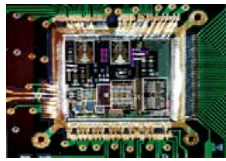
VHDL

Instructor: Dr. Saraju P. Mohanty

In this Lecture

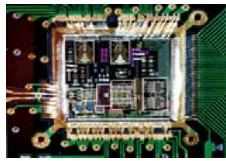
- Packages
- Resolved Signals
- Generic Constants
- Components and Configurations
- Generate Statements
- Guards and Blocks
- Files and Input/Output
- Attributes and Groups

Note: The slides are from text book or reference book authors or publishers.



Libraries and Packages

- VHDL provides two constructs for declaring types and sub-programs.
  - Libraries
  - Packages
- The objective is to group together declarations in one place.
- The grouped declarations are utilized for modeling a large application.



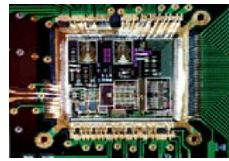
Libraries

- A library refers to a collection of declarations (type, entity, sub-program) and their implementations (architecture, sub-program body).

- The actual specification of a library varies from one simulation package to another.

- A library can with its collections of declarations and design units can be made visible as follows:

  Lib_clause <=
  library *id* { , … } **;**



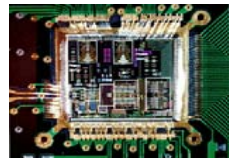
Special Library : Work

•The identifier “work” is a special library that maps on to the present directory.

•All the design units in the present directory are visible to all models.

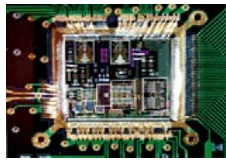
•Hence, an explicit declaration of “work” library is not required.

•However, one needs to specify the “work” when accessing declarations and design units in other files.



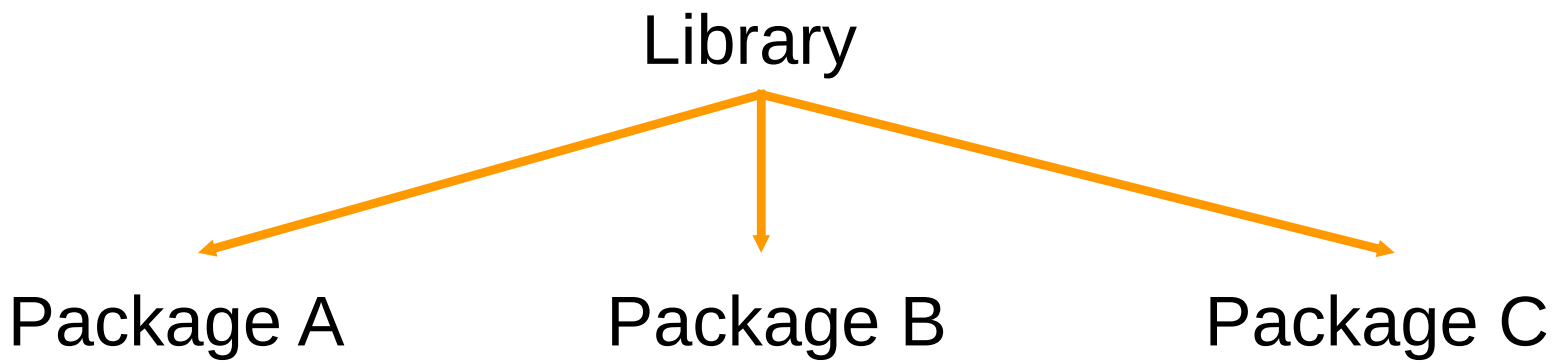
Example: Work Library

```
entity test_bench is
end entity test_bench;

architecture test_reg4 of test_bench is
    signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : bit;
begin
    dut : entity work.reg4(behav)
        port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );
    stimulus : process is
    begin
        d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';  wait for 20 ns;
        en <= '0';  clk <= '0';  wait for 20 ns;
        en <= '1';  wait for 20 ns;
        clk <= '1';  wait for 20 ns;
        d0 <= '0';  d1 <= '0';  d2 <= '0';  d3 <= '0';  wait for 20 ns;
        en <= '0';  wait for 20 ns;
        …
        wait;
    end process stimulus;
end architecture test_reg4;
```



Packages

- A library may include one or more packages.



