

Introduction to VHDL

Programming Language

- Can we use C or Java as HDL?
- A computer programming language
- Develop of language
 - Semantics (“meaning”)
 - Syntax (“grammar”)
- Study the characteristics of the underlying processes
 - Develop syntactic constructs and their associated semantics to model and express these characteristics

Traditional PL

- Modeled after a sequential process
 - Operations performed in a sequential order
 - Help human's thinking process to develop an algorithm step by step
 - Resemble the operation of a basic computer model

- Characteristics of digital hardware
 - Connection of parts
 - Concurrent operations
 - Concept of propagations delay and timing
- Characteristics cannot be captured by traditional PLs
 - Require new language : **HDL**

Use of an HDL program

- Formal documentation
- Input to simulator
- Input to synthesizer

Modern HDL

- Capture characteristics of a digital circuit:
 - Entity
 - Connectivity
 - Concurrency
 - Timing
- Cover description
 - In Gate level and RT (Register Transfer) level
 - In structural view and behavior view

Highlights of Modern HDL:

- Encapsulates the concepts of entity, connectivity, concurrency, and timing
- Incorporate propagation delay and timing information
- Consist of constructs for structural implementation
- Incorporate constructs for behavioral description (sequential execution of traditional PL)
- Describe the operations and structures in gate level and RT level
- Consists of constructs to support hierarchical design process

Advantages of HDL

- We can verify design functionality early in the design written an HDL description.
- Design simulation at this higher level before implementation at gate level, allow you to test architecture and design decision.
- Reduced non-recurring engineering costs.
- Design reused in enabled.
- Increase flexibility to design changes.
- Better and easier design auditing and verification.

Two HDLs used today

- i) **VHDL**, and ii) **Verilog**
- Syntax and “appearance” of the two language are very different
- Capabilities and scopes are quite similar
- Both are industrial standards and are supported by most software tools

Hardware Description Language

- In computer engineering, a **hardware description language (HDL)** is a specialized computer language used to describe the structure and behavior of electronic circuits, and most commonly, digital logic circuits.
- A hardware description language enables a precise, **formal** description of an electronic circuit that allows for the **automated analysis** and **simulation** of an electronic circuit.
- It also allows for the **synthesis** of a HDL description into a **netlist** (a specification of physical electronic components and how they are connected together), which can then be placed and routed to produce the set of masks used to create an integrated circuit.

Hardware Description Language

- A hardware description language looks much like a **programming language** such as C; it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs **explicitly include the notion of time**.
- HDLs form an integral part of **electronic design automation (EDA)** systems, especially for complex circuits, such as **application-specific integrated circuits, microprocessors, and programmable logic devices**.

Introduction to VHDL

- VHDL stands for **VHSIC** Hardware **D**escription **L**anguage. VHSIC stands for **V**ery **H**igh **S**peed **I**ntegrated **D**esign. Therefore, VHDL expanded is Very High Speed Integrated Circuit Hardware Description Language.
- VHDL is a language that is used to describe the behavior of digital circuit designs.
- Hierarchical use of VHDL designs permits the rapid creation of complex digital circuits designs.
- Initially sponsored by USA DoD (Department of Defense) as a hardware documentation standard in early 80s.
- Transferred to IEEE and ratified it as IEEE (Institute of Electrical and Electronics Standard) standard 1176 in 1987 (known as VHDL-87)
- Major modification in '93 (known as VHDL- 93)
- Revised continuously.

