University of Basrah

College of Engineering

Computer Engineering Department

Digital Systems Design (CoE233)

First Course 2022-2023

Hardware Description Language

Introduction to VHDL

- **VHDL** is a **H**ardware **D**escription **L**anguage (**HDL**). It describes the behavior of an electronic circuit or system.
- **VHDL** is a standard, technology/vendor independent language, and is therefore portable and reusable.
- **VHDL** code has been written, it can be used either to implement the circuit in a programmable device (from Altera, Xilinx, Atmel, etc.) or any other chips.
- The two most popular hardware description languages are **VHDL** and **Verilog**.
- The acronym VHDL stands for
  - ⇨ **VHSIC** **H**ardware **D**escription **L**anguage **(VHSIC** =**V**ery **H**igh-**S**peed **I**ntegrated **C**ircuits).
- A **H**ardware **D**escription **L**anguage (**HDL**) allows a digital system to be designed and debugged at a higher level before implementation at the gate and flip-flop level.
- **VHDL** is insensitive, that is, capital and lower-case letters are treated the same by the compiler and the simulator.

  For example:

  CLK <= NOT clk;

  clk <= not CLK;
- The symbol **"<="** is the signal assignment operator which indicates that the value computed on the right-hand side is assigned to the signal on the left side.
- Comments in **VHDL** are indicated with a "double dash", i.e., "--".

  Example:

  -- main program

  Data_in <= Data_bus; ***-- reading data***
- **VHDL Simulator** is used to verify the correct functioning of the circuit in terms of waveforms showing input and output signal variations.
- **VHDL Synthesizer** is used to translate the source code to a description of the actual hardware circuit that implements the code. The output of synthesizer can be used directly to implement actual circuit in **FPGA**.
- **VHDL description includes two parts:**
  - ➢ **Entity**
  - ➢ **Architecture**

