International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 2, Issue 10 [Jun. 2013] PP: 72-77

VLSI Design and Implementation of Arithmetic and Logic Unit Using VHDL

Tiwary[1], B rajeshwar[2], D matsyagiri[3]
[1,2,3] *ECE DEPT,SREE DATTHA INSTITUTE OF ENGINEERING & SCIENCE*

Abstract: *The main objective of our project is to design a 32 bit Arithmetic Logic Unit which is a digital circuit that performs arithmetic and logical operations using VHDL. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs. This behavioral design can be made synthesizable and thus can be used for layout and fabrication on FPGA based digital circuits.*

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. 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 architecture and is a concurrent statement. However, the statements inside a process are

II. PROPOSED SYSTEM

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 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. 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.

III. TO MIGRATE A PROJECT

- In the ISE 12 Project Navigator, select File > Open Project.
- In the Open Project dialog box, select the .xise file to migrate.

Note: You may need to change the extension in the Files of type field to display .npl (ISE 5 and ISE 6 software) or .ise (ISE 7 through ISE 10 software) project files.

- In the dialog box that appears, select Backup and Migrate or Migrate Only.
- The ISE software automatically converts your project to an ISE 12 project.

Note: If you chose to Backup and Migrate, a backup of the original project is created at project_name_ise12migration.zip.

- Implement the design using the new version of the software.

Note: Implementation status is not maintained after migration.

IV. OPERATING SYSTEM SUPPORT

Xilinx officially supports Microsoft Windows, Red Hat Enterprise 4, 5, & 6 Workstations (32 & 64 bits) and SUSE Linux Enterprise 11 (32 & 64 bits). Certain other GNU distributions can run Xilinx ISE Web Pack with some modifications or configurations, including Gentoo Linux, Arch Linux, FreeBSD and Fedora.

1.1Properties:

For information on properties that have changed in the ISE 12 software, see ISE 11 to ISE 12 Properties Conversion.

1.2 IP Modules:

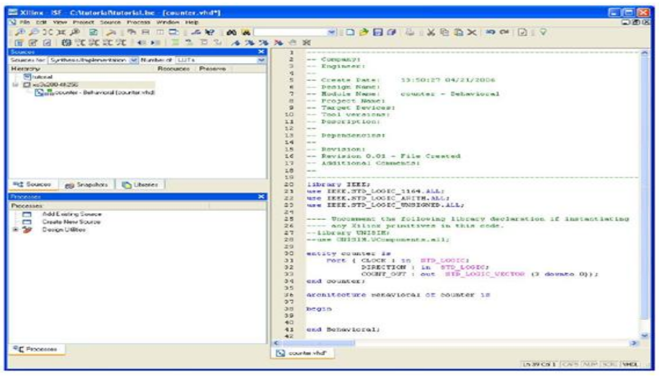
If your design includes IP modules that were created using CORE Generator software or Xilinx Platform Studio (XPS) and you need to modify these modules, you may be required to update the core. However, if the core net list is present and you do not need to modify the core, updates are not required and the existing netlist is used during implementation.

1.3Design Panel:

Project Navigator manages your project based on the design properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx device. For information on changing design properties, see Changing Design Properties.

You can now perform any of the following:

1. Create new source files for your project.
2. Add existing source files to your project.
3. Run processes on your source files. Modify process properties.


Xilinx - ISE - C:\tutorial\tutorial.ise - [counter.vhd*]
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Implementation
Hierarchy
tutorial
xc3s200-4ft256
counter - Behavioral (counter.vhd)
Sources
Snapshots
Libraries
Processes
Add Existing Source
Create New Source
Design Utilities
counter.vhd*
Ln 39 Col 1
CAPS
NUM
SCRL
VHDL

```
------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:50:27 04/21/2006
-- Design Name:
-- Module Name:    counter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter is
    Port ( CLOCK : in  STD_LOGIC;
           DIRECTION : in  STD_LOGIC;
           COUNT_OUT : out  STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is

begin


end Behavioral;
```

Figure 1: Design Panel

New project in ISEUsing Language Templates (VHDL).The next step in creating the new source is to add the behavioral description for the counter. To do this you will use a simple counter code example from the ISE Language,Templates and customize it for the counter design.

