International Journal for Research in Applied Science & Engineering Technology (IJRASET)

VHSIC-A higher approach

Chaynika Kapoor[1], Ankit Yadav[2], Apoorva Adlakha[3]

Abstract- VHDL - VHSIC Hardware Description Language. Here, VHSIC stands for Very High Speed Integrated Circuit* VHSIC Hardware Description Language *is a hardware description language used to describe a logic circuit by function, data flow behavior, or structure. It can also be used as a general purpose parallel programming language i.e. commands, which correspond to logic gates, are executed (computed) in parallel, as soon as a new input arrives. Like a program written in any language, can be executed, a program written in VHDL can be executed. But, since the language is used to model designs, it is always pronounced as simulated instead of getting executed. A language for describing the structural, physical and behavioral characteristics of digital systems. Execution of a VHDL program results in a simulation of the digital system allows us to validate the design prior to fabrication. The definition of the VHDL language provides a range of features that support simulation of digital systems VHDL supports both structural and behavioral descriptions of a system at multiple levels of abstraction. Structure and behavior are complementary ways of describing systems. A description of the behavior of a system says nothing about the structure or the components that make up the system. There are many ways in which you can build a system to provide the same behavior

KEYWORDS- RTL (Register Transfer Level), MUX (Multiplexer), RST (Reset), CLK (Clock)

I. INTRODUCTION

VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language. Which is one of the programming language used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC programe. In 1983 IBM, Texas Instruments and Intermetrics started to develop this language. In 1985 VHDL 7.2 version was released. In 1987 IEEE standardized the language. VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC is yet another achronym which stands for Very High Speed Integrated Circuits. If you can remember that, then you're off to a good start. The language has been known to be somewhat complicated, as its title (as titles go). The acronym does have a purpose, though; it is supposed to capture the entire theme of the language that is to describe hardware much the same way we use schematics.

VHDL can wear many hats. It is being used for documentation, verification, and synthesis of large digital designs. This is actually one of the key features of VHDL, since the same VHDL code can theoretically achieve all three of these goals, thus saving a lot of effort. In addition to being used for each of these purposes, VHDL can be used to take three different approaches to describing hardware. These three different approaches are the structural, data flow, and behavioral methods of hardware description. Most of the time a mixture of the three methods are employed. The following sections introduce you to the language by examining its use for each of these three methodologies. There are also certain guidelines that form an approach to using VHDL for synthesis, which is not addressed by this tutorial.VHDL is a standard (VHDL-1076) developed by IEEE (Institute of Electrical and Electronics Engineers). The language has been through a few revisions, and you will come across this in the VHDL community. Currently, the most widely used version is the 1987 (std 1076-1987) version, sometimes referred to as VHDL'87, but also just VHDL. However, there is a newer revision of the language referred to as VHDL'93. VHDL'93 (adopted in 1994 of course) is fairly new and is still in the process of replacing VHDL'87.

II. DESIGN

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a *test bench.* VHDL has constructs to handle the parallelism inherent in hardware designs, but these constructs (*processes*) differ in syntax from the parallel constructs in Ada (*tasks*). Like Ada, VHDL is strongly typed and is not case sensitive. In order to directly represent operations whi ch are

common in hardware, there are many features of VHDL which are not found in Ada, such as an extended set of Boolean operators including nand and nor. VHDL also allows arrays to be indexed in either ascending or descending direction; both conventions are used in hardware, whereas in Ada and most programming languages only ascending indexing is available. One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the RTLschematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate testbench. To generate an appropriate testbench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.[8]

A final point is that when a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip.

A. Describing a design

In VHDL an entity is used to describe a hardware module. An entity can be described using,

1) Entity declaration: It defines the names, input output signals and modes of a hardware module.

Syntax:
entity entity_name is
Port declaration;
end entity_name;

An entity declaration should starts with ‘entity’ and ends with ‘end’ keywords. Ports are interfaces through which an entity can communicate with its environment. Each port must have a name, direction and a type. An entity may have no port declaration also. The direction will be input, output or inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source.

2) Architecture: It describes the internal description of design or it tells what is there inside design. Each entity has atleast one architecture and an entity can have many architecture. Architecture can be described using structural, dataflow, behavioral or mixed style. Architecture can be used to describe a design at different levels of abstraction like gate level, register transfer level (RTL) or behavior level.

