Introduction to VHDL

- In the 1980s rapid advances in integrated circuit technology lead to efforts to develop standard design practices for digital circuits.
- VHDL was initiated in 1981 by the Department of Defense in an effort to create a hardware development language that was independent of any specific technology or design method.
- VHDL is a type of hardware description language (HDL) used to program PLDs (Programmable Logic Devices).
- The V in VHDL stands for VHSIC (Very High Speed Integrated Circuit).
- The original IEEE standard for VHDL was adopted in 1987 and called IEEE 1076. A revised standard was adopted in 1993 and called IEEE 1164.
- VHDL is an extremely complex, sophisticated language. Learning all of its feature is a difficult task. However, for use in synthesis only a subset of these features is important.
- Other examples of HDLs are Verilog, AHDL, ABEL, and CUPL. Verilog is also an IEEE standard language and is similar to VHDL. AHDL, ABEL, and CUPL are proprietary languages owned by PLD manufacturers.

Using VHDL

- Using VHDL to program a digital logic design
  - Define What the program is going to do.
  - Determine How the task is to be accomplished.
  - Create the VHDL program code.
  - Test and debug the program code.
  - Implement the design on the target device.

Create VHDL code

- Define program requirements.
- Determine a solution.
- Write VHDL code using development software, e.g., MAX+Plus II (a product from Altera) and WebPACK (a product from Xilinx).
  - Select target device
  - Assign pin numbers for inputs and outputs
  - Compile the program
  - Test the program
  - Download to the target device
Programming a CPLD/FPGA

Representation of Digital Signals in VHDL

- Each logic signal in the circuit is represented in VHDL code as a data object.
- Just as the variables declared in any high-level programming language have associated types, such as integers or characters, data objects in VHDL can be of various types.
- The original VHDL standard, IEEE 1076, includes a data type called BIT.
- An object of this type is well suited for representing digital signals because BIT objects can have only two values, 0 and 1.

The Entity (I)

- The Entity describes the external aspects of the logic function.
  - The entity describes a given logic function by naming the function and defining the inputs and outputs and the type of data set.
  - Each input and output of a logic function is called a port.

The Entity (II)

- Consider the logic circuit in Figure 2.30.
- If we wish to write VHDL code to represent this circuit, the first step is to declare the input and output signals. This is done using a construct called an entity. An entity must be assigned a name, e.g., example1.
- The input and output signals for the entity are called its ports, and they are identified by the keyword PORT. Each port has an associated mode that specifies whether it is an input (IN) to the entity or an output (OUT) from the entity. Each port represents a signal, hence it has an associated type.
- An entity specifies the input and output signals for a circuit, but it does not give any details as to what the circuit represents.
- An appropriate entity for this example appears in Figure 2.31.
**Signal Names**

- In Figure 2.31 we have used simple signal names $x_1$, $x_2$, $x_3$, and $f$ for the entity’s ports.
- Signal names in VHDL can include any letter or number, as well as the underscore character ‘_’.
- A signal name must begin with a letter, and a signal name cannot be a VHDL keyword.

**The Architecture**

- The Architecture describes the internal aspects of the logic function.
  - For every entity there must be at least one architecture.
  - The circuit’s functionality must be specified with a VHDL construct called an architecture. An architecture for the logic circuit in Figure 2.30 appears in Figure 2.32.
  - VHDL has built-in support for the following Boolean operators: AND, OR, NOT, NAND, NOR, XOR, and XNOR.
  - Following the BEGIN keyword, the architecture specifies, using the VHDL signal assignment operator $<=$, that output $f$ should be assigned the result of the logic expression on the right-hand side of the operator.
Figure 2.30. A simple logic function.

ENTITY example1 IS
  PORT ( x1, x2, x3 : IN   BIT ;
         f : OUT   BIT ) ;
END example1 ;
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
  f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;

Figure 2.33. Complete VHDL code for the circuit in Figure 2.30.

Figure 2.34. VHDL code for a four-input function.

ENTITY example2 IS
  PORT ( x1, x2, x3, x4 : IN   BIT ;
         f, g : OUT   BIT ) ;
END example2 ;
ARCHITECTURE LogicFunc OF example2 IS
BEGIN
  f <= (x1 AND x3) OR (NOT x3 AND x2) ;
  g <= (NOT x3 OR x1) AND (NOT x3 OR x4) ;
END LogicFunc ;
Libraries

- A library is used to hold packages which are reusable code such as components, functions, and procedures and make them available to a VHDL program.
- There are two types of libraries: standard library and user-defined library.
- The IEEE standard library:
  - The std_logic_1164 package in the IEEE library provides common procedures, functions, and data types.