1. Place the cursor just below the begin statement within the counter architecture.
2. Open the Language Templates by selecting Edit
Language Templates
Note*:* You can tile the Language Templates and the counter file by selecting
Vertically to make them both visible.
3. Using the "+" symbol, browse to the following code example:VHDL

- Synthesis Constructs
- Coding Examples
- Counters
- Binary
- Up/Down Counters
- Simple Counter

4. With Simple Counter selected, select Edit Use in File, or select the Use Template in File toolbar button. This step copies the template into the counter source file.
5. Close the Language Templates.Final Editing of the VHDL Source
1. Add the following signal declaration to handle the feedback of the counter output below the architecture declaration and above the first begin statement: signal count_int: std_logic_vector (3 downto 0):"0000";
2. Customize the source file for the counter design by replacing the port and signal name Place holders with the actual ones as follows:

- replace all occurrences of <clock> with CLOCK
- replace all occurrences of <count_direction> with DIRECTION
- replace all occurrences of <count> with count_int

3. Add the following line below the end process; statement:COUNT_OUT <= count_int;
4. Save the file by selecting File Save.When you are finished, the counter source file will look like the following:

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if instantiating
-- Any Xilinx primitive in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
Entity counter is
Port (CLOCK: in STD_LOGIC;
DIRECTION: in STD_LOGIC;
COUNT_OUT: out STD_LOGIC_VECTOR (3 downto 0));
End counter;
Architecture Behavioral of counter is
Signal count_int : std_logic_vector(3 downto 0) := "0000";
Begin
Process (CLOCK)
Begin
If CLOCK='1' and CLOCK 'event then
If DIRECTION='1' then
count_int <= count_int + 1;
else
count_int <= count_int - 1;
End if;
End if;
End process;
COUNT_OUT <= count_int;
End Behavioral;
```

You have now created the VHDL source for the tutorial project. Skip past the Verilog Sections below, and proceed to the "Checking the Syntax of the New Counter Module "section.

V. DESIGN OF 32-BIT ALU

When designing the ALU we will follow the principle "Divide and Conquer" in order to use a modular design that consists of smaller, more manageable blocks, some of which can be re-used. Instead of designing the 4-bit ALU as one circuit we will first design one bit ADDER, SUBTRACTOR, OR, AND, NOT, XOR, LEFT SHIFT, RIGHT SHIFT UNIT. These bit-slices can then be put together to make a 32-bit ADDER, SUBTRACTOR, OR, AND, NOT, XOR, LEFT SHIFT, RIGHT SHIFT UNIT.

5.1 32 bit Arithmetic unit:

An Arithmetic unit does the following task: Addition, Addition with carry, Subtraction, Subtraction with borrow, Decrement, Increment and Transfer function. Now first of all we start with making one bit Full Adder, then a 4-bit Ripple Carry Adder using four numbers of Full Adder and at last a 32-bit Ripple Carry Adder using eight numbers of 4-bit Ripple Carry Adder. Then we have designed thirty two numbers of single-bit 4:1 Multiplexer. The diagram of a 32-bit Arithmetic Unit is shown in Fig. 6. The circuit has a 32-bit parallel adder and thirty two multiplexers for 32-bit arithmetic unit. There are two 32-bit inputs A and B and 33-bit output is RESULT. The size of each multiplexer is 4:1. The two common selection lines for all thirty two multiplexers are S_0 and S_1. C_in is the carry input of the parallel adder and the carry out is Cout. The thirty two inputs to each multiplexer are B- value, Complemented B-value, logic-0 and logic-1.
The output of the circuit is calculated from the following arithmetic sum:
RESULT = A + Y + Cin
Where A is a 32-bit number, Y is the 32-bit output of multiplexers and Cin is the carry input bit to the parallel adder.