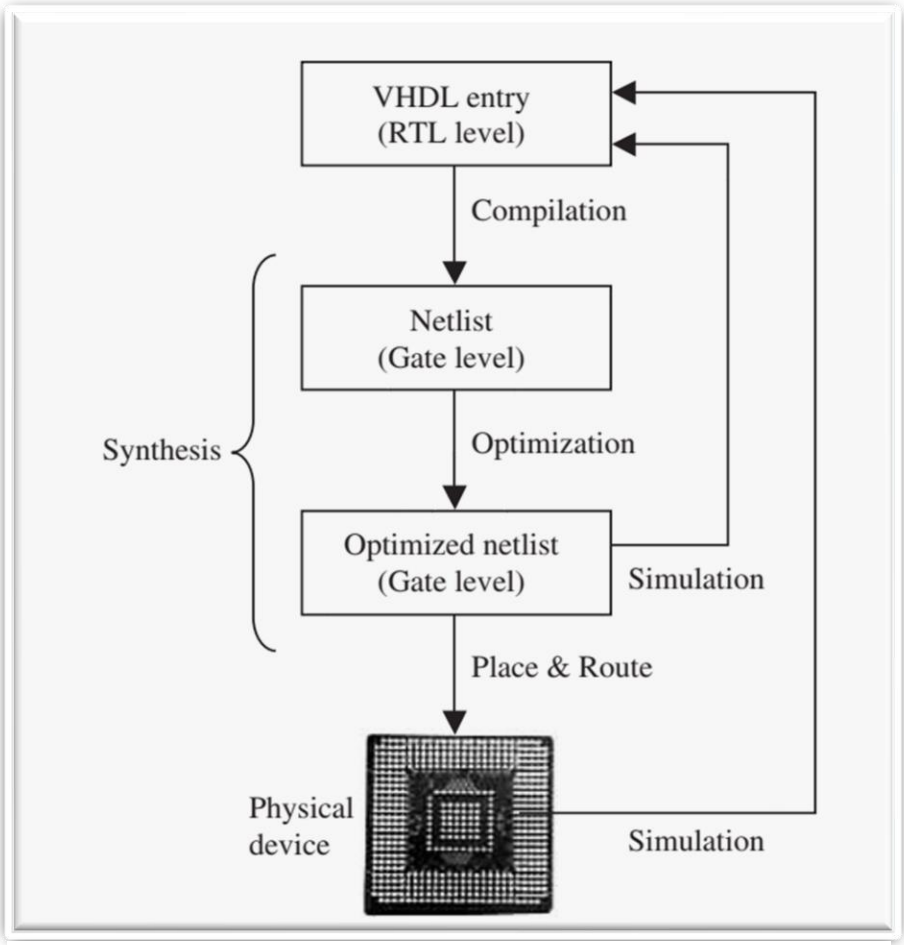


Figure 1 Summary of VHDL design flow

- One of the major utilities of VHDL is that it allows the synthesis of a circuit or system in a programmable device (PLD or FPGA). The steps followed during such a project are summarized in Figure 1.
- We start the design by writing the VHDL code, which is saved in a file with the extension .vhd and the same name as its ENTITY's name.
- The first step in the synthesis process is compilation.
    - ⇨ Compilation is the conversion of the high-level VHDL language, which describes the circuit at the Register Transfer Level (RTL), into a netlist at the gate level.
- The second step is optimization, which is performed on the gate-level netlist for speed or for area. At this stage, the design can be simulated.
- Finally, a place and route (fitter) software will generate the physical layout for a PLD/FPGA chip.

VHDL Code Structure

VHDL consists at least three fundamental sections.

	Section	Detail
1	LIBRARY declarations	Contains a list of all libraries to be used in the design. For example: IEEE, STD, work, etc.
2	ENTITY	Specifies the I/O pins of the circuit.
3	ARCHITECTURE	Contains the VHDL code proper, which describes how the circuit should behave(function).

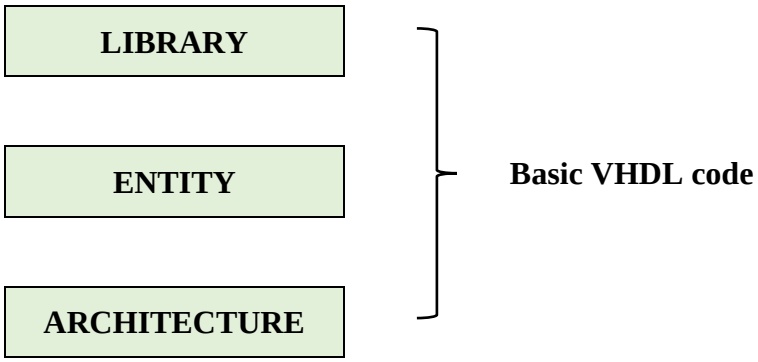


Figure 2 Fundamental sections of a basic VHDL code

1. Library

A LIBRARY is a collection of commonly used pieces of code. Placing such pieces inside a library allows the programmer to be reused or shared by other designs. The typical structure of a library is illustrated in the Figure 3:

