Design with Hardware Description Languages (HDL)

(DHDL-94952)

Amin Mehranzadeh, Ph.D.

CE Department of
Islamic Azad University of Dezful

mehranzadeh@iaud.ac.ir
Mehran.students@gmail.com

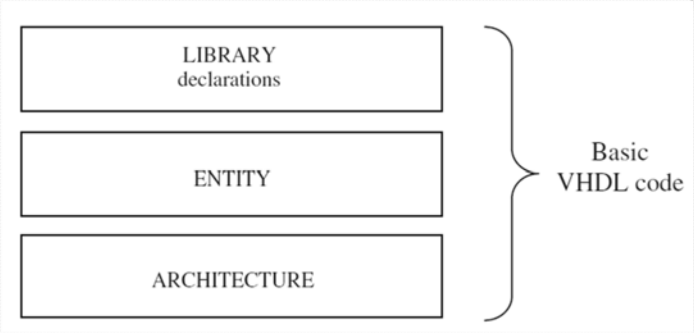
Code Structure

Fundamental VHDL Units:

- a standalone piece of VHDL code is composed of at least three fundamental sections:

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

Fundamental VHDL Units:



LIBRARY declarations

ENTITY

ARCHITECTURE

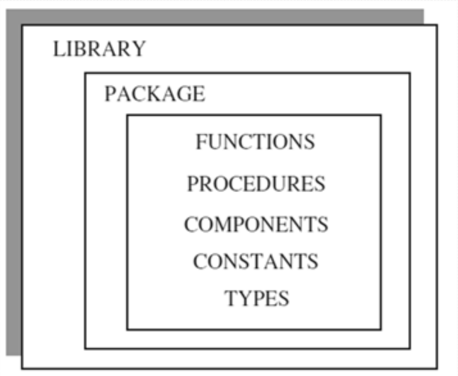
Basic VHDL code

Figure 2.1: Fundamental sections of a basic VHDL code.

LIBRARY:

- A LIBRARY is a collection of commonly used pieces of code. Placing such pieces inside a library allows them to be reused or shared by other designs.
- The typical structure of a library is usually written in the form of FUNCTIONS, PROCEDURES, or COMPONENTS, which are placed inside PACKAGES, and then compiled into the destination library.

LIBRARY:



LIBRARY

PACKAGE

FUNCTIONS

PROCEDURES

COMPONENTS

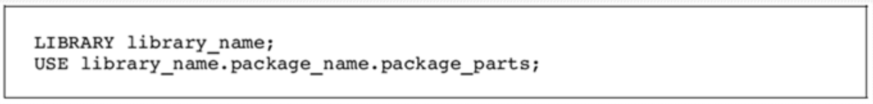
CONSTANTS

TYPES

Figure 2.2 :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;
```

Amin Mehranzadeh, Ph.D. CE Department of Islamic Azad University of Dezful

Library Declarations:

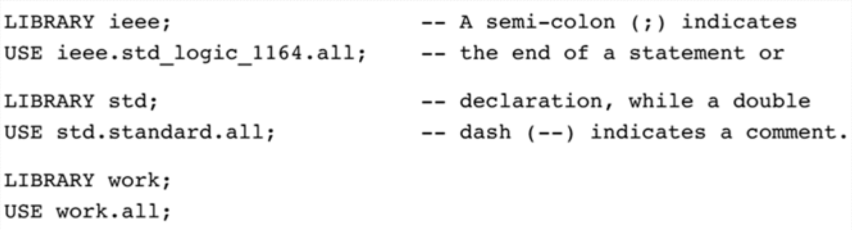
- At least three packages, from three different libraries, are usually needed in a design:

ieee.std_logic_1164 (from the ieee library),
standard (from the std library), and
work (work library).

Amin Mehranzadeh, Ph.D. CE Department of Islamic Azad University of Dezful

Library Declarations:

- Their declarations are as follows:



```
LIBRARY ieee;                     -- A semi-colon (;) indicates
USE ieee.std_logic_1164.all;      -- the end of a statement or

LIBRARY std;                      -- declaration, while a double
USE std.standard.all;             -- dash (--) indicates a comment.

