EE 595

Part II

Design Units in VHDL
Design Units

- There are five types of design units in VHDL:
  - Entities
  - Architectures
  - Packages
  - Package Bodies
  - Configurations

- Entities and architectures are the only two design units that you must have in any VHDL design descriptions.

- Packages and configuration are optional.
Design Units (cont’d)

- Primary Design Units
  1. Entity Declaration
  2. Package Declaration
  3. Configuration Declaration

- Secondary Design Units
  1. Architecture Body
  2. Package Body.
**VHDL Design Units**

- **Entity**
  Primary unit of VHDL designs. It is the design’s interface to the outside world. Even the top most level of a hierarchy design must have an entity. Entities define I/O ports and timing information (generics) but can also used to do complete setup/hold checking.

- **Architecture**
  Describes behavior and/or structure of a specific entity. One entity can have many architectures associated with it but only one can be used with a given entity during simulation. Architectures are always compiled after compiling the entities they reference.
VHDL Design Units (cont’d)

- **Configuration**
  Used to bind entities to architectures for simulation. Required for simulating structural designs (ones with underlying components - netlists) but is not required for purely behavioral designs. Configurations are generally the final design unit to be compiled.

- **Package**
  “toolbox” (similar to a C “include file”) that can contain the declaration for common data types, constants, subprograms, and component declarations. Packages must be compiled first, before anything that depends on them (there are both standard packages and user defined packages)
VHDL Design Units (cont’d)

- PACKAGE BODY
  The code for the bodies of the subprograms declared in the PACKAGE. Has the same name as Package so there can be only be one Package Body for each Package. Can be compiled at any time after Package; other design unit’s dependencies are on the Package, not the Package body.
Entity Declaration

Design Entity-Overall View

Internal View (ARCHITECTURE)

The External View (ENTITY)

Entity Input Ports

Entity Output Ports

The BLACK BOX

A

B

Z

California State University
Northridge

EE 595 EDA / ASIC Design Lab
The primary purpose of the entity is to declare the signals in the component’s interface:
- The interface signals are listed in the PORT clause.
  - In this respect, the entity is akin to the schematic symbol for the component.
- Additional entity clauses and statements will be introduced later in this and subsequent modules.

entity HALF_ADDER is
  generic (PROP_DELAY : TIME := 10 ns);
  port (X, Y, ENABLE : in Bit;
     CARRY, RESULT : out Bit);
end HALF_ADDER;
Entity Declaration (cont’d)

- PORT clause declares the interface signals of the object to the outside world.
- Three parts of the PORT clause:
  - Name
  - Mode
  - Data type
- Example PORT clause:

  ```
  port (SIGNAL_NAME : MODE DATA_TYPE);
  port (INPUT : in Bit_Vector(3 DOWNTO 0);
       READY, OUTPUT : out Bit);
  ```

- Note port signals (i.e. ‘ports’) of the same mode and type or subtype may be declared on the same line.
The port mode of the interface describes the direction in which data travels with respect to the component.

The five available port modes are:
- **In** - read only (expression, if expression, wait on, wait until expression)
- **Out** - data travels out of this port, write only (signal assignment)
- **Buffer** - read/write, data may travel in either direction, but only one signal driver may be on at any one time
- **Inout** - read/write, data may travel in either direction with any number of active drivers allowed; requires a Bus Resolution Function
- **Linkage** - direction of data flow is unknown
Generics may be used for readability, maintenance and configuration.

Generic clause syntax:

```
generic (GENERIC_NAME : type [:= default_value]);
```

- If optional default_value missing in generic clause declaration, it must be present when component is to be used (i.e. instantiated).

Generic clause example:

```
generic (MY_ID : Integer := 37);
```

- The generic My_ID, with a default value of 37, can be referenced by any architecture of the entity with this generic clause.
- The default can be overridden at component instantiation.
Entity Declaration (cont’d)

- General Mechanism to pass instance specific data
- Like a function parameter, bus width.
- Declared in the entity declaration
- Place holders for information that is likely to change, data passed is constant, cannot be modified
- Can change generic value several places during design process
- Good for timing and bus width specifications
Architecture

- Describes behavior of entity
- Must be associated with a specific entity
- Single entity can have many architectures
Architecture (cont’d)