Extensions to VHDL

- **IEEE 1076.1**
 - IEEE Standard VHDL Analog and Mixed-Signal Extensions
- **IEEE 1076.2**
 - IEEE Standard VHDL Mathematical Packages
- **IEEE 1076.3**
 - IEEE Standard VHDL Synthesis Package
- **IEEE 1076.4**
 - IEEE Standard VITAL ASIC Modeling Specification
- **IEEE 1076.5**
 - IEEE Standard VHDL Utilities Packages [Not Standard]
- **IEEE 1076.6**
 - IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

Related IEEE Standard

- IEEE 1164
 - Standard Multivalued Logic System for VHDL Model Interoperability
- IEEE 164
 - IEEE Standard Description Language Based on the Verilog Hardware Description Language

Features of VHDL

- VHDL has powerful constructs.
- In VHDL, design may be decomposed hierarchically.
- Each design element has a well defined interface useful for connecting it to other elements.
- Each design elements has a precise behavioral specification useful for simulating it.
- VHDL handles asynchronous as well as synchronous sequential circuits.
- In VHDL, timing and clocking can be modeled.
- In VHDL, design is target independent.
- VHDL, supports design library.
- The language is not case sensitive.

VHDL References

- Douglas Perry, VHDL, 3rd Edition, McGraw Hill, New York, NY, 1998
- Peter J. Ashenden, The Designer's Guide to VHDL, 2nd Edition, Morgan Kaufmann Publishers, Inc, San Francisco, CA, 2002
- Stephen Brown and Zvonko Vranasic, Fundamentals of Digital Logic with VHDL Design, 2nd Edition, McGraw-Hill, New York, NY, 2004

VHDL Fundamentals

Naming Conventions

- For the purpose of this tutorial, the following naming conventions will be used:
 - All the VHDL keywords are shown in uppercase.
 - All identifiers are shown in lowercase.
 - The color highlighting has been used to enhance the readability of the VHDL code fragments.

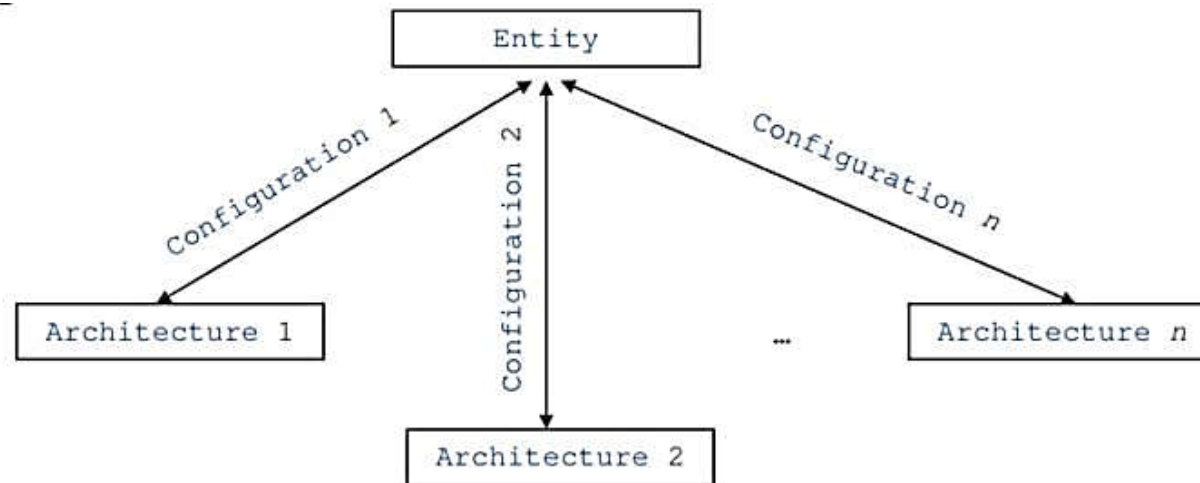
Libraries and Packages

- Libraries provide a set of packages, components, and functions that simplify the task of designing hardware.
- Packages provide a collection of related data types and subprograms in a design.
- Think of a package as a tool-box that contains tools used to build designs.
- The following is an example of the use of the ieee library and its std_logic_1164 package:

```
LIBRARY ieee;  
  
USE ieee.std_logic_1164.ALL;
```

Entities, Architectures, and Configurations

- A VHDL design consists of entities, architectures, and configurations.



