Introduction to VHDL

Yvonne Avilés
Colaboration: Irvin Ortiz Flores
Rapid System Prototyping Laboratory (RASP)
University of Puerto Rico at Mayaguez
What is VHDL?

- **Very High Speed Integrated Circuit Hardware Description Language**
- A programming language designed and optimized for describing the behavior of digital systems.
- Allows for automatic circuit synthesis and system simulation.
- A standard in the electronic design community.
  - IEEE Standard 1076 defines the complete VHDL language
  - The IEEE 1076-1987 and IEEE 1164 standards together form the complete VHDL standard in widest use today
Every VHDL design description consists of at least one entity/architecture pair.

A VHDL entity is a statement that defines the external specification of a circuit or sub-circuit.

Provides the complete interface for a circuit
- Names, data types, direction of ports

```vhdl
entity compare is
  port( A, B: in bit_vector(0 to 7);
       EQ: out bit);
end compare;
```
Every referenced **entity** in a VHDL design description must be bound with a corresponding architecture.

The architecture describes the actual function—or contents—of the entity to which it is bound.

```vhdl
architecture compare1 of compare is
begin
    EQ <= '1' when (A = B) else '0';
end compare1;
```

Hierarchy and subprogram features of the language allow lower-level components, subroutines and functions in architectures; a **process** allows complex registered sequential logic as well.

VHDL allows more than one alternate architecture for an entity.
VHDL Architecture Structure

architecture name_arch of entity is
begin
  Signal assignments
Concurrent statements
Process 1
Concurrent statements
Process 2
Concurrent statements
end name_arch;

Processes contain sequential statements, but execute concurrently within the architecture body.
VHDL Process Syntax

\[ P1: \text{process} (\langle\text{sensitivity~list}\rangle) \]
\[ \langle\text{variable~declarations}\rangle \]
\[ \text{begin} \]
\[ \langle\text{sequential~statements}\rangle \]
\[ \text{end~process~P1;} \]

- Within a process: Variables are assigned using := and are updated immediately. Signals are assigned using <= and are updated at the end of the process.
Signals Vs Variables

**Signals**
- Used to connect components or to carry information between processes
- When inside of a process, its value is updated when the process suspends
- Signal assignment operator: <=

**Variables**
- Local to a process
- Not visible outside the process
- Values are updated immediately after the assignment
- Variable assignment operator: :=
### Signals Vs Variables

<table>
<thead>
<tr>
<th>Signals</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial values: A=5, X=10, B=15</td>
<td>Initial values: X=10</td>
</tr>
<tr>
<td>Final values: A=10, B=5</td>
<td>Final values: A=10, B=10</td>
</tr>
</tbody>
</table>

**Signals**

```
Sigproc:  
  process (A,X)  
  Begin  
    A <= X;  
    B <= A;  
  End process Sigproc;
```

**Variables**

```
Sigproc:  
  process (X)  
  Variable   A,B : integer;  
  Begin  
    A := X;  
    B := A;  
  End process Sigproc;
```
Like a high-level software programming language, data is represented in terms of high-level data types.

Every data type in VHDL has a defined set of values and valid operations.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>‘1’, ‘0’</td>
<td>Q&lt;=‘1’;</td>
</tr>
<tr>
<td>Bit_vector</td>
<td>(array of bits)</td>
<td>DataOut &lt;= &quot;00010101&quot;;</td>
</tr>
<tr>
<td>Boolean</td>
<td>True, False</td>
<td>EQ &lt;= True;</td>
</tr>
<tr>
<td>Integer</td>
<td>-2, -1, 0, 1, 2, 3, 4 . . .</td>
<td>Count &lt;= Count + 2;</td>
</tr>
<tr>
<td>Real</td>
<td>1.0, -1.0E5</td>
<td>V1 = V2 / 5.3</td>
</tr>
<tr>
<td>Time</td>
<td>1 ua, 7 ns, 100 ps</td>
<td>Q &lt;= ‘1’ after 6 ns;</td>
</tr>
<tr>
<td>Character</td>
<td>‘a’, ‘b’, ‘2’, ‘$’, etc.</td>
<td>CharData &lt;= ‘X’;</td>
</tr>
<tr>
<td>String</td>
<td>(Array of characters)</td>
<td>Msg &lt;= &quot;MEM: &quot; &amp; Addr</td>
</tr>
</tbody>
</table>
Design Units

