ECE 526 Digital Circuit Design II

VHDL

Presented by Chunyan Liu
July 18, 2011
## Introduction

### VHDL vs. Verilog

<table>
<thead>
<tr>
<th></th>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation</td>
<td>Compile</td>
<td>Interpretative</td>
</tr>
<tr>
<td>Libraries</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Resuability</td>
<td>Package</td>
<td>Include</td>
</tr>
<tr>
<td>Readability</td>
<td>ADA</td>
<td>C &amp; ADA</td>
</tr>
<tr>
<td>Easy to Learn</td>
<td>Less intuitive</td>
<td>Easy</td>
</tr>
</tbody>
</table>
Overview

- Introduction
  - History of VHDL
  - VHDL vs. Verilog
  - VHDL Advantages

- Basic Constructs
  - VHDL Syntax
  - Basic Operators
  - Objects
  - Types

- Behavioral VHDL
  - Concurrent Signal Assignments
  - Process Statements

- Application Examples
Introduction

History

VHDL is an acronym for the VHSIC Hardware Description Language. VHSIC is an acronym for the US Department of Defense Very High Speed Integrated Circuits Program. At the beginning, VHDL was only used for documentation, and quickly was adopted for simulation and synthesis.

1981—VHDL was developed by the Department of Defense, as a language to describe the structure and function of hardware.

1987—VHDL was standardized by IEEE. (IEEE 1076)

1993—VHDL was adopted by IEEE.

Introduction

VHDL Advantages

- Clear definition of design requirements
- Efficiency in design cycle time
- Flexibility in adapting to design changes
- Reuse of designs and packages
- Technology independent
- Easy analysis of various architecture/implementations
- Design verification and auto-regression tests
- VHDL is recommended for government contracts
- VHDL commercial models are available for purchase
Basic Constructs

VHDL Syntax

The basic unit of VHDL design is called a block. It has inputs and outputs specified in an entity declaration. An architecture body defines what the block does.

VHDL separates the architecture body from the entity declaration to allow multiple for a single block, and permits selecting among multiple architectures for a single entity.
Basic Constructs

VHDL Syntax

- Entity
- Architecture 1
- Architecture N
- process
- subprograms
- block
- procedure
- function
Basic Constructs

VHDL Syntax

Entity:

Entity <name> is
[ports illustration]
End Entity;

the syntax of port illustration is:

port(port name1... port name N: direction: type)

Direction: in, out, inout, buffer, linkage
Basic Constructs

VHDL Syntax

Architectures:
Architecture <name> of <block> is
  begin
    [body goes here]
  end <name> ;
Basic Constructs
Example:

library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all;
entity count is
  port ( clock,reset: in STD_LOGIC;
         dataout: out STD_LOGIC_VECTOR (3 downto 0)   );
end count;
architecture behavioral of count is
begin
  dataout<=databuffer;
  process(clock,reset)
    begin
      if (reset='1') then databuffer<="0000";
      elsif (clock'event and clock='1') then
        if databuffer="1000" then
          databuffer<="0000";
        else databuffer<=databuffer+'1';
        end if;
        end if;
    end process;
end behavioral;
Basic Constructs

VHDL Syntax

In the beginning of the example, standard libraries are used to define the data types and some other attributes. And many CAD vendors have adopted freely available libraries to support the standard libraries.