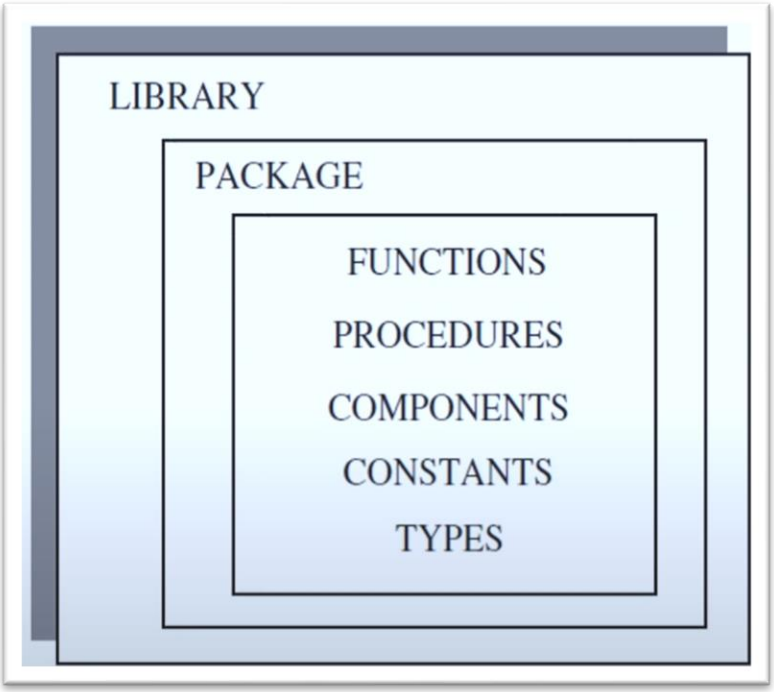


Figure 3 Fundamental parts of a LIBRARY

Library Declarations

To declare a LIBRARY (that is, to make it visible to the design) two lines of code are needed, one containing the name of the library, and the other a use clause, as shown in the syntax below:

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```

At least three packages, from three different libraries, are usually needed in a design:

- IEEE.STD_LOGIC_1164 (from the IEEE library);
  - Specifies digital logic system, including STD_LOGIC, and STD_LOGIC_VECTOR types.
- Standard (from the STD library);
  - Specifies built-in data types (BIT, BOOLEAN, INTEGER, REAL, SIGNED, UNSIGNED, etc.), arithmetic operations, basic type conversion, etc.
- Work (work library);
  - Holds current designs after compilation

Their declarations are as follows:

LIBRARY IEEE;
USE ieee.std_logic_1164.all;

LIBRARY std;
USE std_standard.all;

LIBRARY work;
USE work.all;

The IEEE library contains several packages:

- **STD_LOGIC_1164**
- **STD_LOGIC_ARITH**
- **STD_LOGIC_SIGNRD**
- **STD_LOGIC_UNSIGNED**

2. Entity

The main part of an ENTITY is PORT, which is a list with specifications of all input and output ports (pins) of the circuit.

Entity *declaration form*:

```
ENTITY entity_name IS
PORT (signal_name1 : mode   signal type;
      signal_name2 : mode   signal type;
            :
      signal_nameN : mode   signal type);
END entity_name;
```

Note:

- The name of the entity can be basically any name, except VHDL reversed words.
- *Ports have name, mode, and datatype.*
- *Port names always begin with a letter and consist of letters, digits, and/or underscores.*
- *The mode of signal can be* **IN**, **OUT**, **INOUT**, or **BUFFER**.
- **IN** *and* **OUT** *are truly unidirectional pins, while* **INOUT** *is bidirectional.*
- **BUFFER** *is used when the output signal must be used(read) internally as shown in Figure 4 below:*

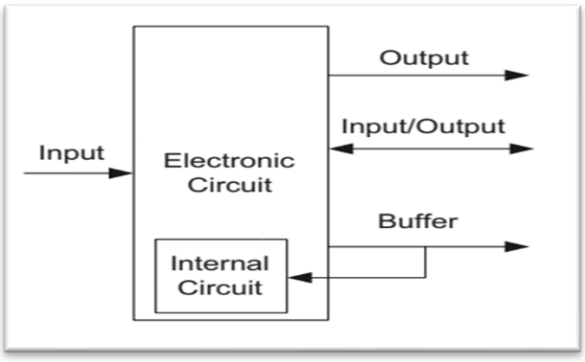
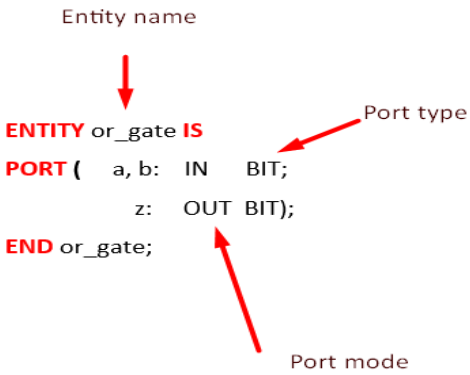
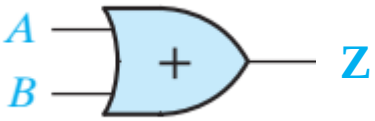


Figure 4 Signal modes

Example: Write the appropriate entity definition for OR gate with 2 inputs.