Syntax:
architecture architecture_name of entity_name
architecture_declarative_part;
begin
Statements;
end architecture_name;

Here we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

3) Configuration: If an entity contains many architectures and any one of the possible architecture binding with its entity is done using configuration. It is used to bind the architecture body to its entity and a component with an entity.

Volume 3 Issue IV, April 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Syntax:

```
configuration configuration_name of entity_name is
  block configuration_name _ configuration;
end .
```

Block_configuration defines the binding of components in a block. This can be written as

```
for block_name
  component_binding;
end for;
```

block_name is the name of the architecture body. Component binding binds the components of the block to entities. This can be written as,

```
for component_labels:component_name

  block_configuration;
end for;
```

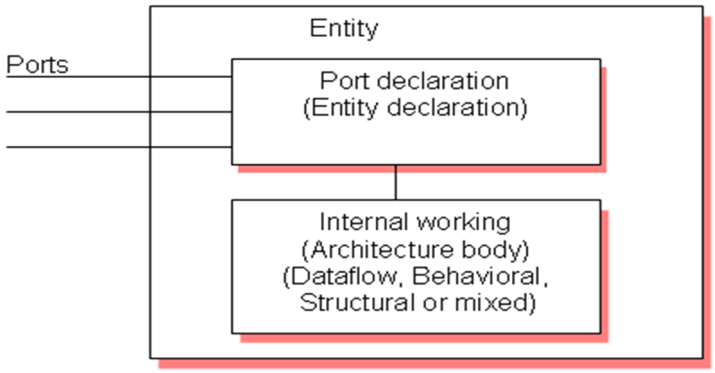

Fig.(a)Structure of Entity

B. Types of Modeling

1) Dataflow modeling: In this style of modeling, the internal working of an entity can be implemented using concurrent signal assignment.

Let's take half adder example which is having one XOR gate and a AND gate.

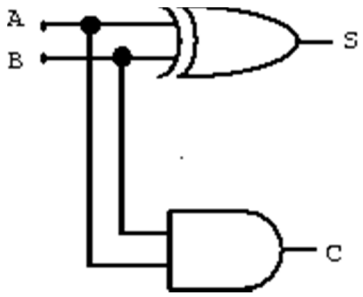

Fig.(b)Logical Diagram

Volume 3 Issue IV, April 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ha_en is
        port (A,B:in bit;S,C:out bit);
end ha_en;
architecture ha_ar of ha_en is
begin
        S<=A xor B;
        C<=A and B;
end ha_ar;
```

Fig.(c) Dataflow Model

Here STD_LOGIC_1164 is an IEEE standard which defines a nine-value logic type, called STD_ULOGIC. use is a keyword, which imports all the declarations from this package. The architecture body consists of concurrent signal assignments, which describes the functionality of the design. Whenever there is a change in RHS, the expression is evaluated and the value is assigned to LHS.

2) Behavioral modeling: In this style of modeling, the internal working of an entity can be implemented using set of statements.
It contains:

a) Process statements
b) Sequential statements
c) Signal assignment statements

Wait statements

Process statement is the primary mechanism used to model the behavior of an entity. It contains sequential statements, variable assignment (:=) statements or signal assignment (<=) statements etc. It may or may not contain sensitivity list. If there is an event occurs on any of the signals in the sensitivity list, the statements within the process is executed. Inside the process the execution of statements will be sequential and if one entity is having two processes the execution of these processes will be concurrent. At the end it waits for another event to occur.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ha_beha_en is
    port(
        A : in BIT;
        B : in BIT;
        S : out BIT;
        C : out BIT
        );
end ha_beha_en;
architecture ha_beha_ar of ha_beha_en is
begin
    process_beh:process(A,B)
    begin
       S<= A xor B;
       C<=A and B;
    end process process_beh;
end ha_beha_ar;
```

Fig.(d)Behavioral Modeling

Here whenever there is a change in the value of a or b the process statements are executed.

3) Structural modeling: The implementation of an entity is done through set of interconnected components. It contains:

a) Signal declaration.
b) Component instances
c) Port maps.
d) Wait statements.