- Alternately, a package could be defined as a stand alone design unit.

Predefined Package Standard

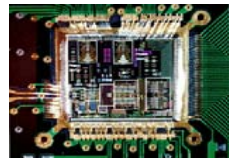
•VHDL includes declarations of predefined types and operators that are stored in the library “std”.

•A user is not required to explicitly declare the standard library.

•The following declaration is implicit for each VHDL model file: library std, work; use std.standard.all;
NOTE: It does not include “use work.all”;

•IEEE has defined packages for modeling and synthesizing hardware.

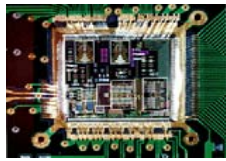
```
library ieee;
use ieee.std_logic_1164.std_ulogic;
```

IEEE Multi-valued Logic System

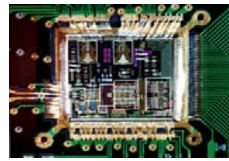
- Multi-valued logic with 9 possible values:

```
type std_ulogic is ( 'U', -- uninitialized
                     'X', -- forcing unknown
                     '0', -- forcing 0
                     '1', -- forcing 1
                     'Z', -- high impedance
                     'W', -- weak unknown
                     'L', -- weak 0
                     'H', -- weak 1
                     '-', -- don't care
                     );
```

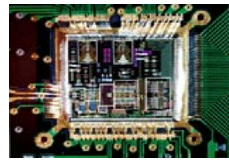
VHDL Mathematical Packages

- Real number mathematical package and complex number mathematical package.
- Real number package defines various constants and functions.
- Complex number package defines complex numbers and functions that operate on complex numbers.



Resolved Signals

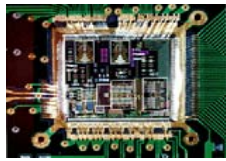
- So far we have assumed only one driver per signal.
- That is, only one process can apply signal assignments to a signal.
- Resolved signals handle the case when there are more than one driver for a signal.
- A resolved signal includes a function called resolution function in its definition.
- This is needed to model when a node is connected with multiple outputs.



Resolved Signals: Example

```
architecture b of e is
    signal x: bit;
begin
    p1: process is
    begin
        …
        x <= '1';
        …
    end process;
    p2: process is
    begin
        …
        x <= '0';
        …
    end process;
```

- p1 and p2 are said to have "drivers" for the signal x.
- Since, both p1 and p2 are driving x, x is required to be a resolved signal.



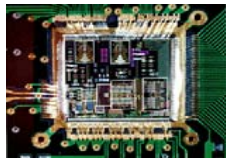
Resolution function

•Resolved signals are characterized by a resolution function.

•The resolution function takes as input an unconstrained array of signal transaction values and returns the resolved value for the signal.

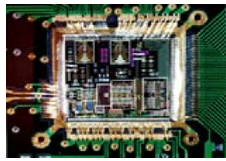
•The resolution function is invoked when all the processes are suspended.

•The scheduled transactions from the various signal drivers are passed to the function and it determines the final value.