- Entity : Interface
- Architecture : Implementation, behavior, function
- Configuration : Model chaining, structure, hierarchy
- Process : Concurrency, event controlled
- Package : Modular design, standard solution, data types, constants
- Library : Compilation, object code
Levels of Abstraction

- **Behavior**
  - Relies on processes to implement sequential statements

- **Dataflow**
  - *Describe circuit in terms of how data moves through the system.*
  - *Also referred to as register transfer logic,* or RTL.

- **Structure**
  - Used to describe a circuit in terms of its components
  - Requires hierarchical constructs

- **Mixed Method**
  - Any combination of methods
Entity Declaration

- Specifies the unit's ports
- States the port's name type, mode
- Ports can be in, out or inout
- Port size can be from a bit to a bit vector

Entity name
Entity Adder is

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>in std_logic</td>
<td>4 downto 0</td>
</tr>
<tr>
<td>Cin</td>
<td>in std_logic</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>out std_logic</td>
<td>4 downto 0</td>
</tr>
<tr>
<td>Cout</td>
<td>out std_logic</td>
<td></td>
</tr>
</tbody>
</table>

End Adder;
An Architecture Example

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

Entity Adder is
  port (A,B : in std_logic_vector(4 downto 0);
        Cin : in   std_logic;
        Sum : out std_logic_vector(4 downto 0);
        Cout : out std_logic);
End Adder;

architecture a_adder of adder is
signal AC,BC,SC : std_logic_vector(5 downto 0);
begin
  AC <= '0' & A;
  BC <= '0' & B;
  SC <= unsigned(AC) + unsigned(BC) + Cin;
  Cout <= SC(5);
  Sout <= SC(4 downto 0);
end a_adder;
```

- **Library declaration section**: Library declaration section contains the use of libraries.
- **Architecture declaration**: Architecture declaration is associated with the entity.
- **Signal declaration**: Signal declaration can also be placed component, constants, types, declarations.
- **Concurrent Statements**: Concurrent Statements are processed at the same time. Also component instantiations, and processes can be placed.
A “D” Flip-Flop

entity dff is
port ( d, clock : in bit;
       q: out bit);
end dff;

architecture arch of dff is
begin
  process (clock)
  begin
    if (clock'event and clock=1) then
      if (d='1') then
        q <= '1';
      else
        q <= '0';
      end if;
    end if;
  end process;
end arch;

Refers to the rising edge of the clock
Explicit comparations and assignments to port and signals uses ‘ ’ for one bit and “ ” for multiple bits
D Flip-Flop Behavioral

--Active low preset and clear inputs
entity dffpc2 is
  port(d,clock,clrn,prn:in bit;
    q,qbar:out bit);
end dffpc2;

architecture arch of dffpc2 is
begin
  process(clock,clrn,prn)
  begin
    if(clock'event and clock = '1') then
      q <= not prn or (clrn and d);
      qbar <= prn and (not clrn or not d);
    end if;
  end process;
end arch;
**D Flip-Flop Dataflow**

```vhdl
-- D flip-flop dataflow
-- Includes preset and clear
entity dff_flow is
  port (d, prn, clrn: in bit;
       q, qbar: out bit);
end dff_flow;

architecture arch1 of dff_flow is
begin
  q <= not prn or (clrn and d);
  qbar <= prn and (not clrn or not d);
end arch1;
```
D Flip-Flop Structural

```
---A two input nand gate
entity nandtwo is
    port(x, y: in bit;
         z: out bit);
end nandtwo;
architecture anandtwo of nandtwo is
begin
    z <= not(x and y);
end anandtwo;
```

Component declaration. Port appears exactly as in the entity declaration.

