جامعة نينوى
كلية هندسة الإلكترونيات

Circuit Design with VHDL

Textbook: Volnei A. Pedroni

Submitted By: Hussein Aideen

VHDL> Introduction

- **VHDL** stands for **V**ERY **H**IGH SPEED **I**NTEGRATED **C**IRCUIT **H**ardware **D**escription **L**anguage.
- **VHSIC** is itself an abbreviation for **V**ery **H**igh **S**peed **I**ntegrated **C**ircuits.
- Describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be implemented.
- first HDL standardized, IEEE 1076 standard.

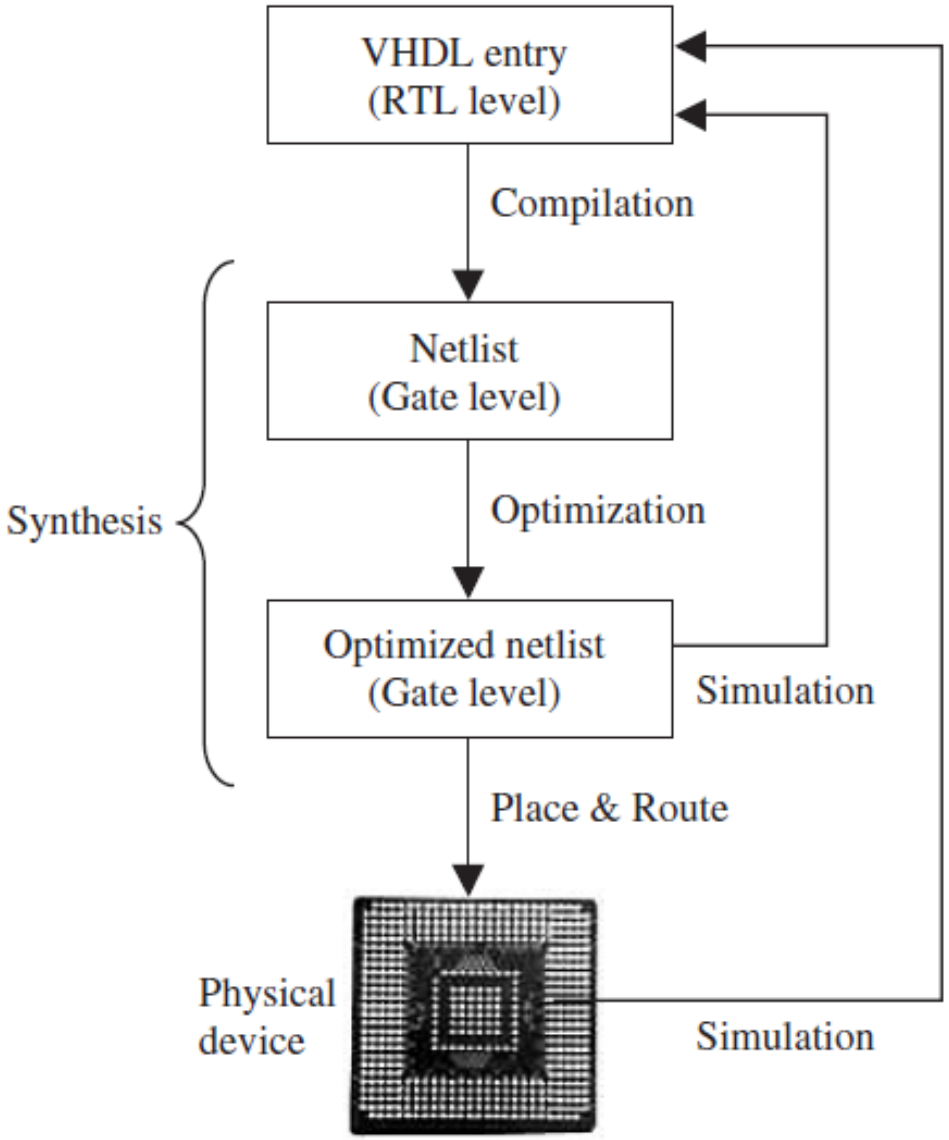
VHDL> Introduction

- Once the **VHDL** code has been written:
  - used either to implement the circuit in a programmable device (from Altera, Xilinx, Atmel, etc.)
  - or can be submitted to a foundry for fabrication of an **ASIC** chip.

