Abstract—Digital design is an amazing and very broad field. The applications of digital design are present in our daily life, including computers, calculators, video cameras etc. The VHDL (VHSIC Hardware Description Language) has become an essential tool for designers in the world of digital design. This paper presents implementation of a 32-bit Arithmetic Logic Unit (ALU) using VHDL. Here the behavioral VHDL model of ALU is designed to perform 16 operations which includes both logical and arithmetic operations. The VHDL implementation and functionality test of the 32-bit ALU is done by using the Modelsim 5.4a tool.

Index Terms—32-bit ALU, VHDL, Behavioral

I. INTRODUCTION

The Arithmetic Logic Unit (ALU) is a fundamental building block of the Central Processing Unit (CPU) of a computer. Even one of the simplest microprocessor contains one ALU for purposes such as maintaining timers. We can say that ALU is a core component of all central processing unit within in a computer and is an integral part of the execution unit. ALU is capable of calculating the results of a wide variety of basic arithmetical and logical computations. The ALU takes as input the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. The output is the result of the computation. The ALU implemented will perform the following operations:

- Arithmetic operations (addition, subtraction, increment, decrement, transfer)
- Logic operations (AND, NOT, OR, NAND, NOR, EX-OR, EX-NOR)

A digital system can be represented at different levels of abstraction [1]. This keeps the description and design of complex systems manageable. The highest level of abstraction is the behavioral level that describes a system in terms of what it does (or how it behaves) rather than in terms of its components and interconnection between them.

Here the 32-bit ALU is implemented by using the behavioral modeling style to describe how the operation of ALU is being processed. This is accomplished by using a hardware description language VHDL.

The behavioral style makes use of a process statement. A process statement is the main construct in behavioral modeling that allows using sequential statements to describe the behavior of a system over time. Process is declared within an architecture and is a concurrent statement. However, the statements inside a process are executed sequentially. A process do read and write signals and values of the interface (input and output) ports to communicate with the rest of the architecture just like other concurrent statements.

In this paper, section II deals with modeling style, block diagram, specifications and VHDL code for 32-bit ALU, section III presents simulation results and discussion and section IV follows conclusions.

II. MODELING STYLE, BLOCK DIAGRAM, SPECIFICATIONS AND VHDL CODE FOR 32-BIT ALU

A. Modeling Style

High level design methodology allows managing the design complexity in a better way and reduces the design cycle without continuing the trend to compromise evaluation of design implementation options [2]. A high-level model makes the description and evaluation of the complex systems easier. The behavioral capabilities of VHDL can be more powerful and more convenient for this design. The uniqueness of ALU is that it is built entirely of combinational Logic, without any latches or registers. The combinational logic can be described by logical equations, sequential control statements (like CASE, IF then Else, and so on), subprograms, or through concurrent statements [3].

Designing at a higher level of abstraction delivers the following benefits [2]:

---

Geetanjali is with Department of Electronics Design & Technology, National Institute of Electronics and Information Technology Gorakhpur, India, 273010.(e-mail: ergeetanjali2009@gmail.com )

Nishant Tripathi is with Department of Electronics Design & Technology, National Institute of Electronics and Information Technology Gorakhpur, India, 273010.(e-mail: nishantri@gmail.com).
- Manages complexity: Fewer lines of code improves productivity, reduces error.
- Increases design reuse: Implementation of independent designs.
- Improves verification: Starts earlier in process runs significantly faster.

**B. Block Diagram and Specifications for 32-bit ALU**

The following block diagram shows the input and output signals of the ALU to be implemented which is followed by a table of the required ALU specifications:

![Block Diagram for 32-bit ALU](image)

**C. VHDL Code for the 32-Bit ALU**

The following portion of the code describes the behavioral description that was used for implementing the above specifications of 32-bit ALU:

```vhdl
entity ALU is
generic (width: integer:=32);
port ( A: in std_logic_vector((width-1) downto 0);
      B: in std_logic_vector((width-1) downto 0);
      Sel: in std_logic_vector(3 downto 0);
      Y: out std_logic_vector((width-1) downto 0)
    );
end ALU;
architecture behv of ALU is
begin
  process(A,B,Sel)
  begin
    case Sel is
      when "0000" =>
        Y <= A + B;
      when "0001" =>
        Y <= A + (not B) + 1;
      when "0010" =>
        Y <= A + 1;
      when "0011" =>
        Y <= A - 1;
      when "0100" =>
        Y <= B + 1;
      when "0101" =>
        Y <= B - 1;
      when "0110" =>
        Y <= A;
      when "0111" =>
        Y <= B;
      when "1000" =>
        Y <= not A;
      when "1001" =>
        Y <= not B;
      when "1010" =>
        Y <= A and B;
      when "1011" =>
        Y <= A or B;
      when "1100" =>
        Y <= A nand B;
      when "1101" =>
        Y <= A nor B;
      when others =>
        Y <= "XXXX";
    end case;
  end process;
end behv;
```