The syntax to load a library and use all of its functions is:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
```
Basic Constructs

Basic Operators

VHDL defines a number of operators:

- Multiplying Operators: *, /, mod, rem
- Adding Operators: +, –
- Relational Operators: =, /=, <, <=, >, >=
- Logical Operators: not, and, or, nand, nor, xor, xnor
Basic Constructs

Basic Operators

Example:

library IEEE; use IEEE.STD_LOGIC_1164.all
entity gates is
  port( a, b: in STD_LOGIC_VECTOR (3 downto 0)
       y1, y2, y3, y4, y5: out STD_LOGIC_VECTOR (3 downto 0));
end;

architecture synth of gates is
begin
  --Five different two-input logic gates acting on 4 bit busses
  y1 <= a and b;
  y2 <= a or b;
  y3 <= a xor b;
  y4 <= a nand b;
  y5 <= a nor b;
end;
Basic Constructs
Basic Operators

The operator precedence of VHDL is similar to those used in most programming languages or in mathematics. There is only one point we should pay more attention: AND does not take place before OR in VHDL, unlike what we would expect in a conventional Boolean equation.

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>Higest</td>
</tr>
<tr>
<td>*, /, mod, rem</td>
<td></td>
</tr>
<tr>
<td>+, -, &amp;</td>
<td></td>
</tr>
<tr>
<td>=, /=, &lt;, &lt;=, &gt;, &gt;=</td>
<td></td>
</tr>
<tr>
<td>=, ==, !=</td>
<td></td>
</tr>
<tr>
<td>and, or, nand, nor, xor</td>
<td>Lowest</td>
</tr>
</tbody>
</table>
Basic Constructs

Objects

Objects used in VHDL:

- Constant: can not be evaluated in procedures
- Variable: can be evaluated in procedures (\(\text{:=}\)), and changed to the new value as soon as evaluation. Only can be defined in process and subprogram.
- Signal: can be evaluated in procedures (\(\text{<=}\)), but does not changed to the new value until the process is suspended. Can not be defined in process and subprogram.
Basic Constructs

Objects

- Attributes of Object
  is similar to other object oriented programming languages, such as VB, VC, DELPHI

- The syntax of attribute definition:
  `object'attribute`
  
  Example: `clk'event`  -- the attribute of the signal `clk` is event
Basic Constructs

Objects

- Stock Attributes of Signal:
  - event: return Boolean value, return true when signals are changed.
  - last_value: return the value before signals changing.
  - last_event: return the interval space between last change and current change.
  - delayed: return the delay of the signal.
  - stable: return Boolean value, if the signal is not changed between the given time, return true.
Example:

Event and last_value are usually used to determine the rising edge and falling edge of a signal.

To determine the rising edge of the signal:

```
if (clk'event) and (clk='1') and (clk'last_value='0') then
```

To determine the falling edge of the signal:

```
if (clk'event) and (clk='0') and (clk'last_value='1') then
```
Basic Constructs

Types

There are ten standard data types we can use directly in VHDL:

- **bit**: `0`, `1`
- **bit-Vector**: `00110`
- **Boolean**: `true`, `false`
- **time**: `1 us`, `100 ms`, `3 s`
- **character**: `'a`, `'n`, `'1`, `'0`
- **string**: such as “sdfsdf”, “my design”
- **integer 32bit**: `1`, `234`, `-2134234`
- **real**: `-1.0E38`~`+1.0E38`, such as `1.0`, `2.834`, `3.14`, `0.0`
- **natural**: natural number
- **severity level**: usually used with assert clauses, including note, warning, error, failure
Basic Constructs
Types

We also can define data types by ourselves:
The syntax of type definition:

```plaintext
type <name> is <the definition of the type>
```

The types we used to define are:
enumeration, integer, real, array, record, time, file, access
Basic Constructs

Types

- enumeration

Syntax

```haskell
type <name> is (element1,element2......);
```

Examples:

```haskell
type week is (sun,mon,tue,thu,fri,sat);
type std_logic is (‘1’,’0’,’x’,’z’);
```
Basic Constructs
Types

- Integer and Real
  Syntax
  type <name> is <type range>;
  Examples:
  type week is integer range 1 to 7;
  type current is real range –1E4 to 1E4
Basic Constructs
Types

- Array
  Syntax
  type <name> is array (range) of <type>
  Examples:
  type week is array (1 to 7) of integer;
  type deweek is array (1 to 7) of week;
Basic Constructs
Types

- Time

Syntax:
type <name> is range
   units ultimate unit;
   unit;
end units

Example:
type time is range –1E18 to 1E18
   units
   us;
   ms=1000 us;
   sec=1000 ms;
   min=60 sec;
end units
Behavioral VHDL

There are two general styles of descriptions:

- **Structural VHDL** – describes how a module is composed of simpler modules or basic primitives such as gates or transistors.
- **Behavioral VHDL** – describes how the outputs are computed as functions of the inputs. There are two types of statements used in behavioral VHDL.
  - -- Concurrent signal assignment implies combinational logic, the execution order of statements are independent with the writing order.
  - -- Processes can imply combinational logic or sequential logic, depending how they are used. When the processes are used to imply sequential logic, the execution order of statements are decided by the writing order of the clauses.
Behavioral VHDL
Concurrent Signal Assignments

- Signal evaluation

Icon”<=” is used to evaluate the value to a signal. It can be used both in concurrent states and processes.

-- in concurrent states, when the value of the right side of “<=” is changed, a new value will be evaluated to the signal. If there is no change to the right side of “<=”, this clause will not be executed.

-- It is not applicable for processes.
Behavioral VHDL

Concurrent Signal Assignments

Examples:

Myblock: Block

begin
  clr <= '1' after 10 ns;
  clr <= '0' after 20 ns;
end block myblock

process
begin
  clr <= '1' after 10 ns;
  clr <= '0' after 20 ns;
end process
Behavioral VHDL

Concurrent Signal Assignments

- Conditional Signal Assignment States
  Syntax
    target signal <= expression1 when condition1
    else expression2 when condition2
    else expression3 when condition3
    ....
    else expression4

architecture synth of mux4 is
begin
    y<=d0 when s=‘00’ else
d1 when s=‘01’ else
d2 when s=‘10’ else
d3;
end;
Behavioral VHDL

Concurrent Signal Assignments

- Selected Signal Assignment Statements
  with expression select
    target signal <= expression 1 when condition1,
    expression 2 when condition2,
    ..... 
    expression n when condition n;

Example:
architecture synth2 of mux4 is 
beg
  with s select y <=
    d0 when “00”
    d1 when “01”
    d2 when “10”
    d3 when “11”
end;
Behavioral VHDL

Process Assignments

--Process assignment can implement flip-flops, latches, counters, memories and combinational logic.

-- the clauses used in process are:
- wait statement
- assert statement
- If statement
- case statement
- for loop statement
- while statement
Behavioral VHDL

Process Assignments

- Wait statement

Syntax

```vhdl
wait; --wait for ever
wait on [signal list]; --wait until the signal is changed
wait until [condition]; --wait until the condition is reached
wait for [time value]; --wait until the required time is reached
```

Example:
```
process
begin
  wait on a, b;
y<=a and b;
end process
```
Behavioral VHDL

Process Assignments

- Assert statement

Syntax

    assert condition [report output message] [severity]

If the condition is true, execute the next clause; otherwise report error

Example:

    assert (sum=100) report "sum /=100" severity error;
    next statement
Behavioral VHDL
Process Assignments

- If statement
  if condition1 then
    [serial execution]
  elsif condition2 then
    [serial execution]
  elsif condition3 then
    [serial execution]
  .....
  [else]
  end if

Examples:
process (A, B, C, X)
begin
  if (X= "0000") then
    Z<=A;
  elsif (X <="0101") then
    Z<=B;
  else
    Z<=C;
  end if;
end process;
Behavioral VHDL

Process Assignments

- Case statement
  Case expression is
    when condition expression => serial execution
    when condition expression => serial execution
    .......
    when condition expression => serial execution
  end case

Examples:

process (A, B, C, X)
begin
  case X is
    when 0 to 4 =>
      Z<=B;
    when 5 =>
      Z<=C;
    when 6 =>
      Z<=A;
    when others =>
      Z<=0;
  end case;
end process;
Application

Finite State Machine (FSM)

There are two styles of finite state machine.

Mealy FSM—the output is a function of the current state and inputs

Moore FSM— the output is a function of only the current state.

FSMs can be implemented in VHDL with process defining the state registers and combinational logic defining the next state and output logic.
Application

Example: MIPS Processor