Lab 5
VHDL Tips & Tricks

Faculty of Engineering - Cairo University
Computer Architecture Course

Objectives

- Learn
  - Easier instantiation method
  - Functions and procedures
  - Packages
  - Synthesize

Component instantiation

- **Remember to use portMapping you need**
  1. to include the component definition in the Arch
     - can replaced by calling the entity with respect to its library

```
entity smallComponent is
   Port ( a,b : in  STD_LOGIC; result: out  STD_LOGIC);
end smallComponent ;
```

```
entity bigComponent is
   Port ( x,y,s1,s0: in  STD_LOGIC;  z: out  STD_LOGIC);
end bigComponent ;
Arch test of bigComponent is

component smallComponent is
   Port ( a,b : in  STD_LOGIC; result: out STD_LOGIC);
end component ;

begin
--somecode
HA: smallComponent portmap (x,y,z);



--somecode
end
```

```
entity smallComponent is
   Port ( a,b : in  STD_LOGIC; result: out  STD_LOGIC);
end smallComponent ;
```

```
entity bigComponent is
   Port ( x,y,s1,s0: in  STD_LOGIC;  z: out  STD_LOGIC);
end bigComponent ;
Arch test of bigComponent is

 component smallComponent is
    Port ( a,b : in  STD_LOGIC; result: out STD_LOGIC);
 end component ;

begin
--somecode

 HA: smallComponent portmap (x,y,z);
 HA : entity work.smallComponent portmap (x,y,z);


--somecode
end
```

Component instantiation

- **Remember to use portMapping you need**
  1. to include the component definition in the Arch
     - can replaced by calling the entity with respect to its library
  2. to call the component using the same exact ports order as they are defined
     - can be replaced by using the "=>" operator

```
entity smallComponent is
   Port ( a,b : in  STD_LOGIC; result: out  STD_LOGIC);
end smallComponent ;
```

```
entity bigComponent is
   Port ( x,y,s1,s0: in  STD_LOGIC;  z: out  STD_LOGIC);
end bigComponent ;
Arch test of bigComponent is
begin
--somecode

HA : entity work.smallComponent portmap (x,y,z);

--somecode
end
```

```
entity smallComponent is
   Port ( a,b : in  STD_LOGIC; result: out  STD_LOGIC);
end smallComponent ;
```

```
entity bigComponent is
   Port ( x,y,s1,s0: in  STD_LOGIC;  z: out  STD_LOGIC);
end bigComponent ;
Arch test of bigComponent is
begin
--somecode
HA : entity work.smallComponent portmap ~~(x,y,z);~~
          ( b => y ,
           result=>z ,
           a => x );

--somecode
end
```

Functions

- Parameters are passed by Value
- Only one return value
- Functions are sequential .
- can be called anywhere
- Used for house keeping not for describing type hierarchy

```
Function maxval(a1,a2:integer) return integer is
           Variable result : integer ;
  Begin
        If a1 > a2 then
               Result :a1;
        Else
               Result := a2;
        End if;
     Return result;
    End maxval;
```

```
entity bigComponent is
   Port ( x, y, s1, s0: in  integer;  z: out  integer);
end bigComponent ;
Arch test of bigComponent is
  Function maxval(a1,a2:integer) return integer is
           Variable result : integer ;
    Begin
        If a1 > a2 then
             Result :a1;
        Else
             Result := a2;
        End if;
      Return result;
     End maxval;
begin
--somecode
process(x, y, z)
begin
   z <= maxval(x, y);

end
--somecode
end
```

Procedures

- Parameters are passed by reference & hence must have direction.
- Procedures are sequential .

```
Procedures dff(signal Rst,Clk: in std_logic;
Signal D : in std_logic_vector(7 downto 0);
Signal Q : out std_logic_vector(7 downto 0))  is
Begin
        If Rst = ‘1’ then Q <= (others =>’0’);
        Elsif  Clk=‘1’ and Clk’event then
                Q<=D;
        End if;
End dff;
```

```
entity bigComponent is
   Port ( rst, clk, s1, s2: in  std_logic;
   w: in std_logic_vector(7 downto 0) ;
    z: out  std_logic_vector(7 downto 0));
end bigComponent ;
Arch test of bigComponent is
 Procedures dff(signal Rst,Clk: in std_logic;
 Signal D : in std_logic_vector(7 downto 0);
 Signal Q : out std_logic_vector(7 downto 0))  is
  Begin
       If Rst = ‘1’ then Q <= (others =>’0’);
       Elsif  Clk=‘1’ and Clk’event then
            Q<=D;
       End if;
 End dff;
begin
--somecode
process(rst, clk, w, z)
begin
      dff(rst, clk, w, z);
end
end
```

WARNING!!

- Beware Procedures and functions seems easy but they **don't** always synthesize use them **wisely**

Packages … definition

```
Library ieee;
Use ieee.std_logic_1164.all
Package mine is
Constant PER : time := 50 ns;
Subtype byte is std_logic_vector(7 downto 0);
Constant CLEARS : byte := “00000000”;
Procedure dff(signal Rst,Clk: in std_logic; Signal D : in byte;
Signal Q : out byte);
…..
End mine;
```

Packages … body

```
Package body mine is
     Procedures dff(signal Rst,Clk: in std_logic; Signal D : in byte;Signal Q : out
byte) is
   Begin
          If Rst = '1' then Q <= CLEARS;
          Elsif  Clk='1' and Clk'event then
                   Q<=D;
          End if;
     End dff;
 …..
End mine;
```

Packages … usage

```
Library ieee;
Use ieee.std_logic_1164.all
Use work.mine.all
Entity
    ………………

Architecture
 …………………..
 dff(…); ← like software calling of function
End Architecture;
```

WARNING!! AGAIN

- Functions and packages make it easy to get **unsynthesized** code OR **very big hardware**
- Each time you call a function, a dedicated hardware is synthesized corresponding to it. If you call a function 10 times, the hardware will have 10 instances of that function.