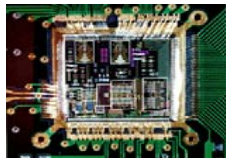
Resolution function: Example

```
type tri_state_logic is (‘0’,’1’,’Z’);
type tri_state_logic_array is array (integer range <>) of tri_state_logic;

-- Declaration of Resolution Function
function resolve_tri_state_logic (values: in tri_state_logic_array) return tri_state_logic;

-- Addition of Resolution Function before signal type
signal s1: resolve_tri_state_logic tri_state_logic;
```

Guards and Blocks

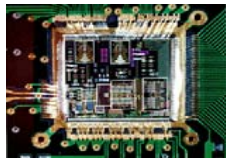
- Guarded signal is a type of resolved signal.
- Blocks, guarded signals can cause automatic disconnection of drivers.
- The main use of the guard expressions in a block statement is to make guarded signal assignments.
- If the target of a concurrent signal assignment is a guarded signal, we must use a guarded signal assignment.



Guards and Blocks: Example

```
reg_read_selector: block (reg_sel = '1' and read =
    '1') is
begin
    dbus <= reg0 when reg_addr = '0' else
    reg1 when reg_addr = '1' else
    "ZZZZZZZ";
end block;
```

- Guard signal is implicitly declared within the block. Its values is updated when a transcation occurs on either reg_sel or read.

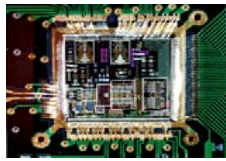


Generic Constants

- This is a mechanism for providing parameterized models.
- Generic interface list included in the declaration to define formal generic constants that parameterize the entity.
- Example: (Parameterizing Behavior)

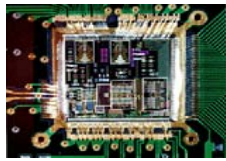
```
entity and2 is
        generic (Tpd: time);
        port (a, b : in std_logic; y: out std_logic);
end and2;
architecture simple of and2 is
begin
          y <= a and b after Tpd;
 end simple;
Gate1: entity work.and2(simple)
    generic map (Tpd => 2ns)
    -----
```

- Visible of the generic constant extends from the end of generic interface to the end of entity declaration and the corresponding architecture body.



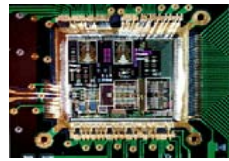
Generic Constants: Parameterizing Structures

```
entity adder is -- entity
    generic (width : integer); -- bit width
    port ( a : in std_logic_vector (width-1 downto 0);
    b : in std_logic_vector (width-1 downto 0);
    sum : out std_logic_vector (width-1 downto 0) );
end adder;

architecture structure of adder is -- architecture
    --------
    --------
end architecture structure;

add1: entity work.adder(structure) -- creates an instance
    generic map (width => 4)
    port map ( …. );
```

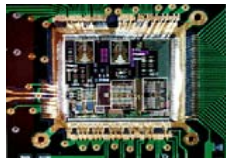
Components and Configurations

- During design of a large circuits, it is possible that various parts of the design are in different stages of completion.
- VHDL “component” is a mechanism for specifying virtual parts (or components).
- The top level design can then be described to be constructed of “virtual” components.
- As the design progresses, various sub-parts are completed; sub-parts are no longer “virtual”.
- Configuration is a mechanism for mapping a component to an actual entity-architecture pair.



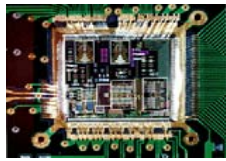
Design Flow

- Partition design in to sub-parts.
- Define the interface between various sub-parts.
- Construct components to specify the sub-parts.
- Design actual implementations for the components.
- Use configuration to integrate and simulate.



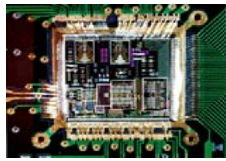
Component Declaration and Instantiation

- A component is declared inside an architecture or package.
- A component declaration is almost identical to an entity declaration.
- A component is instantiated inside the architecture body as a concurrent statement.
- A component label is necessary to identify the component instance.



