AN Introduction to VHDL
Overview

Dinesh Sharma

Microelectronics Group, EE Department
IIT Bombay, Mumbai

January, 2014
Part I

VHDL Design Units

1 Design Units in VHDL
   - entity
   - Architecture
   - Component
   - Configuration
   - Packages and Libraries

2 Object and Data Types
   - Scalar data types
   - Composite Data Types
An Introduction to VHDL

VHDL is a hardware description language which uses the syntax of ADA. Like any hardware description language, it is used for many purposes.

- For describing hardware.
- As a modeling language.
- For simulation of hardware.
- For early performance estimation of system architecture.
- For synthesis of hardware.
- For fault simulation, test and verification of designs.

etc.
VHDL versions

1987  First revised “Public” version.
1993  Probably the most widely used version. Made syntax consistent, added xnor function etc.
2000  Minor revision: Introduced protected types.
2002  Minor revision: Rules with regard to buffer ports relaxed.
2008  Major revision: General clean up, introduced the use of external signals.

In this course, version 93 should be taken to mean 93 and later.
Design Elements in VHDL: ENTITY

The basic design element in VHDL is called an ‘ENTITY’.

- An ENTITY represents a template for a hardware block.
- It describes just the outside view of a hardware module – namely its interface with other modules in terms of input and output signals.
- The hardware block can be the entire design, a part of it or indeed an entire “test bench”.
- A test bench includes the circuit being designed, blocks which apply test signals to it and those which monitor its output.
- The inner operation of the entity is described by an ARCHITECTURE associated with it.
ENTITY DECLARATION

The declaration of an ENTITY describes the signals which connect this hardware to the outside. These are called \textit{port} signals. It also provides optional values of manifest constants. These are called generics.

**VHDL 93 (and later)**

\begin{verbatim}
entity name is
   generic(list);
   port(list);
end entity name;
\end{verbatim}

**VHDL 87**

\begin{verbatim}
entity name is
   generic(list);
   port(list);
end name;
\end{verbatim}
ENTITY EXAMPLE

VHDL 93 (and later)

entity flipflop is
  generic (Tprop: delay_length);
  port (clk, d: in bit; q: out bit);
end entity flipflop;

VHDL 87

entity flipflop is
  generic (Tprop: delay_length);
  port (clk, d: in bit; q: out bit);
end flipflop;

The entity declares port signals, their directions and data types.

These signals are used by an architecture associated with this entity.
Design Elements in VHDL: ARCHITECTURE

An ARCHITECTURE describes how an ENTITY operates. An ARCHITECTURE is always associated with an ENTITY.

There can be multiple ARCHITECTURES associated with an ENTITY.

An ARCHITECTURE can describe an entity in a structural style, behavioural style or mixed style.

The language provides constructs for describing components, their interconnects and composition (structural descriptions).

The language also includes signal assignments, sequential and concurrent statements for describing data and control flow, and for behavioural descriptions.
ARCHITECTURE Syntax

VHDL 93 (and later)

```
architecture name of entity-name is
  (declarations)
begin  (concurrent statements)
end architecture name;
```

VHDL 87

```
architecture name of entity-name is
  (declarations)
begin  (concurrent statements)
end name;
```

The architecture inherits the port signals from its entity. It must declare its internal signals. Concurrent statements constituting the architecture can be placed in any order.
ARCHITECTURE Example

VHDL 93 (and later)

architecture simple of dff is
signal ...;
begin
...
end architecture simple;

VHDL 87

architecture simple of dff is
signal ...;
begin
...
end simple;
Signals in an Architecture

- The architecture inherits the port signals from its entity. These signals have global scope inside every architecture associated with this entity.

- In addition to the port signals, an architecture may have internal signals. These must be declared in the declaration section.

- Internal signals declared here will have global scope inside this particular architecture.

- Local signals and variables may be declared inside internal blocks of the architecture.

- Signals are carried by wires, variables are used for array indices, loop counters, etc.
Design Elements in VHDL: COMPONENTS

- An ENTITY ↔ ARCHITECTURE pair actually describes a component type.
- In a design, we might use several instances of the same component type.
- Each instance of a component type may be distinguished by using a unique name.
- Thus, a component instance with a unique instance name is associated with a component type, which in turn is associated with an ENTITY ↔ ARCHITECTURE pair.
- This is like saying U1 (component instance) is a D Flip Flop (component type) which is associated with an entity DFF (which describes its pin diagram) using architecture LS7474 (which describes its inner operation).
Component Example

VHDL 93 (and later)

```vhdl
component name is
  generic(list);
  port(list);
end component name;
```

EXAMPLE:

```vhdl
component flipflop is
  generic (Tprop: delay_length);
  port (clk, d: in bit; q: out bit);
end component flipflop;
```

VHDL 87

```vhdl
component name
  generic(list);
  port(list);
end component;
```

EXAMPLE:

```vhdl
component flipflop
  generic (Tprop: delay_length);
  port (clk, d: in bit; q: out bit);
end component;
```
Design Elements in VHDL: Configuration

Structural Descriptions describe components and their interconnections.

A component is an instance of a component type. Each component type is associated with an ENTITY $\leftrightarrow$ ARCHITECTURE pair.

The architecture used can itself contain other components - whose type will then be associated with other ENTITY $\leftrightarrow$ ARCHITECTURE pairs.