When $S_1S_0 = 00$: if Cin = 0 then RESULT = A+B i.e. Addition. if Cin = 1 then RESULT = A+B+1 i.e. Addition with carry.
When S_1S_0 = 01: if Cin = 0 then RESULT = A-B i.e. Subtraction. if Cin = 1 then RESULT = A-B+1 i.e. Subtraction with borrow.
When S1S0 = 10: if Cin = 0 then RESULT = A-1 i.e. Decrement. if Cin = 1 then RESULT = A i.e. Transfer.
When S1S0 = 11: if Cin = 0 then RESULT = A i.e. Transfer. if Cin = 1 then RESULT = A+1 i.e Increment.

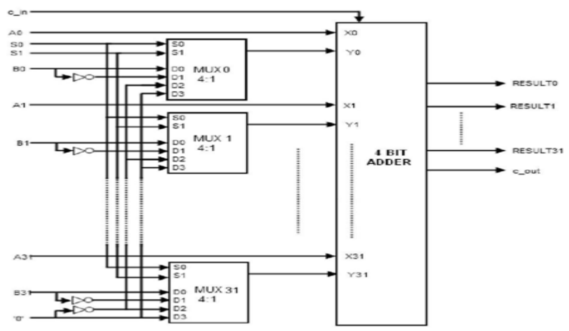


Figure 2: 32 bit Arithmetic unit

5.2 32 bit Logic Unit:

A Logic unit does the following task: Logical AND, Logical OR, Logical XOR and Logical NOT operation. We will design a logic unit that can perform the four basic logic micro-operations: OR, AND, XOR and Complement, because from these four micro-operations, all other logic micro-operations can be derived. A one-stage logic unit for these four basic micro-operations is shown in the Fig. 7. The logic unit consists of four gates and a 4:1 multiplexer. The outputs of the gates are applied to the data inputs of the multiplexer. Using to selection lines S_0 and S_1 one of the data inputs of the multiplexer is selected as the output. For a logic unit of 32-bit, the output will be of 33-bit with 33th bit to be High-impedance. The common selection lines are applied to all the stages.

When S1S0 = 00: RESULT = A.B i.e AND operation
When S1S0 = 01: RESULT = A+B i.e OR operation.
When S1S0 = 10: RESULT = $A \oplus B$ i.e XOR operation
When S1S0 = 11: RESULT = ~A i.e NOT operation.

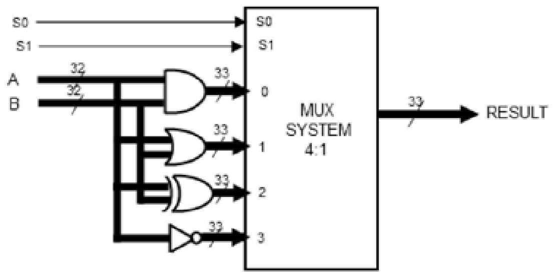


Figure 3: 32 bit Logic unit.

5.3 32 bit Shifter Unit:

Shifter unit is used to perform logical shift micro-operation. The shifting of bits of a register can be in either direction- left or right. A combinational shifter unit can be constructed as Fig. 7. The content of a register that has to be shifted first placed onto common bus. This circuit uses no clock pulse. When the shifting unit is activated the register is shifted left or right according to the selection unit. For a shift unit of 32-bit, the output will be of 33-bit with 33th bit to be the outgoing bit. The circuit of shift unit is shown in Fig. 7.3.
When S_1= 0: RESULT= Shift Right A.
When S_1= 1: RESULT= Shift Left A.

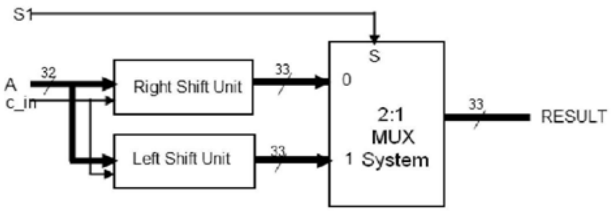


Figure 4: 32 bit shifter unit

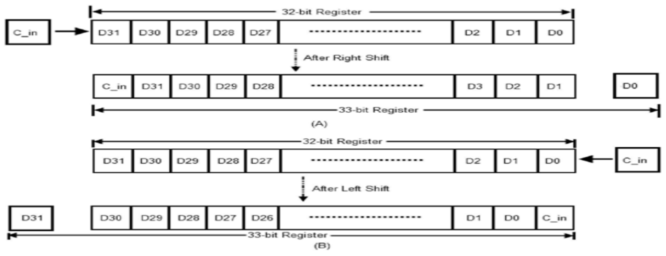


Figure 5: 32 bit right shift operation.

5.4 32 bit Arithmetic and logic unit:

The approach used here is to split the ALU into three modules, one Arithmetic, one Logic and one Shift module. The arithmetic, logic and shifter units introduced earlier can be combined into ALU with common selection lines. The shift micro-operations are often performed in a separate unit, but sometimes the shifter unit made part of overall ALU. Since the ALU is composed of three units, namely Arithmetic, Logic and Shifter Units. For 32-bit ALU a 33 bit 4:1 MUX is needed. A particular arithmetic or logic or shift operation is selected according to the selection inputs S_0 and S_1. The final output of the ALU is determined by the set of multiplexers with selection lines S_2 and S_3. The function table for the ALU is shown in the Table. 1. The table lists 14 micro-operations: 8 for arithmetic, 4 for logic and 2 for shifter unit. For shifter unit, the selection line S_1 is used to select either left or right shift micro-operation.

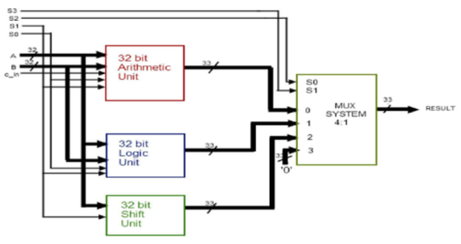
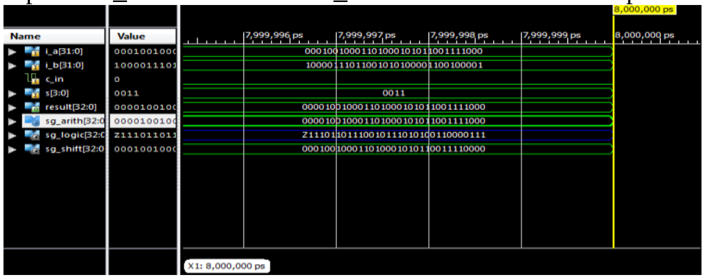


Figure 6: 32 bit arithmetic and logic unit.

VI. RESULTS

6.1Simulation Results:

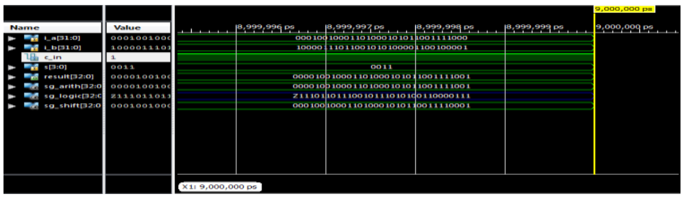
Here we have taken the inputs as i_a = 12345678 and i_b = 87654321. Both the inputs are taken in Hexadecimal