- Describe the operation of the component
- Consist of two parts:
  - Declarative part -- includes necessary declarations, e.g.:
    - type declarations, signal declarations, component declarations, subprogram declarations
  - Statement part -- includes statements that describe organization and/or functional operation of component, e.g.:
    - concurrent signal assignment statements, process statements, component instantiation statements

```vhdl
architecture HALF_ADDER_D of HALF_ADDER is
  signal XOR_RES : Bit;  -- architecture declarative part, visible only to architecture.
begin  -- begins architecture statement part
  CARRY <= ENABLE and (X and Y);
  RESULT <= ENABLE and XOR_RES;
  XOR_RES <= X xor Y;
end HALF_ADDER_D;
```
Architecture (cont’d)

architecture ARCH_NAME of ENTITY_NAME is
    <Declaration Section>
    (Component Declarations, Local Data Types
     Subprograms, Constants, Internal Signals)
begin
    <Statement Section>*
    (Component Instantiations, Processes, Signal Assignments, Blocks and
     Guarded Blocks Procedure Calls, Assertion Statements)
end ARCH_NAME; -- arch_name is optional

* Contains only concurrent statements which then execute when events occur on their sensitive signals (sensitivity list)
Entity And Architecture

Circuit

Shifter

INIT[8]
LOAD
CLK
RST
TEST[8]

Data
Load
Clock
Rst

COMPARATOR

EQ

A
B

LIMIT

EE 595  EDA / ASIC Design Lab
Design Example

```vhdl
-- Eight-Bit Comparator

library IEEE;
use IEEE.Std_Logic_1164.all;
entity COMPARATOR is
  port (A, B: in std_logic_vector (0 to 7);
        EQ: out std_logic);
end COMPARATOR;

architecture ARCH of COMPARATOR is
begin
  EQ <= '1' when (A=B) else '0';
end ARCH;
```
The architecture body defines the internal view of the component.

The internal view can be described with different styles providing equivalent functionality in top-down design approach.

- **Behavioral**: as a set of sequential statements
- **Dataflow or Register Transfer Level**: as a set of concurrent assignment statements.
- **Structural**: as a set of interconnect components
- A mix of the above.
Behavioral Description

- VHDL provides two styles of describing component behavior
  - Data Flow: concurrent signal assignment statements
  - Behavioral: processes used to describe complex behavior by means of high-level language constructs
    - variables, loops, if-then-else statements, etc.
- A behavioral model may bare little resemblance to system implementation
  - Structure not necessarily implied
Behavioral Description (cont’d)

- Behavior style of modeling specifies the behavior of an entity as a set of sequential statements which are executed in specified order.
- This set of statements are specified inside a process statement.
- These sequential statements inside process do not explicitly specify the structure of the entity but nearly specifies the functionality.
- A process as a whole is a concurrent statement that can appear within an architecture body.
Concurrent Statement

- Basic granularity of concurrency is the process
  - Processes are executed concurrently
  - Concurrent signal assignment statements are one-line processes

- Mechanism for achieving concurrency:
  - Processes communicate with each other via signals
  - Signal assignments require delay before new value is assumed
  - Simulation time advances when all active processes complete
  - Effect is concurrent processing
    - I.e. order in which processes are actually executed by simulator does not affect behavior

- Concurrent VHDL statements include:
  - Block, process, assert, signal assignment, procedure call, component instantiation
Sequential Statement

- Statements inside a PROCESS execute sequentially

```vhdl
architecture SEQUENTIAL of TEST_MUX is
begin
    SELECT_PROC : process (X,Y)
    begin
        if (SELECT_SIG = '0') then
            Z <= X;
        elsif (SELECT_SIG = '1') then
            Z <= Y;
        else
            Z <= "XXXX";
        end if;
    end process SEQUENTIAL;
end SEQUENTIAL;
```
Concurrent Versus Sequential Statement
A VHDL process is

- Key structure in behavioral VHDL modeling.
- Only means by which the executable functionality of a component is defined.
- Exists inside an architecture.
- Statements within a process are executed sequentially.
- All processes within an architecture body are executed concurrently.

architecture BEHAVE of CLK_COMP is
begin
    process
        variable PERIODIC: bit := '1';
        begin
            if EN = '1' then
                PERIODIC := not PERIODIC;
            end if;
            CK <= PERIODIC;
            wait for 1 us;
        end process;
    end BEHAVE;
VHDL Process (cont’d)

- A process statement has:
  - A declarative part between keyword process and begin
  - A statement part between keywords begin and end process
  - The statements appearing within the statement part are sequential statements which include variable assignment statements, signal assignment statements, procedure calls, wait statements. “if” clauses, “while” loops, assertion statements, etc.
  - Process statement is invoked whenever there is an event on any signal in the sensitivity list.
  - The use of “process_label” at the beginning and end of the process is optional but recommended to enhance code readability.
VHDL Process (cont’d)