Entities

- An entity is a specification of the design's external interface.
- It defines the names, input output signals and modes of a hardware module.
- Entity declarations specify the following:
 1. The name of the entity
 2. A set of generic declarations specifying instance-specific parameters.
 3. A set of port declarations defining the inputs and output of the hardware design.
- Generic declarations and port declarations are optional

Entities

The following is an example of an entity declaration for an AND gate:

```
ENTITY andgate IS
PORT (  a : IN std_logic;
        b : IN std_logic;
        c : OUT std_logic );
END andgate;
```

```
ENTITY entity_name IS
  GENERIC (
    generic_1_name      : generic_1_type;
    generic_2_name      : generic_2_type;
    generic_n_name      : generic_n_type
  );
  PORT (
    port_1_name      : port_1_dir port_1_type;
    port_2_name      : port_2_dir port_2_type;
    port_n_name      : port_n_dir port_n_type
  );
END entity_name;
```

NOTE:

In the PORT declaration, the semi-colon is used as a separator.

- Port name choices:
 - Consists of letters, digits and/or underscores
 - Always begin with a letter
 - Case insensitive

- Port direction choices:

IN	Input port
OUT	Output port
INOUT	Bidirectional port
BUFFER	Buffered output port

- IEEE recommends the use of the following data types to represent signals in a synthesizable system:

```
std_logic
std_logic_vector (<max> DOWNT0
<min>)
```

Architectures

- An architecture is a specification of the design's internal implementation.
- Multiple architectures can be created for a particular entity.
- For example, you might wish to create several architectures for a particular entity with each architecture optimized with respect to a design goal:
 - Performance
 - Area
 - Power consumption
 - Ease of simulation

Architecture Declarations

- Architecture declarations are specified as follows:

```
ARCHITECTURE architecture_name OF entity_name IS  
Architecture_declarative part;  
  
BEGIN  
    Statements;  
END architecture_name;
```

The following is an example of an architecture declaration for an AND gate:

```
ARCHITECTURE synthesis OF andgate IS  
  
BEGIN  
    c <= a AND b;  
END synthesis
```

NOTE:

The keyword **AND** denotes the use of AND gate.

Configurations

- A configurations is a specification of the mapping between an architecture and a particular instance of an entity.
- By default, a configuration exists for each entity.
- The default configuration maps the most recently compiled architecture to the entity.

Signals

- Signals represent wires and storage elements.
- Signals may only be defined inside architectures.
- Signals are associated with a data types.
- Signals have attributes.
- VHDL is a strongly-typed language:
 - Explicit type conversion is supported.
 - Implicit type conversion is not supported

Built-In Data Types

- VHDL supports a rich set of built-in data types as well as user-defined data types; BIT, BIT_VECTOR, INTEGER, REAL
- Built in data types work well for simulations but not so well for synthesis.

Logical Operators

Logical operators:

- AND
- OR
- NOT
- NAND
- NOR
- XOR

Relations Operators:

=	(Equal)
/=	(Not Equal)
<	(Less Than)
>	(Greater Than)

Mathematical Operators:

+	(Addition)
-	(Subtraction)
*	(Multiplication)
/	(Division)

Assignment Operator:

< =

Modeling

- Data Flow Modeling
- Behavioral Modeling
- Structural Modeling

VHDL code for AND Gate:

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY and_gate IS  
    PORT (x, y : IN std_logic;  
          z : OUT std_logic);  
END and_gate;  
  
ARCHITECTURE and_behav OF and_gate IS  
BEGIN  
    z <= x AND y after 5 ns;  
END and_behav;
```

VHDL code for OR Gate:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity or_gate is  
    port (x, y : in std_logic ;  
          z : out std_logic);  
end or_gate;  
  
architecture or_behav of or_gate is  
begin  
    z <= x or y after 5 ns;  
end or_behav;
```

VHDL code for NOT Gate:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity not_gate is  
    port (x : in std_logic;  
          z : out std_logic);  
end not_gate;  
  
architecture not_behav of not_gate is  
begin  
    z <= not x after 5 ns;  
end not_behav;
```


VHDL code for NAND Gate:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity nand_gate is  
    port (x, y : in std_logic ;  
          z : out std_logic);  
end nand_gate;  
  
architecture nand_behav of nand_gate is  
begin  
    z <= x nand y after 5 ns;  
end nand_behav;
```