- Currently, many complex commercial chips (microcontrollers, for example) are designed using such an approach.

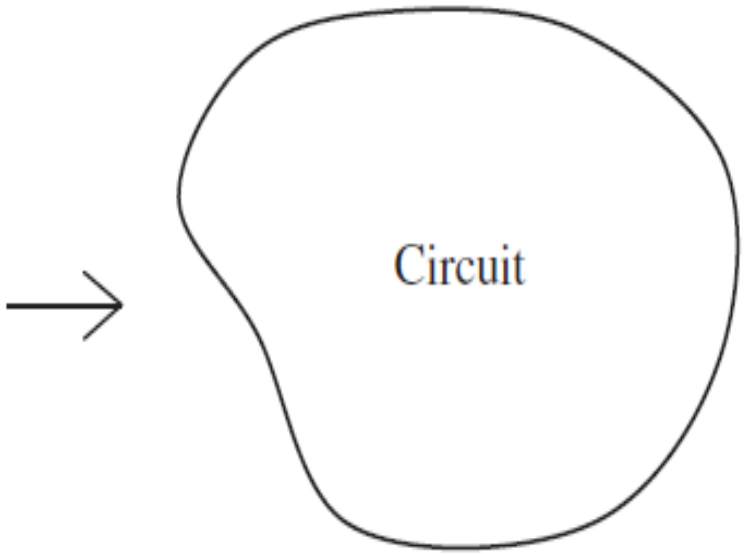
- its statements are concurrent (parallel).

VHDL> Design Flow



VHDL> Design Flow

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
      s, cout: OUT BIT);
END full_adder;
---------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```