3. Architecture

Architecture describes the circuit implementation (systems' behavior).

Architecture *declaration form*:

```
ARCHITECTURE architecture_name OF entity_name IS
[declarations: type, signal, constant, function, procedure, component]
BEGIN
Code (Architecture Body);
END architecture_name;
```

As shown above, architecture has two parts: a declarative part (optional), where signals and constants are declared, and the code part.

The name of an architecture can be basically any name (except VHDL reversed words), including the same name as the entity's name.

Example: Write a VHDL code for OR gate with 2 inputs.

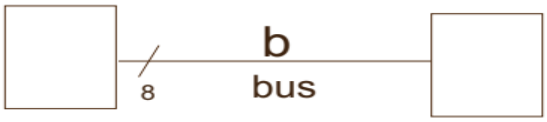
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


Entity or_gate IS
PORT( a,b : IN BIT;
        z: OUT BIT);
END or_gate;


Architecture  circuit OF or_gate IS

BEGIN

z <= a or b;

END circuit;
```

Modeling Wires and Buses

Signals

SIGNAL a: STD_LOGIC;

SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);



Standard Logic Vectors

Example 1:

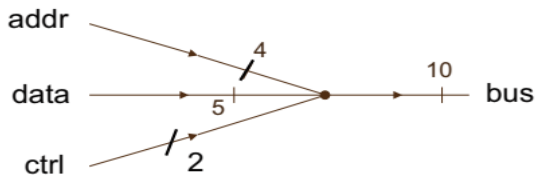
```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(8 DOWNTO 0);
 ……..
a <= '1';        --assign a with logic ONE
b <= "0000";     --Binary base assumed by default
c <= B"0000";    --Binary base explicitly specified
d <= X"AF67";    -- Hexadecimal base
e <= O"723";     -- Octal base
```

Example 2:

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <= "0000";
b <= "1111";
c <= a & b;                 -- c = "00001111"

d <= '0' & "0001111";       -- d <= "00001111"

e <= '0' & '0' & '0' & '0' & '1' & '1' &
     '1' & '1';             -- e <= "00001111"
```

Merging Wires and Buses

Example:




```
SIGNAL addr : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL data : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL ctrl : STD_LOGIC_vector(1 downto 0);
SIGNAL bus  : STD_LOGIC_VECTOR(10 DOWNTO 0);

bus <= addr & data & ctrl;
```

Data Types

There are two types of data types.

1. Pre-defined data types
2. User-defined data types

1. Pre-defined data types

VHDL contains a series of pre-defined data types, such data type definitions can be found in the following packages/ libraries:

Package standard of library **std**: Defines:

- ✓ **BIT**
- ✓ **BOOLEAN**
- ✓ **INTEGER**
- ✓ **REAL**

Package std_logic_1164 of library **ieee**: Defines:

- ✓ **STD_LOGIC**
- ✓ **STD_ULOGIC**

Package std_logic_arith of library **ieee**: Defines:

- ✓ **SIGNED**
- ✓ **UNSIGNED**

Also defines several data conversion functions, like

- ✓ **conv_integer(p)**
- ✓ **conv_unsigned (p,b)**
- ✓ **conv_signed (p,b)**
- ✓ **conv_std_logic_vector(p,b)**

Packages std_logic_signed and **std_logic_unsigned** of library **ieee** contain functions that allow operations with STD_LOGIC_VECTOR data to be performed as if data were of type SIGNED or UNSIGNED, respectively.