VHDL code for NOR Gate:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity nor_gate is  
    port (x, y : in std_logic ;  
          z : out std_logic);  
end nor_gate;  
  
architecture nor_behav of nor_gate is  
begin  
    z <= x nor y after 5 ns;  
end nor_behav;
```

VHDL code for XOR Gate:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity xor_gate is  
    port (x, y : in std_logic;  
          z : out std_logic);  
end xor_gate;  
  
architecture xor_behav of xor_gate is  
begin  
    z <= x xor y after 5 ns;  
end xor_behav;
```

VHDL Code for XNOR Gate

```
library ieee;
use ieee.std_logic_1164.all;

entity xor_gate is
    port (x, y : in std_logic;
          z : out std_logic;
end xor_gate;

architecture xor_behav of xor_gate is
begin
    z <= not (x xor y) after 5 ns;
end xor_behav;
```

VHDL Code for Half-Adder

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
ENTITY half_adder is  
    port (x, y : IN std_logic;  
          c, s : OUT std_logic);  
END half_adder;
```

```
ARCHITECTURE half_adder_behav OF half_adder IS  
BEGIN  
    s <= x XOR y ;  
    c <= x AND y;  
END half_adder_behav;
```

VHDL Code for Half-Subtractor

```
library ieee;
use ieee.std_logic_1164.all;

entity half_sub is
    port (x, y : in std_logic ;
          b, d : out std_logic);
end half_sub;

architecture half_sub_behav of half_sub is
begin
    d <= x xor y ;
    b <= (x and (not y));
end half_sub__behav;
```

VHDL Code for Full-Subtractor

```
library ieee;
use ieee.std_logic_1164.all;

entity full_sub is
    port (x, y, z : in std_logic;
          b, d : out std_logic);
end full_sub;

architecture data of full_sub is
begin
    d <= x xor y xor z ;
    b <= ((y xor z) and (not x)) or (y and z);
end data;
```


VHDL Code for Multiplexer

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX4 is
    port (S0, S1, D0, D1, D2, D3: in std_logic ;
          y : out std_logic);
end MUX4;

architecture dataflow of MUX4 is
    signal select : integer

Begin
    select <= 0 when S0 = '0' and S1 = '0' else
        1 when S0 = '0' and S1 = '1' else
        2 when S0 = '1' and S1 = '0' else
        3;
    Y <= D0 after 10 ns when select = 0 else
        D1 after 10 ns when select = 1 else
        D2 after 10 ns when select = 2 else
        D3 after 10 ns when select = 3 else
end dataflow;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux_4to1 is
  port(

    A,B,C,D : in STD_LOGIC;
    S0,S1: in STD_LOGIC;
    Z: out STD_LOGIC
  );
end mux_4to1;

architecture bhv of mux_4to1 is
begin
  process (A,B,C,D,S0,S1) is
```

```
begin
  if (S0 ='0' and S1 = '0') then
    Z <= A;
  elsif (S0 ='1' and S1 = '0') then
    Z <= B;
  elsif (S0 ='0' and S1 = '1') then
    Z <= C;
  else
    Z <= D;
  end if;

end process;
end bhv;
```

VHDL Code for Demultiplexer

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity DEMUX4 is
```

```
    port ( S0, S1, D      : in std_logic ;  
          Y0, Y1, Y2, Y3 : out std_logic);
```

```
end DEMUX4;
```

```
architecture dataflow of DEMUX4 is
```

```
begin
```

```
    Y0 <= ((not S0) and (not S1) and D);
```

```
    Y1 <= ((not S0) and (S1) and D);
```

```
    Y2 <= ( S0 and (not S1) and D);
```

```
    Y3 <= (S0 and S1 and D);
```

```
End dataflow;
```