VHDL> Code Structure

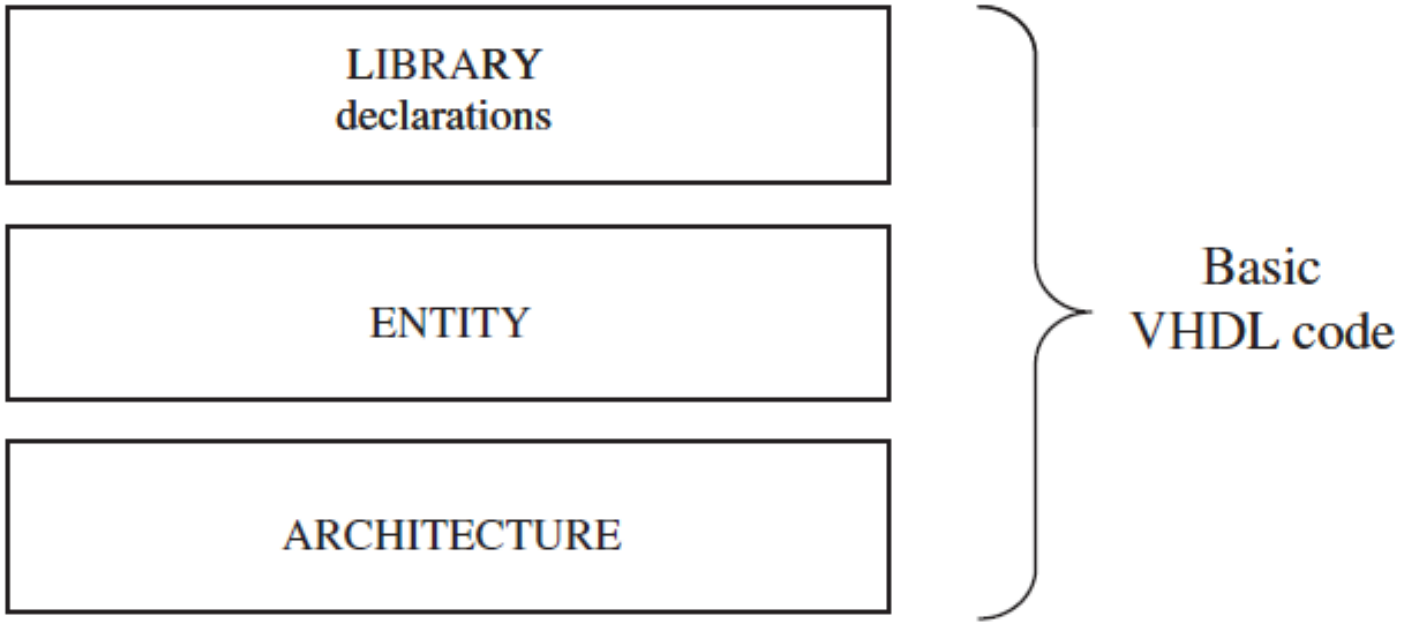


Figure 2.1
Fundamental sections of a basic VHDL code.

VHDL> Code Structure

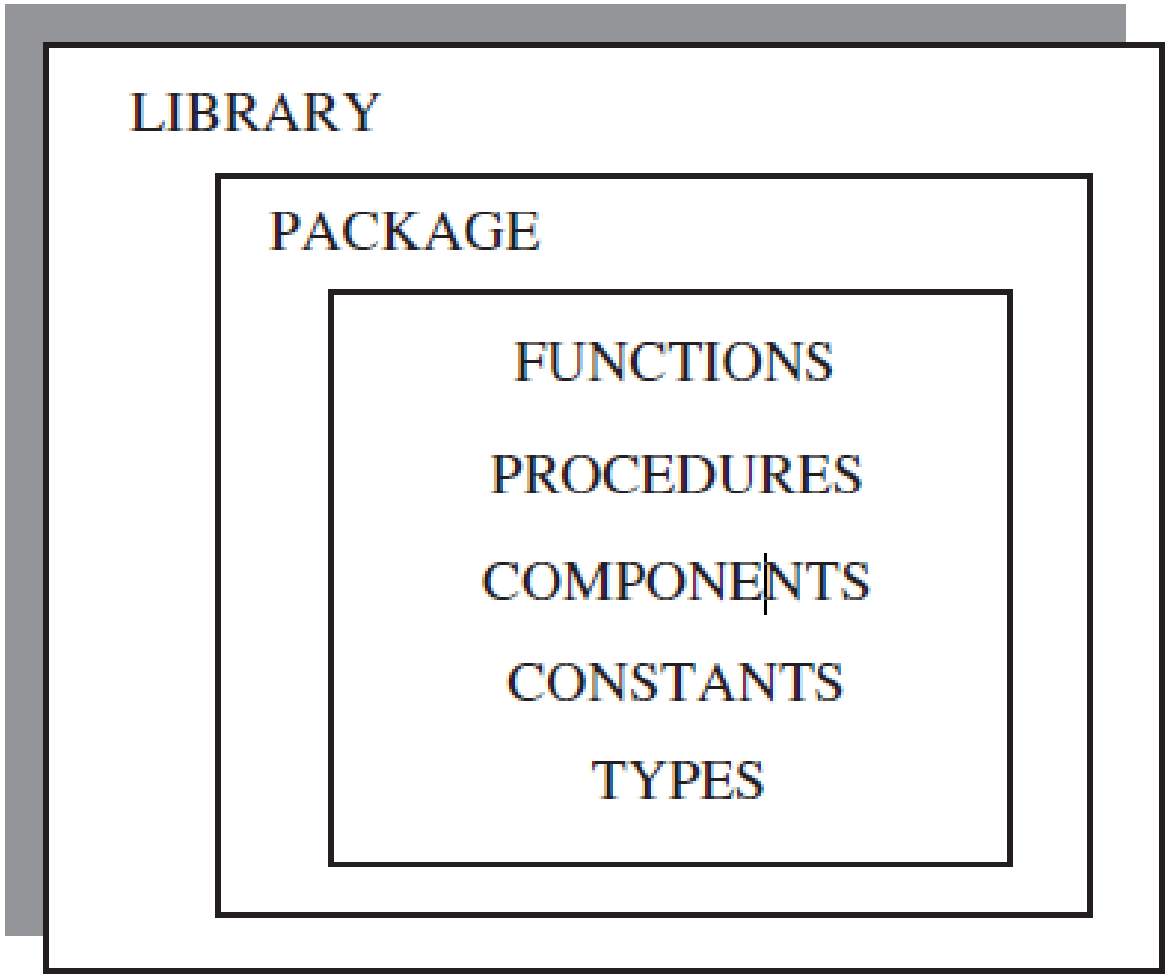
LIBRARY declarations:

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.

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

```
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;

library UNISIM;
use UNISIM.VComponents.all;
```

VHDL> Code Structure



VHDL> Code Structure

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;
```

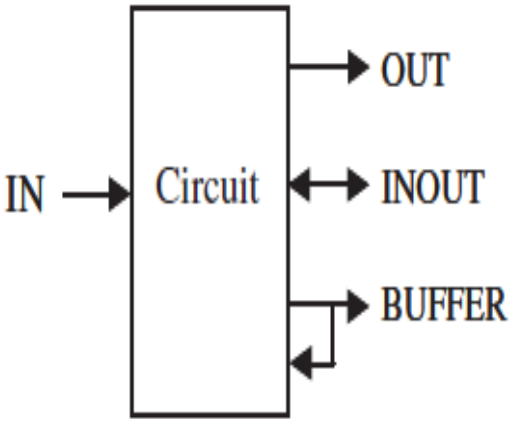
```
ENTITY nand_gate IS
PORT (a, b : IN BIT;
x : OUT BIT);
END nand_gate;
```

```
entity fulladder1 is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          s : out  STD_LOGIC;
          cout : out  STD_LOGIC);
end fulladder1;
```

VHDL> Code Structure

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

- The mode of the signal can be:
  - IN, OUT, INOUT, or BUFFER.
  - **IN** and **OUT** are truly unidirectional pins,
  - **INOUT** is bidirectional.
  - **BUFFER**, output signal must read internally.



- The type of the signal can be BIT, STD_LOGIC, INTEGER, etc. (discussed later).
- Finally, the name any name, except VHDL reserved words

VHDL> Code Structure

- 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;
```

- declarative part (optional), where signals and constants are declared.

- the **code** part (from BEGIN down)

VHDL> Introductory Examples

- **Process**

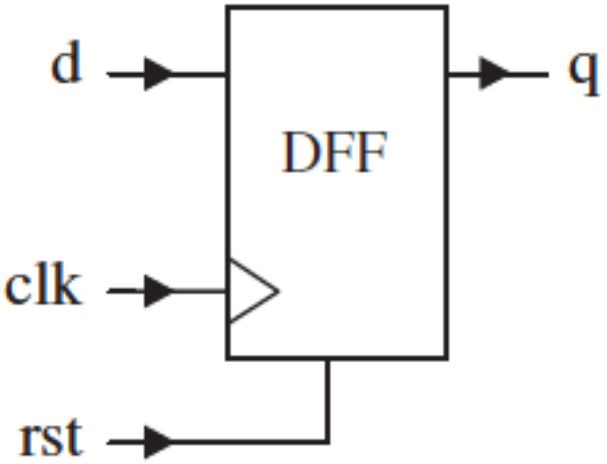
- VHDL is inherently concurrent (contrary to regular computer programs, which are sequential),
- so to implement any clocked circuit (flip-flops, for example) we have to "force" VHDL to be sequential.

```
PROCESS (                    )
BEGIN
      (sequential code)
END PROCESS;
```