```
                                                                                              8,000,000 ps
Name            Value          |7,999,996 ps   |7,999,997 ps   |7,999,998 ps   |7,999,999 ps   |8,000,000 ps
i_a[31:0]       0001001000         00010010001101000101011001111000
i_b[31:0]       1000011101         10000111011001010100001100100001
c_in            0
s[3:0]          0011                              0011
result[32:0]    0000100100         000010010001101000101011001111000
sg_arith[32:0   0000100100         000010010001101000101011001111000
sg_logic[32:0   Z111011011         Z11101101110010111010100110000111
sg_shift[32:0   0001001000         000100100011010001010110011110000

X1: 8,000,000 ps
```

Figure 7: Transfer result when C_{in} = 0.



```
                                                                                              9,000,000 ps
Name            Value          |8,999,996 ps   |8,999,997 ps   |8,999,998 ps   |8,999,999 ps   |9,000,000 ps
i_a[31:0]       0001001000         00010010001101000101011001111000
i_b[31:0]       1000011101         10000111011001010100001100100001
c_in            1
s[3:0]          0011                              0011
result[32:0]    0000100100         000010010001101000101011001111001
sg_arith[32:0   0000100100         000010010001101000101011001111001
sg_logic[32:0   Z111011011         Z11101101110010111010100110000111
sg_shift[32:0   0001001000         000100100011010001010110011110001

X1: 9,000,000 ps
```

Figure 8: Increment result

VII. CONCLUSION AND FUTURE SCOPE

This paper suggests the behavioral design method for VHDL implementation of a 32-bit ALU using Xilinx 9.2i. 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.

In computing, an Arithmetic and logic unit is a digital circuit that performs arithmetic and logic operations. It will find its requirement in the field of Nanotechnology. Commercially it would be very useful in the smart mobile phones and calculating devices

REFERENCES

[1] D. Gajski and R. Khun, "Introduction: New VLSI Tools," IEEE Computer, Vol. 16, No. 12, pp. 11-14, Dec. 1983.

[2] http://www.forteds.com/behavioralsynthesis/index.asp

[3] Douglas L. Perry, VHDL, third edition, McGraw-Hill, pp. 60-63, 238, July 1999.

[4] S.Yalamanchali, "Introductory VHDL: From simulation to synthesis", Prentice Hall, United States, 2002.

[5] V.A.Pedroni, "Circuit design with VHDL", Cambridge, Massachusetts, London, England, 2004.

[6] P.J.Ashenden, "The VHDL Cookbook", University of Adelaide, South Australia, July, 1990.

[7] http://www.xilinx.com