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

The keyword `use` identifies what is to be used from the IEEE library.

The keyword `library` makes packages in IEEE visible.

Packages

- A VHDL program may contain several entity/architecture pairs.
- The `package` is a VHDL unit used to store various items that is commonly used in the program.
- The `library` is a VHDL feature that stores items that can be accessed by several programs.

VHDL language elements (I)

- **Keywords** are reserved words in VHDL and are a type of identifier.
- **Basic identifiers** can be used to name an entity, architecture, and other items. Keywords cannot be used as basic identifiers.
- **Data objects** are used to hold a value or set of values. A data object can be a constant, variable, signal, or file.
- **Data types** are used to specify the type of data. The data type can be **bit** (1 and 0), **boolean** (true and false), and **integer** (in a certain range.)

VHDL language elements (II)

- **Literals** are numeric values that can be characters, strings of characters, strings of bits, and decimal numbers.
- **Operators** are elements that perform certain operations or functions. The logical operators include and, or, and not, etc. Relational operators include equality (=), inequality (/=), etc. The arithmetic operators are used to add, subtract, multiply, and divide, etc.
- **Comments**
  - Lines with comments start with two adjacent hyphens (--) and will be ignored by the compiler.
Identifiers

- Identifiers are user-defined names. They are used to identify VHDL elements and units. We will focus on basic identifier rules:
  - It consists of lowercase or uppercase letters, numeric digits, and single underscores.
  - The first symbol must be a letter.
  - Basic identifiers cannot contain spaces.
  - VHDL keywords or reserved identifiers may not be used as identifiers.
  - Examples of identifier are A, B1, X, myAnd, OR1, Encoder_2.
  - VHDL ignores spaces and is also case insensitive.

Table A.1. The VHDL operators.

<table>
<thead>
<tr>
<th>Operator Class</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest precedence</td>
<td>Miscellaneous **, ABS, NOT</td>
</tr>
<tr>
<td></td>
<td>Multiplying * , /, MOD, REM</td>
</tr>
<tr>
<td></td>
<td>Sign +, -</td>
</tr>
<tr>
<td></td>
<td>Adding +, -, &amp;</td>
</tr>
<tr>
<td></td>
<td>Shift SLL, SRL, SLA, SRA, ROL, ROR</td>
</tr>
<tr>
<td>Lowest precedence</td>
<td>Logical AND, OR, NAND, NOR, XOR, XNOR</td>
</tr>
</tbody>
</table>

Write a VHDL entity (I)

- The entity describes all of the inputs and outputs for a given logic function.
- The entity is where the interface of a logic function to the “outside world” is described.
- The Entity declaration
  
  ```vhdl
  Entity <entity identifier> is [ generic (generic_declarations); ]
  port (signal descriptions);
  End entity <entity identifier>;
  ```

Write a VHDL entity (II)

- The entity declaration begins with keyword entity, followed by an identifier assigned as the name of the entity and finally keyword is.
- generic: generic declarations are optional and determine the local constants used for timing and sizing (e.g. bus widths) the entity.
  
  ```vhdl
  generic (constant_name: type [=value] ;
            constant_name: type [=value] ;
  ```
- The port statement: the VHDL keyword port is used to declare the input and output signal assignments.
- The port direction: in, out, inout, buffer
- The port data type
Data Types (I)

- The data type bit has two possible values: 1 or 0.
- The data type bit_vector identifies a set(array) of indexed items with the same name.
  - Each of the individual elements in an array has a common identifier name with a numerical index, e.g., identifier “A” has A(0), A(1), … as its element.

```
entity decoder is
port (A: in bit_vector(0 to 3); X: out bit_vector(0 to 9));
end entity decoder;
architecture BCD_to_Decimal of decoder is
begin
X(0) <= not A(3) and not A(2) and not A(1) and not A(0);
X(1) <= not A(3) and not A(2) and not A(1) and A(0);
.....
end architecture BCD_to_Decimal;
```

Data Types (II)