1. BIT (‘0’, ‘1’) and BIT_VECTOR

Examples:

```
signal x: bit;
signal y: bit_vector(3 downto 0);
signal w: bit_vector(0 to 7);
-------------------------------------
x <= '1';
y <= "0111";
w <= "01110001";
-------------------------------------
```

2. STD_LOGIC and STD_LOGIC_VECTOR

IEEE standard 1164 provides a standard data type (**STD_LOGIC**) with eight values. Object of these types can have the following values:

Value	Meaning
'X'	Forcing (Strong driven) Unknown
'0'	Normal 0; Forcing (Strong driven)
'1'	Normal 1; Forcing (Strong driven)
'Z'	High Impedance
'W'	Weak (Weakly driven) Unknown Logic Value
'L'	Weak (Weakly driven) 0.
'H'	Weak (Weakly driven) 1.
'-'	Don't Care

Examples:

```
signal x1: std_logic;
signal x2: std_logic_vector(3 downto 0);
-------------------------------------
x1 <= '0';
x2 <= "ZZZZ";
-------------------------------------
```

3. STD_ULOGIC and STD_ULOGIC_VECTOR

9-level logic system introduced in the IEEE 1164 standard ('**U**', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-').

(U: means Uninitialized).

4. BOOLEAN: True, False.

5. INTEGER: 32-bit integers from -2,147,483,648 to +2,147,483,647.

6. NATURAL: Non-negative integers from 0 to +2,147,483,647.

7. POSTIVE: An integer in range 1 to +2,147,483,647

8. REAL: Real numbers ranging from - $1.7e^{38}$ to $+1.7e^{38}$

9. Physical literals:

A physical type is one that contains both a value and units. In VHDL, time is the primary supported physical type.

Type	Values that the type can take on	
Time (unit relationships)	fs	(Femtosecond, 10^{-15}), base unit
	ps = 1000 fs	(Picosecond, 10^{-12})
	ns = 1000 ps	(Nanosecond, 10^{-9})
	µs = 1000 ns	(Microsecond, 10^{-6})
	ms = 1000 µs	(Millisecond, 10^{-3})
	s = 1000 ms	(Second)
	min = 60 s	(Minute)
	h = 60 min	(Hour)

The base unit for time is fs, meaning that, if no units are provided, the value is assumed to be in femtoseconds.

10. **Character literals:** Single ASCII character or a string of such characters.

11. **SIGNED** and **UNSIGNED:** data types defined in the std_logic_arith package of the IEEE library.

Example 1:

```
signal x1, x2: unsigned (3 downto 0);

x1 <= "0101";  -- decimal 5
x2 <= "1101";  -- decimal 13
```

Example 2:

```
signal x1, x2: signed (3 downto 0);

x1 <= "0101";  -- decimal 5
x2 <= "1101";  -- decimal -3
```

Example 3:

```
x1 <= '0';              -- bit, std_logic, or std_ulogic value '0'
x2 <= "00001111";       -- bit_vector, std_logic_vector
x3 <= B"101111";        -- Binary representation of decimal 47
x4 <= O"57";            -- Octal representation of decimal 47
x5 <= X"2F";            -- Hexadecimal representation of decimal 47
x6 <= 1200;             -- Integer
IF ready THEN ….        -- Boolean, executed if ready=TRUE
x7 <= 1.2E-5;           -- Real
x8 <= d after 10 ns;    -- Physical
```

Example 4: Legal and illegal operations between data of different types.

```
signal a: bit;
signal b: bit_vector(7 downto 0);
signal c: std_logic;
signal d: std_logic_vector(7 downto 0);
signal e: integer range 0 to 255;
-------------------------------------------------------------------
a <= b(5);     -- legal(same scalar type: bit)
b(0) <= a;     -- legal(same scalar type: bit)
c <= d(5);     -- legal(same scalar type: std_logic)
d(0) <= c;     -- legal(same scalar type: std_logic)
a <= c;        -- illegal(type mismatch: bit x std_logic)
b <= d;        -- illegal(type mismatch: bit_vector x std_logic_vector)
e <= b;        -- illegal(type mismatch: integer x bit_vector)
e <= d;        -- illegal(type mismatch: integer x std_logic_vector)
-------------------------------------------------------------------
```