Component declaration:

Syntax:

```
component component_name [is]
  List_of_interface ports;
end component component_name;
```

Before instantiating the component it should be declared using component declaration as shown above. Component declaration declares the name of the entity and interface of a component. Let's try to understand this by taking the example of full adder using 2 half adder and 1 OR gate.

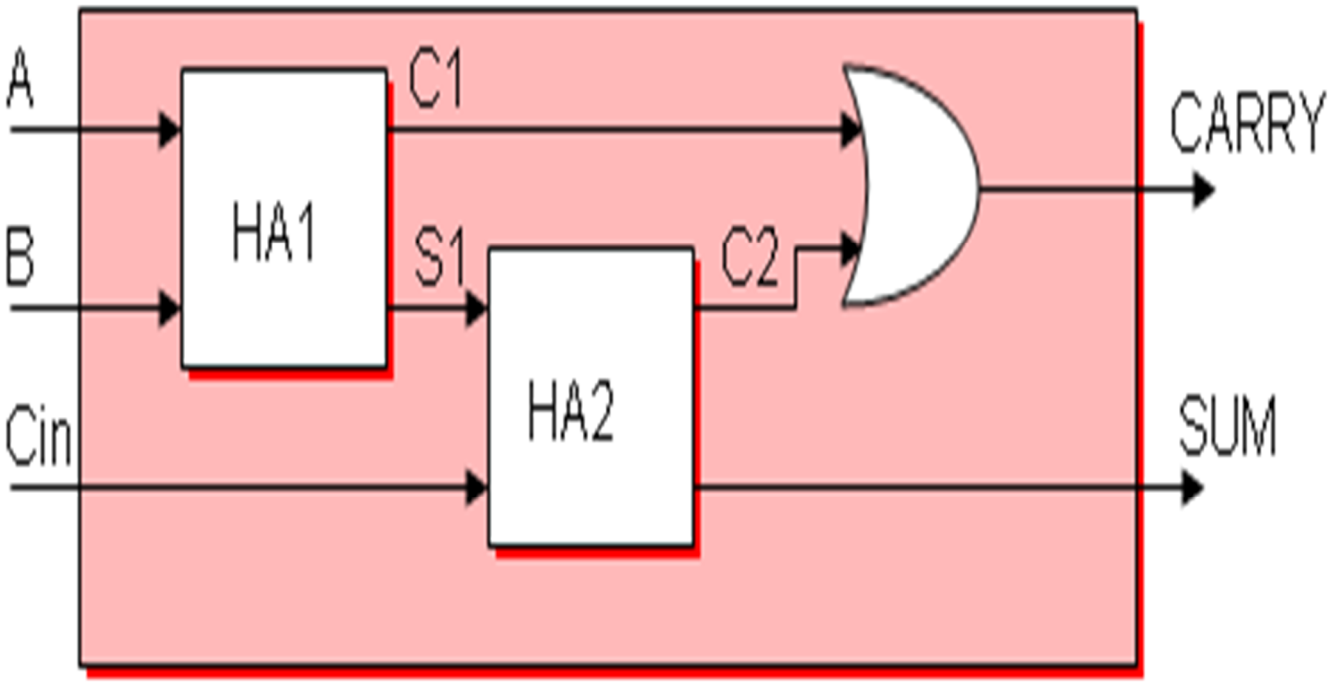

Fig.(e)Component Declaration

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity fa_en is
   port(A,B,Cin:in bit; SUM, CARRY:out bit);
end fa_en;


architecture fa_ar of fa_en is

component ha_en
   port(A,B:in bit;S,C:out bit);

end component;


signal C1,C2,S1:bit;

begin

   HA1:ha_en port map(A,B,S1,C1);
   HA2:ha_en port map(S1,Cin,SUM,C2);
   CARRY <= C1 or C2;