[ PROCESS_LABEL : ] process [(SENSITIVITY LIST)]
  process_declarations
begin
  process_statements
end process [PROCESS_LABEL];

variables, constants, aliases, files and many other things but no signal declarations
Process Example

architecture EXAMPLE of FULL_ADDER is
  -- nothing needed in declarative part.
begin
  SUMMATION: process (A, B, Cin)
    begin
      SUM <= A xor B xor Cin;
    end process SUMMATION;
  CARRY: process (A, B, Cin)
    begin
      Cout <= (A and B) or (A and Cin) or (B and Cin);
    end process CARRY;
end EXAMPLE;

• The SUM and CARRY process executed if there is a value change on A, B, Cin.
• SUM and CARRY processes executed concurrently.
• This model does not exploit explicit time i.e. there are no AFTER phrases or “wait for” statements. This model is purely functional. If timing is important, either AFTER clauses or “wait for” statements are used.

Note: If wait statements are used in a process, the process cannot have a sensitivity list.
Dataflow Description

- In dataflow description:
  - The flow of data from signal to signal and input to output is specified without the use of sequential statements.
  - The flow of data through the entity is expressed primarily using concurrent statements.
  - The structure of the entity is not explicitly specified in this modeling style, but it can be implicitly deduced.

Some authors distinguish between behavioral and dataflow architectures, others lump them together as behavioral description. The primary difference is that one uses process (described in later sections) and other does not. They are both clearly not structural.
This is a dataflow architecture because it specifies how data will be transferred from signal to signal and input to output using concurrent statements without use of sequential statements. It consists of one signal declaration.
Structural Description

- Pre-defined VHDL components are ‘instantiated’ and connected together
- Structural descriptions may connect simple gates or complex, abstract components
- Mechanisms for supporting hierarchical description
- Mechanisms for describing highly repetitive structures easily
As a third method, a structural description can be created from pre-described components.

These gates can be pulled from a library of parts.
Structural Description Example (cont’d)

architecture HALF_ADDER_C of HALF_ADDER_NTY is

component AND2
  port (IN0, IN1 : in bit;
       OUT0 : out bit);
end component;

component AND3
  port (IN0, IN1, IN2 : in bit;
       OUT0 : out bit);
end component;

component XOR2
  port (IN0, IN1 : in bit;
       OUT0 : out bit);
end component;

for all : and2 use entity GATE_LIB.AND2_NTY(AND2_A);
for all : and3 use entity GATE_LIB.AND3_NTY(AND3_A);
for all : xor2 use entity GATE_LIB.XOR2_NTY(XOR2_A);
-- continue on next page
-- continuing

    signal XOR_RES : bit; -- INTERNAL SIGNAL
    -- NOTE THAT OTHER SIGNALS ARE ALREADY DECLARED IN ENTITY

    begin

        A0 : AND2 port map (ENABLE, XOR_RES, RESULT);
        A1 : AND3 port map (X, Y, ENABLE, CARRY);
        X0 : XOR2 port map (X, Y, XOR_RES);

    end HALF_ADDER_C;
An entity may be implemented with different architectures.
The architectures have one thing in common, i.e., an identical interface.
Design Styles may vary: block function model, hardware model.
Design Speed, size may be different; confidential or disclosed.
The designer selects a particular architecture for a given design; for example: use entity Work.A(BEHAVIOR).
Architecture Example

**Dataflow**

```vhdl
architecture DATAFLOW of MUX is
begin
    Z <= A when SEL = '1' else B;
end DATAFLOW;
```

**Behavioral**

```vhdl
architecture BEHAVIOR of MUX is
begin
    process
    begin
        if (SEL = '1') then
            Z <= A;
        else
            Z <= B;
        end if;
        wait on A, B, SEL;
    end process;
end BEHAVIOR;
```

**Structural**

```vhdl
use Work.XYZ_GATES.all;
architecture STRUCTURAL of MUX is
signal SELBAR, ASEL, BSEL: Bit;
begin
    U0: INV port map (SEL, SELBAR);
    U1: NAND2 port map (A, SEL, ASEL);
    U2: NAND2 port map (SELBAR, B, BSEL);
    U3: NAND2 port map (ASEL, BSEL, Z);
end STRUCTURAL;
```
Architecture Summary

- Describes the functionality
- Structural architecture: similar to schematic
- Behavioral architecture: similar to C program
- Can have many architectures per entity
Complete VHDL Design Example, Comparator

Designing a comparator which accepts two 1-bit inputs, compares them, and produces a 1-bit result (either 1, indicating a match, or 0, indicating a difference between the two input values.)
Comparator
Entity Specification

entity COMPARATOR is
  port(
    A, B, SEL : in Bit;
    Z: out Bit
  );
end COMPARATOR;