LIBRARY work;
USE work.all;
```

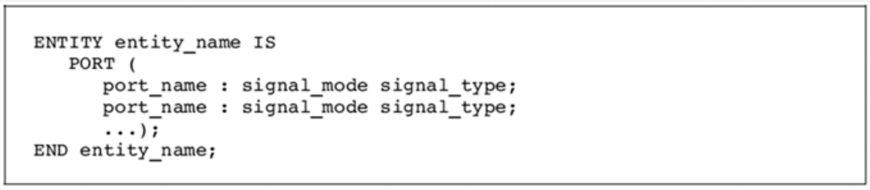
The libraries std and work shown above are made visible by default, so there is no need to declare them; only the ieee library must be explicitly written. However, the latter is only necessary when the STD_LOGIC (or STD_ULOGIC) data type is employed in the design.

Library Declarations:

- The std_logic_1164 package of the ieee library specifies a multi-level logic system.
- std is a resource library (data types, text i/o, etc.) for the VHDL design environment.
- work library is where we save our design (the .vhd file, plus all files created by the compiler, simulator, etc.).

ENTITY:

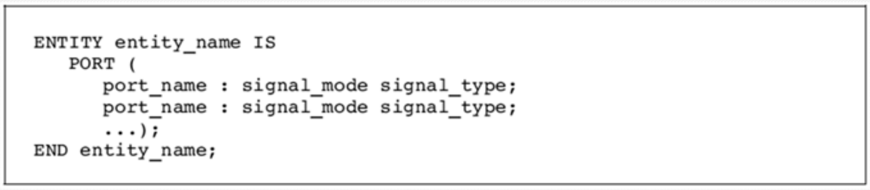
- An ENTITY is a list with specifications of all input and output pins (PORTS) of the circuit. Its syntax is shown below.



```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

ENTITY:

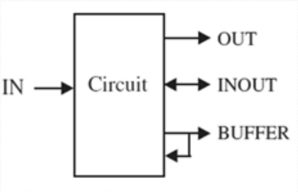
- An ENTITY is a list with specifications of all input and output pins (PORTS) of the circuit. Its syntax is shown below.



ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;

ENTITY:

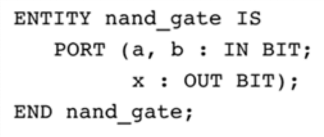
- The mode of the signal can be IN, OUT, INOUT, or BUFFER. IN and OUT are truly unidirectional pins, while INOUT is bidirectional. BUFFER is employed when the output signal must be used (read) internally.



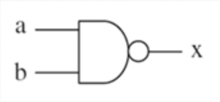
The type of the signal can be BIT, STD_LOGIC, INTEGER, etc.

ENTITY:

- Example: Let us consider the NAND gate. Its ENTITY can be specified as:



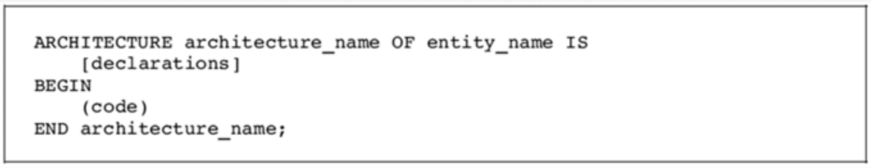
```
ENTITY nand_gate IS
   PORT (a, b : IN BIT;
         x : OUT BIT);
END nand_gate;
```



Note: the name of the entity can be basically any name, except VHDL reserved words.

ARCHITECTURE:

- The ARCHITECTURE is a description of how the circuit should behave (function). Its syntax is the following:

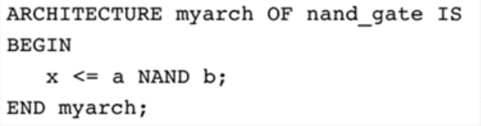


```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

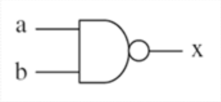
As shown above, an architecture has two parts: a declarative part (optional), where signals and constants (among others) are declared, and the code part (from BEGIN down).

ARCHITECTURE:

- Example: Let us consider the NAND gate.