A “configuration” describes linkages between component types and ENTITY $\leftrightarrow$ ARCHITECTURE pairs. It specifies bindings for all components used in an architecture associated with an entity.
Design Elements in VHDL: Packages

Related declarations and design elements like subprograms and procedures can be placed in a "package" for re-use.

A package has a declarative part and an implementation part.

This is somewhat like entity and architecture for designs.

Objects in a package can be referred to by a packagename.objectname syntax.

A description can include a ‘use’ clause to incorporate the package in the design. Objects in the package then become visible to the description without having to use the dot reference as above.
Design Elements in VHDL: Libraries

Many design elements such as packages, definitions and entire entity architecture pairs can be placed in a library.

The description invokes the library by first declaring it:
For example, Library IEEE;

Objects in the Library can then be incorporated in the design by a ‘use’ clause.
For example, Use IEEE.std_logic_1164.all

In this example, IEEE is a library and std_logic_1164 is a package in the library.
VHDL defines several types of objects. These include constants, variables, signals and files.

The types of values which can be assigned to these objects are called data types.

Same data types may be assigned to different object types. For example, a constant, a variable and a signal can all have values which are of data type BIT.

Declarations of objects include their object type as well as the data type of values that they can acquire. For example signal Enable: BIT;
Data Types

Scalar
- Discrete
  - Integer
  - enumeration
- Floating Pt.
  - real
  - time
- Physical

Access
- boolean
- PhysicalFloatingPt.Discrete

File
- unconstrained array
- string
  - file_open_kind
- bit
  - bit_vector
- character
  - Severity Level
  - file_open_status

Composite
- constrained array
Enumerated Types

VHDL enumeration types allow us to define a set of values that an object of this type can acquire. For example, we can define a data type by the following declaration:

\[ \text{type instr is (add, sub, adc, sbb, rotl, rotr);} \]

Now a variable or a signal defined to be of type instr can only be assigned values enumerated above – that is: add, sub, adc, sbb, rotl and rotr.

In actual implementation, these values may be mapped to a 3 bit value. However, an attempt to assign, say, “010” to a variable of type instr will result in an error. Only the enumerated values can be assigned to a variable of this type.
A few enumeration types are pre-defined in the language. These are:

```vhdl
type bit is ('0', '1');
type Boolean is (false, true);
type severity_level is (note, warning, error, failure);
type file_open_kind is (read_mode, write_mode, append_mode);
type file_open_status is
  (open_ok, status_error, name_error, mode_error);
```

In addition to these, the character type enumerates all the ASCII characters.
A signal type defined in the IEEE Library is std_logic. This is a signal which can take one of 9 possible values. It is defined by:

type std_logic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

A subtype of this kind of signal can be defined, which can take the four values ‘X’, ‘0’, ‘1’, and ‘Z’ only. This can be defined to be a subtype of std_logic

subtype fourval_logic is std_logic range 'X' to 'Z';

Similarly, we may want to constrain some integers to a limited range of values. This can be done by defining a new type:

subtype bitnum is integer range 31 downto 0;
Physical Types

Objects which are declared to be of Physical type, carry a value as well as a unit. These are used to represent physical quantities such as time, resistance and capacitance.

The Physical type defines a basic unit for the quantity and may define other units which are multiples of this unit.

Time is the only Physical type, which is pre-defined in the language. The user may define other Physical types.
Pre-defined Physical Type: Time

```vhdl
type time is range 0 to ...

units
  fs;
  ps = 1000 fs;
  ns = 1000 ps;
  us = 1000 ns;
  ms = 1000 us;
  sec = 1000 ms;
  min = 60 sec;
  hr = 60 min;
end units time;
```

The user may define other physical types as required.
As an example of user defined Physical types, we can define the resistance type.

```vhdl
type resistance is range 0 to 1E9
    units
    ohm;
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
end units resistance;
```
Composite data types are collections of scalar types. VHDL recognizes records and arrays as composite data types. Records are like structures in C. Arrays are indexed collections of scalar types. The index must be a discrete scalar type. Arrays may be one-dimensional or multi-dimensional.
Arrays

Arrays can be constrained or unconstrained.

- In constrained arrays, the type definition itself places bounds on index values. For example:

  ```vhdl
  type byte is array (7 downto 0) of bit;
  type rotmatrix is array (1 to 3, 1 to 3) of real;
  ```

- In unconstrained arrays, no bounds are placed on index values. Bounds are established at the time of declaration.

  ```vhdl
  type bus is array (natural range <> ) of bit;
  ```

The declaration could be:

```vhdl
signal addr_bus: bus(15 downto 0);
signal data_bus: bus(7 downto 0);
```
Built in Array types

VHDL defines two built in types of arrays. These are: bit_vectors and strings. Both are unconstrained.

```vhdl
type bit_vector is array (natural range <> ) of bit;
type string is array (positive range <> ) of character;
```

As a result we can directly declare:

```vhdl
variable message: string(1 to 20);
signal Areg: bit_vector(7 downto 0);
```
Records

While an array is a collection of the same type of objects, a record can hold components of different types and sizes. This is like a struct in C.

The syntax of a record declaration contains a semicolon separated list of fields, each field having the format name, . . . , name : type or name, . . . , name : subtype
For example:

```vhdl
type resource is record
(P_reg, Q_reg : bit_vector(7 downto 0); Enable: bit)
end record resource;
```