Port Names
Entity Name
Directions
Data Types
Comparator

Behavioral Architecture Specification

```vhdl
architecture BEHAVIOR of COMPARATOR is
begin
    process (A, B)
    begin
        if (A = B) then
            Z <= '1';
        else
            Z <= '0';
        end if;
    end process;
end BEHAVIOR;
```
Comparator

Dataflow Architecture Specification

```
architecture DATAFLOW of COMPARATOR is
begin
    Z <= not (A xor B);
end DATAFLOW;
```
Comparator

Structural Architecture Specification

```vhvl
design
architecture STRUCTURAL of COMPARATOR is
  signal I: Bit
  component XR2 port (x,y: in Bit; Z: out Bit); end component;
  component INV port (x: in Bit; Z: out Bit); end component;
  begin
    U0: XR2 port map(A,B,I);
    U1: INV port map (I,Z);
  end STRUCTURAL;
```

EE 595 EDA / ASIC Design Lab
Packages and Libraries

- User defined constructs declared inside architectures and entities are not visible to other VHDL components
  - Scope of subprograms, user defined data types, constants, and signals is limited to the VHDL components in which they are declared
  - Packages and libraries provide the ability to reuse constructs in multiple entities and architectures
  - Items declared in packages can be *used* (i.e. included) in other VHDL components
Packages

- Packages consist of two parts
  - Package declaration -- contains declarations of objects defined in the package
  - Package body -- contains necessary definitions for objects in package declaration
    - e.g. subprogram descriptions

- Examples of VHDL items included in packages:
  - Basic declarations
    - Types, subtypes
    - Constant
    - Subprograms
    - Use clause
  - Signal declarations
  - Attribute declarations
  - Component declarations
Packages (cont’d)

**Package Header (Declaration)**

```vhdl
package PACK_NAME is
  function DEC(IO:Bit) return Bit;
end PACK_NAME;
```

**Package Body**

```vhdl
package body PACK_NAME is
  function DEC(IO:Bit) return Bit is
    begin
      return NOT IO;
    end DEC;
end PACK_NAME;
```
Packages (cont’d)

- Stores Declarations of common types, subprograms, constants, etc.
- Allows sharing between design units (without recompiling)
- Package Header (Declaration)
  - subprogram declaration
  - type declaration
  - component declarations
  - deferred constant declarations
- Package Body
  - subprogram body, deferred constant value
Package Declaration

An example of a package declaration:

```vhd
package MY_STUFF is
  type BINARY is (ON, OFF);
  constant PI : real := 3.14;
  constant MY_ID : integer;
  procedure ADD_BITS3 (signal A, B, EN : in Bit;
                        signal TEMP_RESULT, TEMP_CARRY : out Bit);
end MY_STUFF;
```

Note some items only require declaration while others need further detail provided in subsequent package body.

- for type and subtype definitions, declaration is sufficient
- subprograms require declarations and descriptions
The package body includes the necessary functional descriptions for objects declared in the package declaration

- e.g. subprogram descriptions, assignments to constants

```vhdl
package body MY_STUFF is
  constant MY_ID : integer := 2;

  procedure ADD_BITS3 (signal A, B, EN : in Bit;
                       signal TEMP_RESULT, TEMP_CARRY : out Bit) is
  begin  -- this function can return a carry
    TEMP_RESULT <= (A XOR B) AND EN;
    TEMP_CARRY <= A AND B AND EN;
  end ADD_BITS3;
end MY_STUFF;
```
Package Body

Constant value in header is visible

```vhdl
package MATH is
    constant MEAN_VALUE : real := 110.0;
end MATH;
```

No Package Body

Constant value in header is visible

```vhdl
package MATH is
    constant MEAN_VALUE : real := 110.0;
end MATH;
```

Package Body

```vhdl
package body MATH is
    constant MEAN_VALUE : real := 110.0;
end MATH;
```

Package Body
Packages must be made visible before their contents can be used.