2. User-defined data types

- VHDL also allows the user to define his/her own data types.
- There are two types of user-defined data types: integer and enumerated.
- General form:

  Type type_name **is** type_definition;

1. User-defined integer types

Examples:

```
Type integer is range -2147483647 to +2147483647;
Type natural is range 0 to +2147483647;
Type my_integer is range -32 to 32;
Type student_grade is range 0 to 100;
```

2. User-defined enumerated types

Examples:

```
Type bit is ('0','1');
Type my_logic is ('0','1','Z');
Type color is (red, green, blue, white);
```

Note: Assume Red = 00

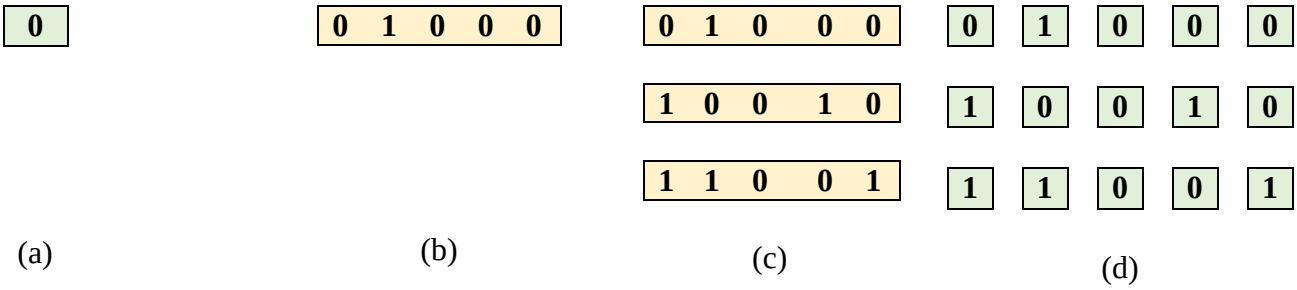
Green = 01

Blue = 10

White = 11

Array

- Arrays are collections of objects of the same type.
- They can be one-dimensional(1D), two-dimensional (2D), or (1D x 1D).
- They can also be of higher dimensions.
- Useful for modeling ROMs, RAMs, Busses.
- The construction of data arrays can be illustrated as shown below:
  - A single value (scalar) is shown in (a)
  - A vector (1D array) in (b)
  - An array of vectors (1D x 1D) in (c), and
  - An array of scalars (2D array) in (d).



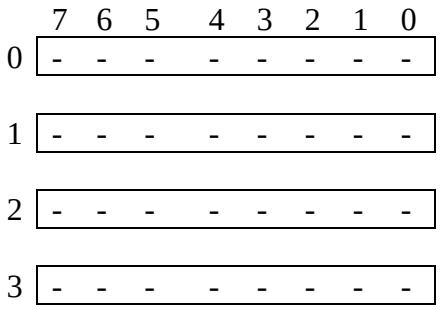
- Scalars: bit, std_logic, and Boolean.
- Vectors: bit_vector, std_logic_vector, integer, signed, and unsigned.
- To specify a new array type:

```
Type type_name is array(specification)of data_type;
Signal signal_name: type_name;
```

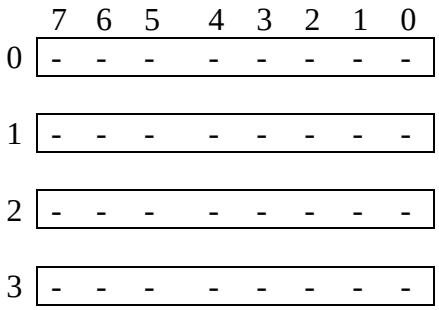
- **Example: 1D x 1D array**

```
Type row is array (7 downto 0) of std_logic;    --1D array
Type matrix is array (0 to 3) of row;           --1D x 1D array
Signal a: matrix;
Signal b: row;
Signal c: std_logic_vector(7 downto 0);
```

Is b equal to c?