```
ARCHITECTURE myarch OF nand_gate IS
BEGIN
   x <= a NAND b;
END myarch;
```

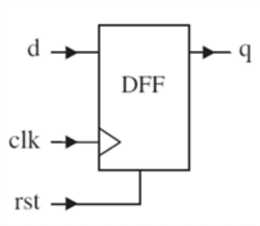


In this example, there is no declarative part, and the code contains just a single assignment.

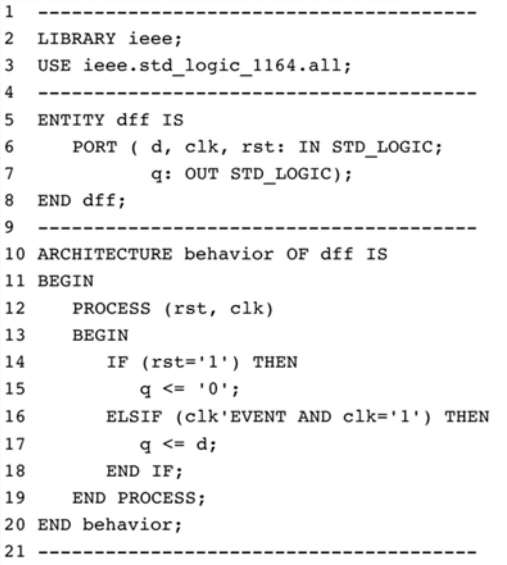
Examples:

- Example 2.1: DFF with Asynchronous Reset.

Figure shows the diagram of a D-type flip-flop (DFF), triggered at the risingedge of the clock signal (clk), and with an asynchronous reset input (rst). When rst = ‘1’, the output must be turned low, regardless of clk. Otherwise, the output must copy the input (that is, q <= d) at the moment when clk changes from ‘0’ to ‘1’ (that is, when an upward event occurs on clk).



Example 2.1 (DFF) :

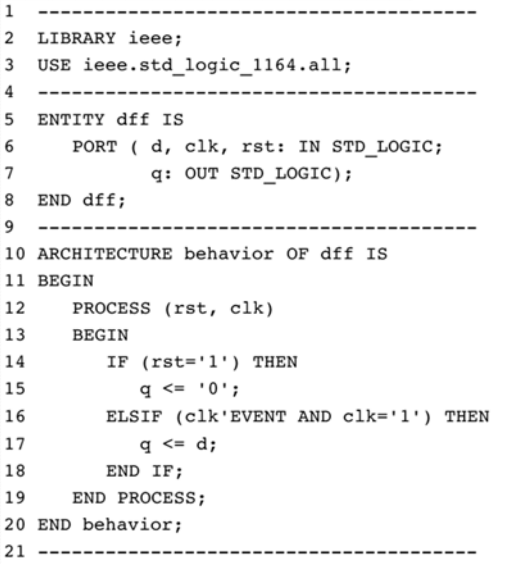


```
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
---------------------------------------
ARCHITECTURE behavior OF dff IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END behavior;
---------------------------------------
```

Note1:
One thing to remember: however, is that VHDL is inherently concurrent (contrary to regular computer programs, which are sequential), so to implement any clocked circuit (flip-flops, for example) we have to ‘‘force’’ VHDL to be sequential. This can be done using a PROCESS.

Example 2.1 (DFF) :

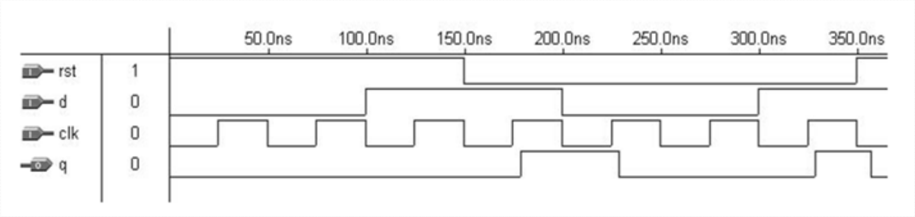
Note2:
The ‘‘<=’’ operator is used to assign a value to a SIGNAL. In contrast, ‘‘:=’’ would be used for a VARIABLE. All ports in an entity are signals by default.



```
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
---------------------------------------
ARCHITECTURE behavior OF dff IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END behavior;
---------------------------------------
```

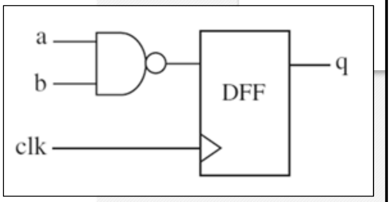
Example 2.1 (DFF) :

Simulation results:



Example 2.2 (DFF plus NAND Gate) :

```
1  ---------------------------------------
2  ENTITY example IS
3     PORT ( a, b, clk: IN BIT;
4            q: OUT BIT);
5  END example;
6  ---------------------------------------
7  ARCHITECTURE example OF example IS
8     SIGNAL temp : BIT;
9  BEGIN
10    temp <= a NAND b;
11    PROCESS (clk)
12    BEGIN
13       IF (clk'EVENT AND clk='1') THEN q<=temp;
14       END IF;
15    END PROCESS;
16 END example;
17 ---------------------------------------
```



Notice that there is no mode declaration (mode is only used in entities).

line 10 is executed concurrently with the block 11–15.