- The data type integer can contain positive and negative whole numbers.
  - The data subtype natural can contain whole numbers starting at 0 to a specified limit.
  - The data subtype positive can contain whole numbers starting at 1 to a specified limit.
  
```
Example:
entity DataType is
port (A, B: in bit; Z: out integer range 0 to 7);
end entity DataType
```

- The data type Boolean has two possible values, true or false.

```
variable V1:boolean := false;
```

Data Types (III)

- Real
  - Values: 1.0, -1.0E5
  - Example: V1 = V2 / 5.3
- Time
  - Values: 1 us, 7 ns, 100 ps
  - Example: s1 <= (a xor b) after 3 ns;
- Character
  - Example: CharData <= ‘X’;
- String
  - Values: (Array of characters)
  - Example: Msg <= "MEM: "]

Table A.2. The possible modes for signals that are entity ports.
IEEE Standard 1164

- std_ulogic allows signals to have one source (driver)
  - unresolved logic from multiple drivers causes a simulation error message.
- std_logic allows signals to have multiple drivers
  - resolved logic is implemented with the simulation software to get a resolution between two sources, e.g., drivers conflict between '0' and '1' result in 'X'.
- std_logic, std_ulogic, std_logic_vector, std_ulogic_vector: can have 9 values to indicate the value and strength of a signal. Std_ulogic and std_logic are preferred over the bit or bit_vector types.

IEEE 1164 Value System

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'U'</td>
<td>Uninitialized</td>
</tr>
<tr>
<td>'X'</td>
<td>Forcing unknown</td>
</tr>
<tr>
<td>'0'</td>
<td>Forcing 0</td>
</tr>
<tr>
<td>'1'</td>
<td>Forcing 1</td>
</tr>
<tr>
<td>'Z'</td>
<td>High impedance</td>
</tr>
<tr>
<td>'W'</td>
<td>Weak unknown</td>
</tr>
<tr>
<td>'L'</td>
<td>Weak 0</td>
</tr>
<tr>
<td>'H'</td>
<td>Weak 1</td>
</tr>
<tr>
<td>'-'</td>
<td>Don't care</td>
</tr>
</tbody>
</table>

Entity declaration example

- Write an entity declaration for a 2-input AND gate.
  entity AND_gate is
  port (A,B: in bit; X: out bit);
  end entity AND_gate;
- Write an entity declaration for a 3-input AND gate.

Architecture Declaration

- Each architecture declaration must be associated by name with an entity.
  architecture <architecture name> of <entity name> is begin
    The description of the logic function goes here.
  end architecture;<architecture name>;
- Operators: Many types of operators in VHDL can be used in describing a logic function, e.g., the logic operator and, the signal assignment operator <=.
Architecture Declaration Example

- Write an architecture declaration for the 2-input AND gate associated with the previous entity example.
  architecture ANDfunction of AND_gate is begin
  X <= A and B;
  end architecture ANDfunction;

VHDL Description (I)

- The **Data flow** description:
  - Logic functions are described by how the data, represented by signals, flow from inputs to output.
  - Example of data flow describing an inverter with input A and output X is
    \[ X = \text{not} \ A; \]
- The **behavioral** description:
  - It describes what the logic does, based on the state of the function and the inputs at a particular time.
  - Example of behavioral describing the inverter:
    \[ X = '1' \text{ when } (A = '0') \text{ else } '1' \]

VHDL Description (II)

- The **structural** description:
  - Describe logic functions and specify how they are connected together.
  - The VHDL **component** is a way to predefine a logic function for repeated use.
  - The VHDL **signal** is a way to specify a “wire” connection between components.

Figure 2.d VHDL structural implementation

VHDL Components

- A VHDL component describes predefined logic that can be stored as a package declaration in a library.
- Use component to avoid repeating the same code over and over within a program.
  component name_of_component is
    port (port definitions);
  end component name_of_component;
Predefined programs

entity AND_gate is
  port(A,B: in bit; X: out bit);
end entity AND_gate;

architecture ANDfunction of AND_gate is
begin
  X<= A and B;
end architecture ANDfunction;

entity OR_gate is
  port(A,B: in bit; X: out bit);
end entity OR_gate;

architecture ORfunction of OR_gate is
begin
  X<= A or B;
end architecture ORfunction;

