محتوای آنها می تواند هر type باشد مانند std_logic_vector ، integer ، string و ...
- کاربرد فایل ها در ذخیره سازی یا استفاده از منابع داده در testbench است. به طوری که می توان داده های موجود در فایل را به عنوان ورودی یک سیستم تزریق نمود یا اینکه نتایج به دست آمده را در فایل ذخیره نمود به منظور مقایسه، بررسی و ...
- File به نوعی ارتباط بین کد VHDL و سیستم host هستند. محل ذخیره فایل ها در محل ذخیره پروژه است.
- قابل سنتز نیستند.

File Declaration

علت وجود تعاریف مختلف edition های مختلفی است که برای vhdl ارائه شده است.

- نحوه تعریف تایپ File و تعریف File به صورت زیر است:

```
TYPE TEXT is file of string;  --ASCII
TYPE INT is file of integer;  --sequence of integer stored in binary form

FILE text_file : TEXT;        --file handle
FILE integer_file : INT;      --file handle
FILE text_file : TEXT is "myfile.txt";
FILE text_file : TEXT OPEN READ_MODE is "input.txt";
FILE text_file : TEXT OPEN WRITE_MODE is "output.dat";
```

این object را می توان هم در بخش architecture و هم در بخش processes تعریف کرد.

- هر فایل قبل از استفاده می بایست OPEN شود و بعد از پایان استفاده نیز باید CLOSE شود. دستورات آن به شرح زیر است:

```
procedure FILE_OPEN(
  File_Status : out FILE_OPEN_STATUS;
  file_file_handle : FILE_TYPE;
  File_Name : in STRING;
  Open_Kind : in FILE_OPEN_KIND := READ_MODE);
```

```
procedure FILE_CLOSE(file file_handle: FILE_TYPE);
```

Open_ok
Status_error
Name_error
Mode_error

File Declaration

- Subprogram های VHDL برای خواندن و نوشتن در فایل:

```
procedure READ(file file_handle: FILE_TYPE; value: out type);  
procedure WRITE(file file_handle: FILE_TYPE; value: in type);  
function ENDFILE(file file_handle: FILE_TYPE) return Boolean;
```

- این Subprogram ها برای Type های پیشفرض VHDL است و برای type های جدید باید overload شوند یا مجددا تعریف شوند.
- تا به اینجا تمامی object ها یا construct ها در کتابخانه اصلی VHDL تعریف شده هستند.

File Declaration

- Package به نام TEXTIO در کتابخانه STD وجود دارد که دارای مجموعه ای کامل تر از type ها و توابع TEXTIO می باشد. از این پس از همین Package استفاده می کنیم. این Package یعنی STD.textio شامل subprogram های خواندن و نوشتن type های پیش فرض VHDL می باشد مانند Bit, integer, character؛ همچنین Package دیگری با نام std_logic_textio برای type های std_logic و std_logic_vector ... تعریف شده است.

```
use std.textio.all;  
--use ieee.std_logic_textio.all;
```


File Declaration

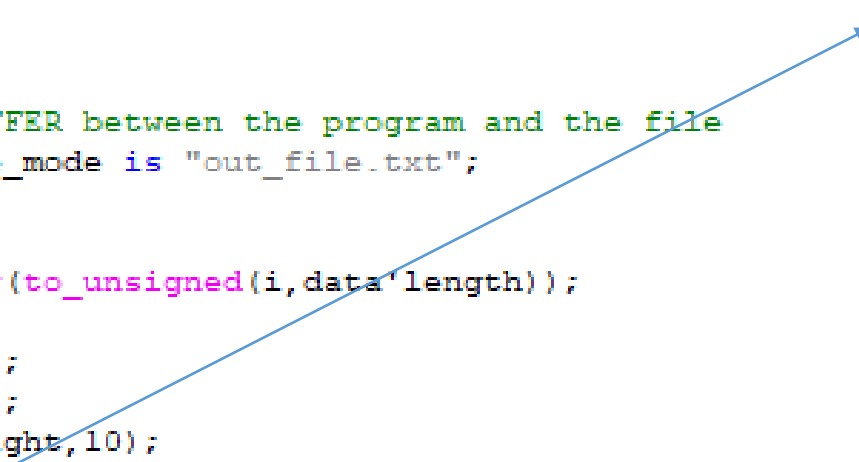
- در این پکیج ها type ها و Subprogram های بسیاری تعریف شده اند از جمله:

```
type LINE is access STRING;  --Line is pointer to string and also a buffer for RW
type TEXT is file of STRING; --file of variable-length ASCII records
procedure READLINE(file F : TEXT; L : out LINE);
procedure READ(L : inout LINE; value : out bit); --overloaded for bit_vector and string
procedure WRITELINE(file F : TEXT; L : inout LINE);
procedure WRITE(L : inout LINE; value: out bit); --overloaded for bit_vector and string
```

- از طرفی Overload های دیگری نیز وجود دارد. مثال مشاهده شود.

```
procedure WRITE(L: inout LINE; VALUE: in
  BIT_VECTOR; JUSTIFIED: in SIDE:= RIGHT,
  FIELD: in WIDTH := 0);
procedure WRITE(L: inout LINE; VALUE: in
  STRING; JUSTIFIED: in SIDE:= RIGHT,
  FIELD: in WIDTH := 0);
```