end fa_ar;
```

Fig.(f) Structural Modeling

The program we have written for half adder in dataflow modeling is instantiated as shown above. ha_en is the name of the entity in dataflow modeling. C1, C2, S1 are the signals used for internal connections of the component which are declared using the keyword signal. Port map is used to connect different components as well as connect components to ports of the entity.

Component instantiation is done as follows.

Component_label: component_name *port map* (signal_list);

Signal_list is the architecture signals which we are connecting to component ports. This can be done in different ways. What we declared above is positional binding. One more type is the named binding. The above can be written as,

HA1:ha_en port map(A => A,B => B, S => S1 ,C => C1)

HA2:ha_en port map(A => S1,B => Cin, S=> SUM, C => C2);

The correctness of the above program can be checked by writing the test bench.

4) Test bench: The test bench is used for generating stimulus for the entity under test. Let's write a simple test bench for full

adder.

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity tb_en is

end tb_en;

architecture tb_ar of tb_en is
signal a_i,b_i,c_i,sum_i,carry_i:bit;

begin

   eut: entity work.fa_en(fa_ar)
       port map(A=>a_i,B=>b_i,Cin=>c_i,SUM=>sum_i,CARRY=>carry_i);

stimulus: process
            begin
               a_i<='1';b_i<='1';c_i<='1';
               wait for 10ns;
               a_i<='0';b_i<='1';c_i<='1';
               wait for 10ns;
               a_i<='1';b_i<='0';c_i<='0';
               wait for 10ns;
               if now=30ns then
                  wait;
               end if;

end process stimulus;

end tb_ar;
```

Fig.(h)Test Bench Approach

Here now is a predefined function that returns the current simulation time What we saw upto this is component instantiation by positional and by name. In this test bench example the entity is directly instantiated. The direct entity instantiation syntax is:

Component_label: *entity* entity_name(architecture_name)
port map(signal_list);

III. ADVANTAGES

The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural

computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time.

A VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure).

A VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies.

A. Some of its benefits are

1) Executable specification
2) Validate spec in system context (Subcontract)
3) Functionality separated from implementation
4) Simulate early and fast (Manage complexity)
5) Explore design alternatives
6) Get feedback (Produce better designs)
7) Automatic synthesis and test generation (ATPG for ASICs)
8) Increase productivity (Shorten time-to-market)
9) Technology and tool independence (though FPGA features may be unexploited)
10) Portable design data

IV. SYNTHESIZABLE CONSTRUCTS AND VHDL TEMPLATES

VHDL is frequently used for two different goals: simulation of electronic designs and synthesis of such designs. Synthesis is a process where a VHDL is compiled and mapped into an implementation technology such as an FPGA or an ASIC. Many FPGA vendors have free (or inexpensive) tools to synthesize VHDL for use with their chips, where ASIC tools are often very expensive.
Not all constructs in VHDL are suitable for synthesis. For example, most constructs that explicitly deal with timing such as wait for 10 ns; are not synthesizable despite being valid for simulation. While different synthesis tools have different capabilities, there exists a common *synthesizable subset* of VHDL that defines what language constructs and idioms map into common hardware for many synthesis tools. IEEE 1076.6 defines a subset of the language that is considered the official synthesis subset. It is generally considered a "best practice" to write very idiomatic code for synthesis as results can be incorrect or suboptimal for non-standard constructs.

A. MUX template

The multiplexer, or 'MUX' as it is usually called, is a simple construct very common in hardware design. The example below demonstrates a simple two to one MUX, with inputs A and B, selector S and output X. Note that there are many other ways to express the same MUX in VHDL.

```
X <= A when S = '1' else B;
```

B. Latch template

A transparent latch is basically one bit of memory which is updated when an enable signal is raised. Again, there are many other ways this can be expressed in VHDL.

```
-- latch template 1:
Q <= D when Enable = '1' else Q;

-- latch template 2:
process(D,Enable)
begin
  if Enable = '1' then
    Q <= D;
  end if;
```

```
end process;
```

C. D-type flip-flops

The D-type flip-flop samples an incoming signal at the rising (or falling) edge of a clock. This example has an asynchronous, active-high reset, and samples at the rising clock edge.

```
DFF : process(RST, CLK)
begin
 if RST = '1' then
   Q <= '0';
 elsif rising_edge(CLK) then
   Q <= D;
 end if;
end process DFF;
```

Another common way to write edge-triggered behavior in VHDL is with the 'event' signal attribute. A single apostrophe has to be written between the signal name and the name of the attribute.

```
DFF : process(RST, CLK)
begin
 if RST = '1' then
   Q <= '0';
 elsif CLK'event and CLK = '1' then
   Q <= D;
 end if;
end process DFF;
```

VHDL also lend itself to "one-liners" such as

```
DFF : Q <= '0' when RST = '1' else D when rising_edge(clk);
```

or

```
DFF : process (RST, CLK) is
begin
 if rising_edge(CLK) then
   Q  <= D;
   Q2 <= Q1;
 end if;
 if RST = '1' then
   Q <= '0';
 end if;
end process DFF;
```

Which can be useful if not all signals (registers) driven by this process should be reset.