**Table- I ALU Specifications**

<table>
<thead>
<tr>
<th>SEL</th>
<th>OPERATION</th>
<th>FUNCTION</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Y &lt;= A+B</td>
<td>Addition</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0001</td>
<td>Y &lt;= A+B'+1</td>
<td>Subtraction</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0010</td>
<td>Y &lt;= A+1</td>
<td>Increment A</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0011</td>
<td>Y &lt;= A - 1</td>
<td>Decrement A</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0100</td>
<td>Y &lt;= B+1</td>
<td>Increment B</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0101</td>
<td>Y &lt;= B - 1</td>
<td>Decrement B</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0110</td>
<td>Y &lt;= A</td>
<td>Transfer A</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>0111</td>
<td>Y &lt;= B</td>
<td>Transfer B</td>
<td>ARITHMETIC</td>
</tr>
<tr>
<td>1000</td>
<td>Y &lt;= not A</td>
<td>NOT A</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1001</td>
<td>Y &lt;= not B</td>
<td>NOT B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1010</td>
<td>Y &lt;= A and B</td>
<td>A AND B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1011</td>
<td>Y &lt;= A or B</td>
<td>A OR B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1100</td>
<td>Y &lt;= A nand B</td>
<td>A NAND B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1101</td>
<td>Y &lt;= A nor B</td>
<td>A NOR B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1110</td>
<td>Y &lt;= A xor B</td>
<td>A EX-OR B</td>
<td>LOGIC</td>
</tr>
<tr>
<td>1111</td>
<td>Y &lt;= not (A xor B)</td>
<td>A EX-NOR B</td>
<td>LOGIC</td>
</tr>
</tbody>
</table>

**Fig 1: Block Diagram for 32-bit ALU**

Now the table for the required specifications of ALU is as following:
For example:
When SEL = 0001,
A = 1011100010111000001000111101110,
B = 0010011110001000100000100100001
Then the output of the ALU is
Y = 10010001100010001001000011001101.
Here the arithmetic unit got selected.

When SEL = 1100,
A = 1011100010111000001000111101110,
B = 00100111100010001
Then the output of the ALU is
Y = 1101111111001111111111101111111.
Here the logic unit got selected.

In the above description we have used the behavioral
method for the purpose of VHDL implementation. While
describing the entity we have used GENERIC since the
generic lets us to produce code that can be changed by the
passed parameter. Here we let the generic width be the bit
width of the two operands, which made the code scalable
to any width of signals. With the use the generic width,
the component became much more reusable which is the
quality always desired by designers. When this style of
description is used to describe systems, it is very powerful
but it contains no structural information regarding the
design.

The main construct in this modeling style is the process
statement. Another important construct here are the case
and when statement. The case statement executes many
sequences of statements, based on the value of a single
expression. Also the case statement does evaluation of the
expression and comparison of the value to each of the
choices while the when clause corresponding to the
matching choice will make its statements to get executed.

The expressive power of VHDL provides us with the
capability of writing testbench model in the same
Language. A testbench has three main purposes viz. one is
to generate stimulus for simulation, second is to apply this
stimulus to the entity under test and collect the output
responses and third is to compare the output responses
with the artificial inputs given.

III. SIMULATION RESULTS AND DISCUSSION

It is needed to test whether the design works to meet
the given specification to ensure that designed entity is
correct. This is verified by the process of simulation. The
process of simulation uses a testbench to test the design
whether it behaves correctly by stimulating it with
artificial input and monitoring the output. The simulation
is carried out by using the Modelsim 5.4a tool and having
the testbench and the behavioral design code for 32-bit
ALU in the same project folder.

We observed from the simulation results that the 32-bit
ALU implemented by the above described method and
code, worked successfully for all the input combinations
and the select codes according to the given specification in
Table I.

The Modelsim simulation results are shown below in
Fig.2

![Simulation Result for Arithmetic Unit](image1)

![Simulation Result for Logic Unit](image2)
IV. Conclusion

This paper suggests the behavioral design method for VHDL implementation of a 32-bit ALU using Modelsim 5.4a tool. Its functionality was discussed for all the operations specified. As per the nature of behavioral description, it is easy to convert the precision to 64-bit or more. This behavioral design can be made synthesizable and thus can be used for layout and fabrication on FPGA based digital circuits.

V. References