VHDL> Introductory Examples

- **DFF**

- Exam: Q1) Write a VHDL code to synthesis the following circuit (DFF) shown in figure below:



VHDL> Introductory Examples

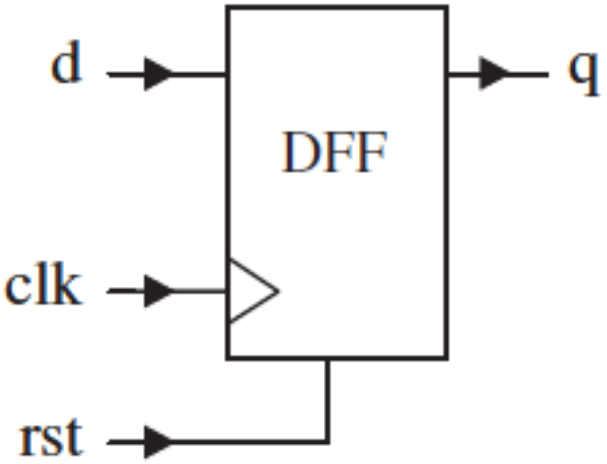
- **DFF**

```
entity dff is
    port (
        data  :in  std_logic; -- Data input
        clk   :in  std_logic; -- Clock input
        q     :out std_logic  -- Q output
    );
end entity;

architecture rtl of dff  is
begin
    process (clk) begin
        if (rising_edge(clk)) then

                q <= data;

        end if;
    end process;
end architecture;
```

VHDL> Introductory Examples

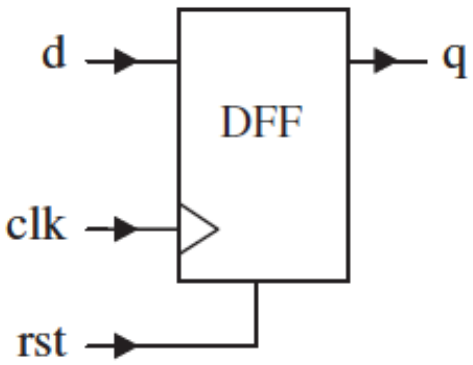
- **DFF with Asynchronous Reset**

- Exam: Q1) Write a VHDL code to synthesis the following circuit (DFF with Asynchronous Reset) shown in figure below:



VHDL> Introductory Examples

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY dff IS
6  PORT ( d, clk, rst: IN STD_LOGIC;
7  q: OUT STD_LOGIC);
8  END dff;
9  ----------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12 PROCESS (rst, clk)
13 BEGIN
14 IF (rst='1') THEN
15 q <= '0';
16 ELSIF (clk'EVENT AND clk='1') THEN
17 q <= d;
18 END IF;
19 END PROCESS;
20 END behavior;
21 ----------------------------------------
```



VHDL> J-k FF

```
entity JK_FF  is
   port( J,K: in  std_logic;
         Reset: in std_logic;
         Clock: in std_logic;
         Output: out std_logic);
end JK_FF;

architecture Behavioral of JK_FF is
   signal temp: std_logic;
begin
   process (Clock)
   begin
      if rising_edge(Clock) then
         if Reset='1' then
            temp <= '0';
         elsif (J='0' and K='0') then
            temp <= temp;
         elsif (J='0' and K='1') then
            temp <= '0';
         elsif (J='1' and K='0') then
            temp <= '1';
         elsif (J='1' and K='1') then
            temp <= not (temp);
         end if;
      end if;
   end process;
   Output <= temp;
end Behavioral;
```

VHDL> T FF

```
architecture rtl of tff_async_reset is
    signal t :std_logic;
begin
    process (clk, reset) begin
        if (reset = '0') then
            t <= '0';
        elsif (rising_edge(clk)) then
            t <= not t;
        end if;
    end process;
    q <= t;
end architecture;
```