- The USE clause makes packages visible to entities, architectures, and other packages.

```vhdl
-- use only the binary and add_bits3 declarations
use MY_STUFF.BINARY, MY_STUFF.ADD_BITS3;

... entity DECLARATION...
... architecture DECLARATION ...
```

```vhdl
-- use all of the declarations in package my_stuff
use MY_STUFF.all;

... entity DECLARATION...
... architecture DECLARATION ...
```
library MY_LIB;
use MY_LIB.COMPONENTS.all;
entity COMPARE is
  port (A,B: in Bit; C:out Bit);
end COMPARE;

architecture STRUCTURAL of COMPARE is
  signal I: Bit;
begin
  U0: XR2 port map(A,B,I);
  U1: INV port map(I,C);
end STRUCTURAL;
Package
Component Declared in Package

- Actual component entity and architectures are compiled outside of the package and in a library.

```vhdl
package COMPONENTS is

  component XR2  port(X,Y: in Bit; Z:out  Bit);
  end component;

  component INV  port(X: in Bit; Z:out  Bit);
  end component;

  ...

end COMPONENTS;
```
IEEE Packages

- **std_logic_1164**
  - Types std_ulogic, std_ulogic_vector, std_logic_vector and resolved subtype std_logic

- **std_logic_textio**
  - TextIO subprograms for use with 1164 types

- **std_logic_components**
  - Logic gates with std_logic_vector/std_logic ports

- **std_logic_signed**
  - 2’s-complement overloaded arithmetic and logical operators for std_logic/std_logic_vector

- **std_logic_unsigned**
  - Unsigned overload arithmetic and logical operators for std_logic/std_logic_vector

- **std_logic_arith**
  - Defines SIGNED and UNSIGNED types and overloaded arithmetic and logical operators
Libraries

- Design Units are stored in libraries
- Need to declare library before using it
- Every vendor will have a different library format
- One design can use many libraries
Libraries (cont’d)

- Library contains compiled entities, architectures, packages and configuration.
- They store VHDL packages, user designs, and ASIC vendor components.
- The results of a VHDL compilation are kept inside of a library for subsequent simulation, or for use in other designs.
Libraries (cont’d)

- A library can contain:
  - A package - shared declaration
  - An entity - shared designs
  - An architecture - shared design implementation
  - A configuration - shared design version.

- The benefits of using a library is to promote the sharing of previously compiled designs and the source design units need not be disclosed to all users.
Libraries (cont’d)

- Analogous to directories of files
  - VHDL libraries contain analyzed (i.e. compiled) VHDL entities, architectures, and packages
- Facilitate administration of configuration and revision control
  - E.g. libraries of previous designs
- Libraries accessed via an assigned logical name
  - Current design unit is compiled into the **Work** library
  - Both **Work** and **STD** libraries are always available
  - Many other libraries usually supplied by VHDL simulator vendor
    - E.g. proprietary libraries and IEEE standard libraries
Libraries (cont’d)

- Certain VHDL objects are kept by name in a library:
  - package -- shared declarations
  - architecture -- shared designs
  - entity -- shared design interfaces
  - configurations -- shared design versions

- VHDL expects: -- your working library to be called **Work**
  -- standard library called **Std**
  -- standard package **standard**
  -- text input/output **Textio**

- Users can have other resource libraries for:
  - synthesis -- project designs
  - ASIC vendor library -- simulation, components
Standard Libraries

- STD: This library is read-only
  - Stores standard package and textio package
- IEEE: This library stores std_logic_1164 package
- Some vendors put other packages into IEEE library

```cpp
library std; -- no need to use
library IEEE; -- needed
use std.standard.all -- no need to use
use std.textio.all -- needed
use IEEE.std_logic_1164.all -- needed
```
Personalized Libraries

- Libraries have a disk name and a logical name
  - project.lib -- disk name, file system name
  - project -- name used in VHDL

```vhdl
library PROJECT;
use PROJECT.TYPES.all;
```

- The working library has two logical names: the logical given by user and the logical name WORK
Vendor Libraries

```vhdl
library SYNOPSYS;
use SYNOPSYS.DISTRIBUTIONS.all;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
-- overloads <, > to do unsigned arithmetic.
```
library Example

package MY_DEFS is
constant UNIT_DELAY: time := 1 ns;
end MY_DEFS

entity COMPARE is
port (A, B: in bit;
     C: out bit);
end COMPARE;

library FREDS_LIBRARY;
use FREDS_LIBRARY.MY_DEFS.all;

architecture FLOW of COMPARE is
begin
C <= not (A xor B) after UNIT_DELAY;
end FLOW;
VHDL Design Organization

- Standard Libraries
- User Libraries
- Entity/Architecture
- Package
- Entity/Architecture
- Entity/Architecture
Putting It All Together

- Package
- Generics
- Entity
- Ports
- Architecture
  - Data Flow
    - Concurrent Statements
  - Behavioral
    - Concurrent Statements
  - Structural
    - Process
      - Sequential Statements
VHDL Language Hierarchy

- Libraries
- Design Units
- Statements
- Expression
- Objects
- Types