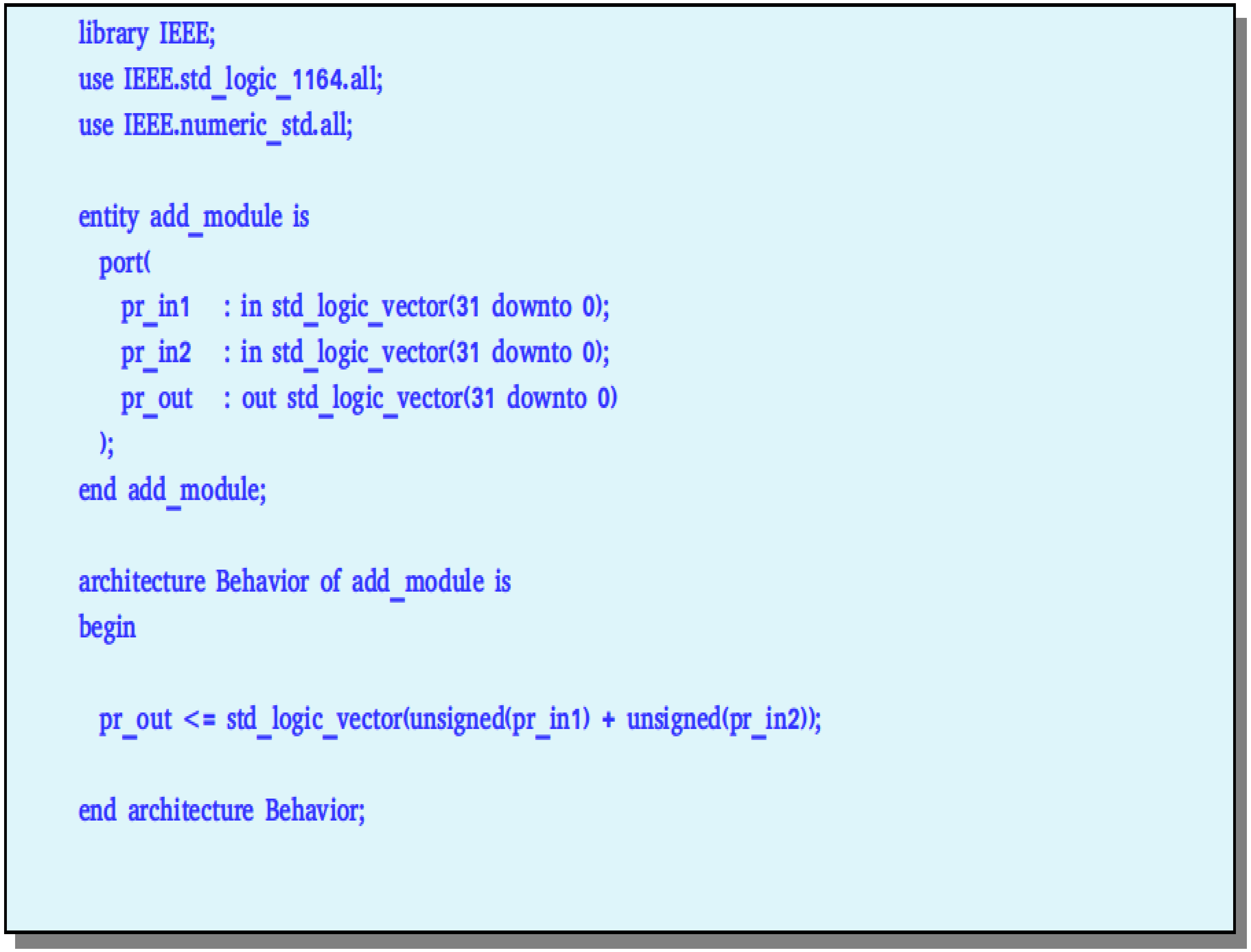
Predefined Operators

- Operators are used in expressions involving signal, variable or constant object types. The following are the types of operators as defined in the VHDL language.
-

- **logical :** AND, OR, NAND, NOR, XOR, XNOR, NOT
- **relational :** =, /=, <, <=, >, >=
- **shift :** SLL, SRL, SLA, SRA, ROL, ROR
- **addition :** +, -, &
- **unary :** +, -
- **multiplying:** *, /, MOD, REM.

Predefined Operators:

Addition Example



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity add_module is
  port(
    pr_in1    : in std_logic_vector(31 downto 0);
    pr_in2    : in std_logic_vector(31 downto 0);
    pr_out    : out std_logic_vector(31 downto 0)
  );
end add_module;

architecture Behavior of add_module is
begin

  pr_out <= std_logic_vector(unsigned(pr_in1) + unsigned(pr_in2));

end architecture Behavior;
```