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4  use STD.textio.all;
5  use ieee.std_logic_textio.all;
6
7  entity TEXTIO_File is
8  end TEXTIO_File;
9
10 architecture my_file of TEXTIO_File is
11     signal data : std_logic_vector(7 downto 0) := X"00";
12 begin
13
14     process is
15         variable row : line; -- BUFFER between the program and the file
16         file dump : text open write_mode is "out_file.txt";
17     begin
18         for i in 1 to 16 loop
19             data <= std_logic_vector(to_unsigned(i,data'length));
20             write(row,i,right,5);
21             write(row,data,right,12);
22             hwrite(row,data,right,8);
23             hwrite(row,X"00"&data,right,10);
24             write(row,string'(time'image(NOW + 1 ns)),right,10);
25             writeline(dump,row); -- writes the buffer(row) in the file and goes to next line
26             wait for 1 ns;
27         end loop;
28         write(row,string'("end of out_file"),right,30);
29         write(row,bit_vector'("01"));
30         writeline(dump,row);
31         --file_close(dump); -- not needed
32         wait;
33     end process;
34 end architecture my_file;
```



تبدیل به string

RAM (Random Access Memory)

هم قابلیت خواندن و هم قابلیت نوشتن را دارد.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 entity RAM is
5     generic(
6         data_width : natural := 16;
7         address_width : natural := 8);
8     port(
9         clk : in std_logic;
10        address : in std_logic_vector(address_width-1 downto 0);
11        rw : in std_logic; -- Read / Write
12        cs : in std_logic; -- Chip Select
13        data : inout std_logic_vector(data_width-1 downto 0)
14    );
15 end entity RAM;
16 architecture Behavioral of RAM is
17 begin
18     process (clk)
19         type mem_array is array (2**address_width-1 downto 0) of std_logic_vector(data_width-1 downto 0);
20         variable memory : mem_array := (others => (others => '0'));
21     begin
22         if rising_edge(clk) then
23             if cs = '1' then
24                 if rw = '1' then --write in memory @ the address
25                     memory(to_integer(unsigned(address))) := data;
26                 else --read from memory @ the address
27                     data <= memory(to_integer(unsigned(address)));
28                 end if;
29             end if;
30         end if;
31     end process;
32 end Behavioral;
```

با استفاده از فایل ها میشه مقداردهی اولیه نمود.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Generic_RAM is
6      generic(
7          data_width : natural := 16;
8          address_width : natural := 8);
9      port(
10         clk : in std_logic;
11         reset : in std_logic;
12         datain : in std_logic_vector(data_width-1 downto 0);
13         address : in std_logic_vector(address_width-1 downto 0);
14         rw : in std_logic;
15         cs : in std_logic;
16         dataout : out std_logic_vector(data_width-1 downto 0)
17     );
18 end entity Generic_RAM;
19
20 architecture Behavioral of Generic_RAM is
21     type mem_array is array (2**address_width-1 downto 0) of std_logic_vector(data_width-1 downto 0);
22     --signal memory : mem_array;
23 begin
24     process (clk)
25         variable memory : mem_array;
26     begin
27         if rising_edge(clk) then
28             if reset = '1' then
29                 for i in memory'range loop
30                     memory(i) := (others => '0');

```

```
31 file file_into_RAM : text;
32 file output_file : text;
33
34 procedure GetData(    -- Gets data from the input file
35 signal data_into_RAM : out std_logic_vector(data_width-1 downto 0);
36 signal addr : out std_logic_vector(address_width-1 downto 0);
37 file RAM_init_file : text
38 --constant tim : time
39 ) is
40     variable row : line;
41     variable t : time := 0 fs;
42     variable add : std_logic_vector(address_width-1 downto 0) := (others => '0');
43     variable data : std_logic_vector(data_width-1 downto 0);
44 begin
45     while not endfile (RAM_init_file) loop
46         readline(RAM_init_file, row);
47         --read(row,data); -- binary
48         hread(row, data); -- hex
49         hread(row, add);
50         read(row, t);
51         data_into_RAM <= data;
52         addr <= add;
53         wait for t;
54         --wait for tim;
55     end loop;
56     file_close(RAM_init_file);
57 end procedure GetData;
```

out_9e.txt					
output.txt					
RAM_init.txt					
1	Data	Address	Time		
2	00FF	01	10 ns		
3	0101	02	10 ns		
4	0A01	0A	10 ns		
5	0223	0F	11 ns	--for example	
6	0E00	14	12 ns	--for example	

این زمان با توجه به دوره کلاک در نظر گرفته شده که چون ۱۰ گذاشته بودیم اینها فرقی ندارند در اینجا

گزارش کار

- یک ROM با خط آدرس ۵ بیتی و داده های ۸ بیتی طراحی نمایید و در شبیه ساز به ازای مقادیر مختلف آدرس محتویات آن را نمایش دهید.
- برای RAM فایلی مشابه RAM_INIT نوشته و به ازای آن شبیه سازی را برای GENERIC_RAM ارائه شده در کلاس انجام و توضیح دهید.