Components: Example

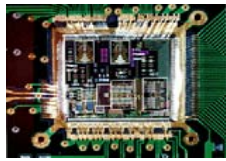
```
architecture struct of reg4 is
    component ff is
        port (clk,clr,d: in bit; q: out bit);
    end component ff;
    begin
        bit0: component ff
            port  map  (clk=>clk,clr=>clr,d=>d(0),q=>q(0));
        bit1: component ff
            port map (clk=>clk,clr=>clr,d=>d(1),q=>q(1));
        ……
end architecture;
```

Configurations

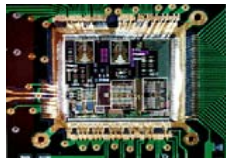
- A configuration is a design unit like entity, package etc. Hence, it can be declared by itself.

```
configuration reg4_gate_level of reg4 is
    for struct
        for bit0,bit1 : ff
          use entity flipflop(behavior);
        end for;
        for others : ff
          use entity flipflop(struct);
      end for;
    end for;
end;
```

Generate Statement

- Useful when design can be expressed as repetition of some subsystems.
- Subsystem is described once and repeatedly instantiated.
- This is a concurrent statement containing statements that are to be replicated.

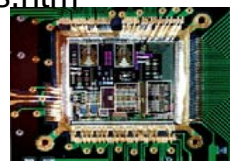


Generate Statement: Example

```
architecture generate_example of my_entity is
    component RAM16X1
    port(A0, A1, A2, A3, WE, D: in std_logic; O: out
    std_logic);
    end component;
begin
     . . .
    RAMGEN: for i in 0 to 3 generate
    RAM: RAM16X1 port map ( . . . );
    end generate;
    . . .
end generate_example;
```

- When evaluated the generate statement will generate four unique instances of component RAM16X1.

Source:http://www.acc-eda.com/vhdlref/refguide/language_overview/concurrent_statements/generate_statements.htm



Files

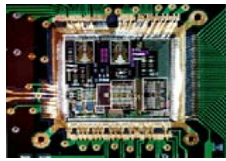
•In all the testbenches we created so far, the test stimuli were coded inside each testbench.

•Hence, if we need to change the test stimuli we need to modify the model or create a new model.

•Input and output files can be used to get around this problem.

•Files can be used to provide long-term data storage, beyond the life time of one simulation.

•VHDL provides specialized file operation for working with text files.

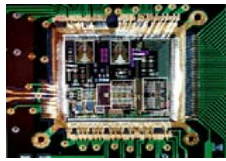


File Reading

Given a file definition, VHDL implicitly provides the following subprograms:

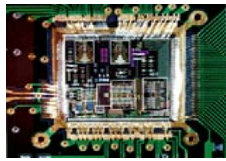
```
type file_type is file of element_type;

procedure read ( file f: file_type; value : out element_type;
                                    length : out natural);
function endfile ( file f: file_type ) return boolean;
```

If the length of the element is greater than the length of the actual data on the file, it is placed left justified in the element.



File Reading: Example

```
p1: process is
      type bit_vector_file is file of bit_vectors;
      file vectors: bit_vector_file open read_mode is “vec.dat”;
      variable next_vector : bit_vector (63 downto 0);
      variable actual_len: natural;
begin
      while not endfile(vectors) loop
            read (vectors,next_vector,actual_len);
            if actual_len > next_vector’length then
                  report “vector too long”;
            else
                  for bit_index in 1 to actual_len loop
                        ….
                  end loop;
            end if;
      end loop;
      wait;
end process;
```

File Writing

•Given a file definition, VHDL implicitly provides the following subprograms:

```
type file_type is file of element_type;
procedure write ( file f: file_type; value : in element_type);
```

•Write a process description that writes the data of integer type from an input signal to a file.

- Assume that input signal “s1” is an “in” port of the top level entity.
- Assume the file name to be “out.dat”.

```
P1: process (s1) is
    type integer_file is file of integer;
    file out_file: integer_file open write_mode is “out.dat”;
begin
        write (out_file,s1);
end;
```