VHDL program

entity AND_OR_Logic is
  port(IN1,IN2,IN3,IN4: in bit; OUT3: out bit);
end entity AND_OR_Logic;

architecture LogicOperation of AND_OR_Logic is
begin
  component AND_gate is
    port(A,B: in bit; X: out bit);
  end component AND_gate;
  component OR_gate is
    port(A,B: in bit; X: out bit);
  end component OR_gate;
signal OUT1, OUT2: bit;
  G1:AND_gate port map (A =>IN1, B=>IN2,X=>OUT1);
  G2:AND_gate port map (A =>IN3, B=>IN4,X=>OUT2);
  G3:OR_gate port map (A =>OUT1, B=>OUT2,X=>OUT3);
end architecture LogicOperation;

Remark: the port map makes all the connections for the logic function using the operator =>.

VHDL Concurrency

- Its statements can be processed simultaneously.
- Concurrency applies to statements in between the begin and end statements in the architecture.
- Concurrent statements: the order in which they appear doesn't matter.

entity combinational is
  port (A,B,C,D: in bit; X,Y: out bit);
end entity combinational;

architecture example of combinational is
begin
  X <= (A and B) or not C;
  Y <= C or not D;
end architecture example;

VHDL Processes and Variables

- A process can be used to execute statements sequentially as well as concurrently.
  Name: process (sensitivity list) is
  Declarations
  begin
  Sequential statements
  end process;
  Sensitivity list is a set of signals to which the process is sensitive.
  The sequential statements are executed in the order in which they appear whenever any event occurs on any of these signals.
  Variables is a type of object that holds data. The other object that holds data is the signal. Variables can be used only in the process, while signal can be used inside or outside of a process.

  Signal assignment uses operator <=
  X <= A
  Variable assignment uses the operator :=
  Y := B

  A variable may be declared between process and begin.
Process example

entity example2 is
  port(A,B,C,D: in bit; X, Y: out bit);
end entity example2;

architecture Logic_ex of example2 is
begin
  process(A,B,C,D)
  variable V1: bit;
  begin
    V1:=(A and B) or (A and C) or
    (A and D) or (B and C) or
    (B and D) or (C and D);
    X<= V1; -- X is assigned the value V1
    Y<= not V1; -- Y is assigned the complemented value of V1
  end process;
end architecture Logic_ex;

Conditional Statements

- The if and case statements will be examples of conditional statements.
- The if and case statements are sequential statements.
- The if statement
  If conditional statement then
    VHDL statements
  end if;

Operators

- Arithmetic operators: +, -, *, /
  Example: W:= 4+5; X:= 3-2; Y := 5*6;
  Z:= 10/2;
- Relational operators: =, /=, <, <=, >, >=
- Logic: and, or, not, nand, nor, xor, xnor

The If Statement

entity Example1 is
  port(A,B: in bit; X:out natural range 0 to 7);
end entity Example1;

architecture MyExample of Example1 is
begin
  process(A,B)
  variable V1:natural range 0 to 7;
  begin
    V1 :=6;
    if (A=B) then
      V1 := V1+1;
    end if;
    X <= V1;
  end process;
end architecture MyExample;
The if-then-else statement

entity Example2 is
  port(A,B: in bit; X: out natural range 0 to 7);
end entity Example2;

architecture MyExample2 of Example2 is
begin
  process(A,B)
  variable V1:natural range 0 to 7;
  begin
    V1:=6;
    if(A=B) then
      V1:=V1+1;
    else
      V1:=0;
    end if;
    X<=V1;
  end process;
end architecture MyExample2;

elsif is used for making multiple choices, e.g., elsif (A = "0") then X <= 0;

The Case Statement

- It's an alternative to the nested if statement.
- The case statement must have all the possible values of the case block.
- The case statement case expression is
  when option 1 =>
    VHDL statements;
  when option 2 =>
    VHDL statements;
  when others =>
    VHDL statements;
  end case;

when others gives an alternate path for any cases not covered with a previous when clause.

The case statement example

entity Decoder is
  port (A: in bit_vector (0 to 1); X: out bit_vector (0 to 2));
end entity Decoder;

architecture Behavior of Decoder is
begin
  process (A)
  begin
    case A is
      when "01" =>
        X <= "001";
      when "10" =>
        X <= "010";
      when "11" =>
        X <= "100";
      when others =>
        X <= "000";
    end case;
  end process;
end architecture Behavior;