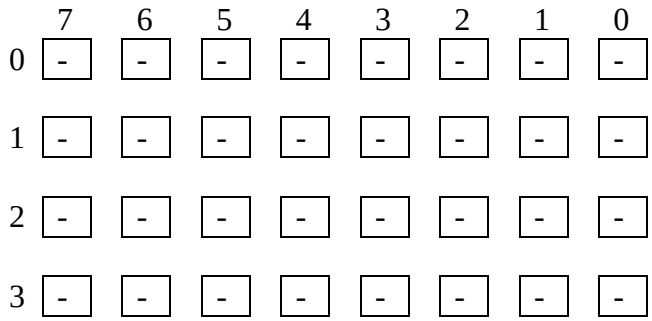


```
Type matrix is array (0 to 3) of std_logic_vector(7 downto 0); --1D array

Signal x1: matrix;
```



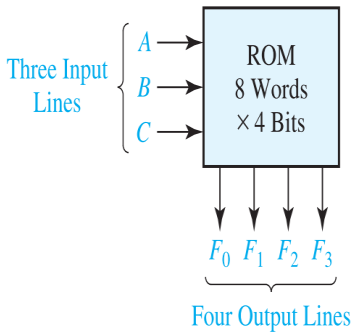
- **Example: 2D array**

```
Type matrix is array (0 to 3, 7 downto 0) of std_logic; --2D array

Signal x: matrix;
```



Example: Write VHDL code to implement the following logic functions using a 8 words × 4 bits ROM.

$F_0 = \Sigma\,(0, 1, 4, 6)$
$F_1 = \Sigma\,(2, 3, 4, 6, 7)$
$F_2 = \Sigma\,(0, 1, 2, 6)$
$F_3 = \Sigma\,(2, 3, 5, 6, 7)$

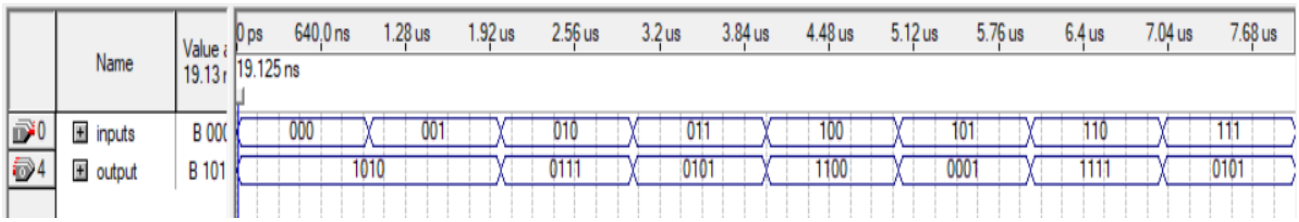


(a) Block diagram

A B C	F_0 F_1 F_2 F_3
0 0 0	1 0 1 0
0 0 1	1 0 1 0
0 1 0	0 1 1 1
0 1 1	0 1 0 1
1 0 0	1 1 0 0
1 0 1	0 0 0 1
1 1 0	1 1 1 1
1 1 1	0 1 0 1

(b) Truth table for ROM

```
1
2   ----------------------------------------ROM----------------------------------
3   Library ieee;
4   Use ieee.std_logic_1164.all;
5   ---------------------------------------------------------------------------------
6   Entity ROM is
7
8   port(inputs:in integer range 0 to 7 ;
9   ·····output:out std_logic_vector(3 downto 0));
10  End ROM;
11  ---------------------------------------------------------------------------------
12  Architecture Logic_circuit of ROM is
13
14  Type Rom_array is array(0 to 7) of std_logic_vector(3 downto 0);
15
16  Constant ROM:Rom_array:=("1010","1010","0111","0101","1100","0001","1111","0101");
17
18  Begin
19
20  ··Output <= ROM(inputs);
21  ··
22  END Logic_circuit;
23  ---------------------------------------------------------------------------------
```



H.W.: Write VHDL code to implement the following logic functions using 16 words ×3 bits ROM.

W = $A'B'C + C'D + ACD'$
X = $A'C' + B'D$
Y = $BD' + B'C'D$