```
entity dff_str is
    port (d : in bit;
          q, qbar : out bit);
end dff_str;
architecture adff_str of dff_str is
component nandtwo
    port(x, y: in bit;
         z: out bit);
end component;
signal qbarinside, qinside, dbar: bit;
begin
    nandq: nandtwo
    port map(qinside, d, qbarinside);
    nandqbar: nandtwo
    port map(qbarinside, dbar, qinside);
    dbar <= not d;
    q <= qinside;
    qbar <= qbarinside;
end adff_str;
```

Entity name

Component instantiation label

Component instantiation. Connections are made by correspondence
Three-Bit Binary Counter

```vhdl
entity count1 is
    port( clock, enable: in bit;
         qa: out integer range 0 to 7);
end count1;

architecture countarch of count1 is
begin
    process (clock)
        variable count: integer range 0 to 7;
    begin
        if (clock'event and clock = '1') then
            if enable = '1' then
                count := count + 1;
            end if;
        end if;

        qa <= count after 10 ns;
    end process;
end countarch;
```

<table>
<thead>
<tr>
<th>Sensitivity list. Process is executed everytime one of this parameters change.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable declaration</td>
</tr>
<tr>
<td>Variable assignment operator</td>
</tr>
<tr>
<td>Sequential statements.</td>
</tr>
</tbody>
</table>

Signal assignment operator
D Flip-Flop with Asynchronous Preset and Clear

entity dffapc is
  port(clock, d, prn, clrn : in bit;
       q : out bit);
end dffapc;
architecture arch1 of dffapc is
begin
  process(clock, clrn, prn)
  variable reset, set: integer range 0 to 1;
  begin
    if (prn='0') then
      q <= '1';
    elsif (clrn='0') then
      q <= '0';
    elsif (clock'event and clock='1') then
      q <= d;
    end if;
  end process;
end arch1;

Integer range definition. Range 0 to 1 defines one bit.
Four Bit Adder

--A VHDL 4 bit adder
entity fourbadd is
   port (cin: in integer range 0 to 1;
         addendl: in integer range 0 to 15;
         addend2: in integer range 0 to 15;
         sum: out integer range 0 to 31);
end fourbadd;
architecture a4bitadd of fourbadd is
begin
   sum <= addendl + addend2 + cin;
end a4bitadd;

Integer type allows addition, subtraction and multiplication. Need the following statement at the library declaration section:

use IEEE.STD_LOGIC_ARITH.all
Synthesis

- Synthesis is a process of translating an abstract concept into a less-abstract form.
- The highest level of abstraction accepted by today’s synthesis tools is the dataflow level.
Display Driver

- **state_mach**
- **fsm**
- **clk**
- **reset**
- **stat[1:0]**
- **mux**
- **digit0[3:0]**
- **digit2[3:0]**
- **digit1[3:0]**
- **digit3[3:0]**
- **selec[1:0]**
- **muxout[3:0]**
- **bcd**
- **conv**
- **inbcd[3:0]**
- **outled[6:0]**
- **datout[6:0]**
- **selec[1:0]**
State Machine

\[ R[1:0] \]
\[ Q[1:0][1:2] \]
\[ D[1:0] \]
\[ un3\_count\_int[1:2] + 1 \]
\[ selec[1:0] \]
\[ reset \]
\[ clk \]
\[ stat[1:0] \]
Data Selector
VHDL can capture performance specification for a circuit, in the form of a test bench.

Test benches are VHDL descriptions of circuit stimuli and corresponding expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in tandem with other descriptions of the circuit.
Questions?
Link for VHDL Tutorial

http://www.aldec.com/Downloads/