Example: a counter

The following example is an up-counter with asynchronous reset, parallel load and configurable width. It demonstrates the use of the 'unsigned' type, type conversions between 'unsigned' and 'std_logic_vector' and VHDL *generics*. The generics are very close to arguments or templates in other traditional programming languages like C++.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;    -- for the unsigned type
```

```
entity COUNTER is
  generic (
    WIDTH : in natural := 32);
  port (
    RST   : in std_logic;
    CLK   : in std_logic;
    LOAD  : in std_logic;
    DATA  : in std_logic_vector(WIDTH-1 downto 0);
    Q     : out std_logic_vector(WIDTH-1 downto 0));
end entity COUNTER;

architecture RTL of COUNTER is
  signal CNT : unsigned(WIDTH-1 downto 0);
begin
  process(RST, CLK) is
  begin
    if RST = '1' then
      CNT <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD = '1' then
        CNT <= unsigned(DATA); -- type is converted to unsigned
      else
        CNT <= CNT + 1;
      end if;
    end if;
  end process;

  Q <= std_logic_vector(CNT); -- type is converted back to std_logic_vector
end architecture RTL;
```

More complex counters may add if/then/else statements within the rising_edge(CLK) elsif to add other functions, such as count enables, stopping or rolling over at some count value, generating output signals like terminal count signals, etc. Care must be taken with the ordering and nesting of such controls if used together, in order to produce the desired priorities and minimize the number of logic levels needed.

V. VHDL SIMULATOR

A. *Commercial*

1) Aldec Active-HDL
2) Cadence Incisive (Past products: NC-VHDL)
3) Mentor Graphics ModelSim. Special versions of this product used by various FPGA vendors e.g. Altera, Lattice
4) Synopsys VCS-MX
5) Xilinx Vivado (a.k.a. xsim). Based on iSim from the previous ISE tool-chain. Xilinx Inc.

B. *Other*

1) Boot. from Free Range VHDL based on GHDL and GTKWave
2) GHDL. from ghdl.free.fr/, newer versions available SourceForge
3) Simili
4) Misc EDA Utilities Free VHDL Parser, vhdl2verilog, vhdl2ipxact and many other utilities
5) EDA Playground - Free web browser-based VHDL IDE (uses Riviera-PRO and ModelSim for VHDL simulation)

VI. CONCLUSION

VHDL is a very useful programming language to know and use. Although it only works on FPGA/CPLD type devices, it's important to note that these devices are quickly becoming very popular because of their concurrent execution. Parallel programming has been becoming a hot topic because FPGA/CPLDs can achieve what no processor can, even multi-core processors (Don't quote me though, people will argue this to death). The UP2 provides a very nice platform on which to learn VHDL better and for making complex programs Luckily for the UP1 (UP2) board we don't have to worry about hardware functionality as much since it is nicely laid out on a PCB. The 8-bit counter worked like a charm & made for a good supplemental introduction to VHDL.

REFRENCES

[1] 1076-1987 – IEEE Standard VHDL Language Reference Manual.
[2] 1076-1993 – IEEE Standard VHDL Language Reference Manual.
[3] 1076-2000 – IEEE Standard VHDL Language Reference Manual.
[4] 1076-2002 – IEEE Standard VHDL Language Reference Manual.
[5] 1076c-2007 – IEEE Standard VHDL Language Reference Manual Amendment 1: Procedural language Application Interface
[6] http://en.wikipedia.org/wiki/VHDL
[7] http://www.academia.edu/1492361/VHDL_BASICS_WITH_EXAMPLES
[8] http://www.gmvhdl.com/introduc.htm
[9] http://www.people.vcu.edu/~rhklenke/tutorials/vhdl/modules/m13_23/sld006.html
[10] http://www.pyroelectro.com/tutorials/vhdl_counter/conclusion.html
[11] https://www.doulos.com/knowhow/vhdl_designers_guide/benefits_of_using_vhdl/