A PROJECT REPORT ON

DESIGN AND IMPLEMENTATION OF A 32-BIT ALU ON XILINX FPGA USING VHDL

SUBMITTED BY

ANUSHKA PAKRASHI
ARINDAM BOSE
KAUSIK BHATTACHARYA
MONIGINGIR PAL
TANAYA BOSE

This report is submitted as part requirement for the B.TECH Degree in Electronics and Communications Engineering.

FUTURE INSTITUTE OF ENGINEERING AND MANAGEMENT

(2008-2012)

FUTURE

ACKNOWLEDGEMENT

It gives us great pleasure to find an opportunity to express our deep gratitude and sincerest thank to our project mentor, Mr. Abhisek Bakshi at Ardent Collaborations, Saltlake of Kolkata for giving most valuable suggestion, helpful guidance and encouragement in the execution of this project work. His guidance and cooperation has led us to the successful completion of our project titled: "Design and implementation of a 32-bit ALU on Xilinx ISE 9.2i using VHDL". We are highly indebted to him for the way he modeled and structured our work with his valuable tips and suggestion that he accorded to us in every respect of our work.

Last but not the least we humbly extend our sense of gratitude to other faculty members and staff of the institute for providing us their valuable help and time with a congenial working environment.

ANUSHKA PAKRASHI
ARINDAM BOSE
KAUSIK BHATTACHARYA
MONIGINGIR PAL
TANAYA BOSE

A Project Report

on

Design and implementation of a 32-bit ALU on Xilinx FPGA using VHDL

Submitted by

BONAFIDE CERTIFICATE

Certified that this project report **"Design and implementation of a 32-bit ALU on Xilinx FPGA using VHDL"** is the bonafide work of:

Name of the student:_____

Roll Number:_____

Registration number:_____

who carried out the project work under my supervision.

_____ _____

Signature of the Student **Signature of the Project Coordinator**

_____

Signature of the Examiners

Contents

ABSTRACT

In the present day technology, there is an immense need of developing suitable data communication interfaces for real time embedded systems. Field Programmable Gate Array (FPGA) offers various resources, which can be programmed for building up an efficient embedded system.

A Field-programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence it is named as "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL)

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. It became IEEE standard 1076 in 1987. It was updated in 1993 and is known today as "IEEE standard 1076 1993.

This report proposes a technique to design and implement a 32 bit ALU which is a digital circuit that performs arithmetic and logical operations on Xilinx ISE 9.2i using VHDL.

1. Objective

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.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under Arithmetic and logic structures in the ACM Computing Classification System.

Here, ALU is designed using 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.

2. Introduction

2.1. VHDL Quick Look

A digital system in VHDL consists of a design entity that can contain other entities that are then considered components of the top-level entity. Each entity is modeled by an entity declaration and an architecture body. So a VHDL module has tow parts namely, Entity and Architecture as Fig. 1.

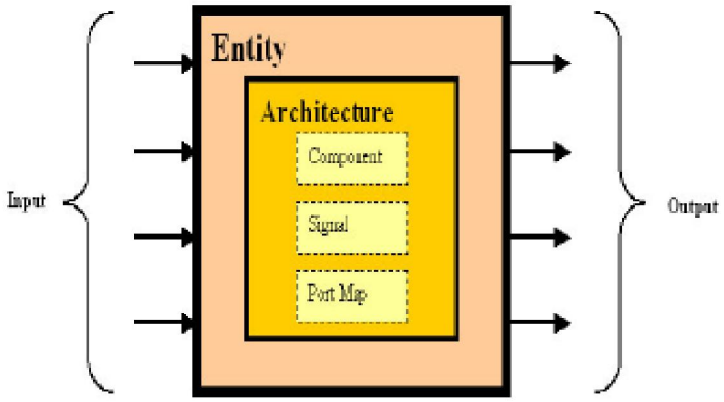

Fig. 1. VHDL Module

2.2. Modeling Digital Systems

The term digital system encompasses a range of systems from low-level components to complete system-on-a-chip and board-level designs. If we are to encompass this range of views of digital systems, we must recognize the complexity with which we are dealing. The most important way of meeting this challenge is to adopt a systematic methodology of design. If we start with a requirements document for the system, we can design an abstract structure that meets the requirements. We can then decompose this structure into a collection of components that interact to perform the same function. Each of these components can in turn be decomposed until we get to a level where we have some ready-made, primitive components that perform a required function. The result of this process is a hierarchically composed system, built from the primitive elements.

The advantage of this methodology is that each subsystem can be designed independently of others. When we use a subsystem, we can think of it as an abstraction rather than having to consider its detailed composition. So at any particular stage in the design process, we only need to pay attention to the small amount of information relevant to the current focus of design. We are saved from being overwhelmed by masses of detail. We use the term model to mean our understanding of a system. The model represents that information which is relevant and abstracts away from irrelevant detail.

There are a number of important motivations for formalizing this idea of a model, including -

- expressing system requirements in a complete and unambiguous way
- documenting the functionality of a system
- testing a design to verify that it performs correctly
- Formally verifying properties of a design
- Synthesizing an implementation in a target technology (e.g., ASIC or FPGA)

2.3. Fundamental Concepts

The unifying factor is that we want to achieve maximum reliability in the design process for minimum cost and design time.

There are two aspects to modeling hardware that any hardware description language facilitates; true abstract behavior and hardware structure. A multitude of language or user defined data types can be used. This will make models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code.

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

2.4. Introduction to Xilinx (Introduction to FPGA Technology)

Here we explain how to download your program to the FPGA and test your implementation.

1. Open up Xilinx project navigator
2. The open window has 4 panes:

   A. A source pane that shows the organization of the source files that make up your design. There are three tabs so you can view the functional modules or HDL libraries for your project or look at various snapshots of the project.

   B. A process pane that lists the various operations you can perform on a given object in the source pane.

   C. A log pane that displays the various messages from the currently running process.

   D. An editor pane where you can enter HDL code. Schematics are entered in a separate window.

One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE) to produce the RTL schematic 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.

3. Generating Programming File

Now that we have synthesized our design and mapped it to the FPGA with the correct pin assignments, we are ready to generate the bit stream that is used to program the actual chip. Highlighted area in Fig. 2 is showing the leddcd object in the Sources pane and double-clicking on the Generate Programming File process will start the job.

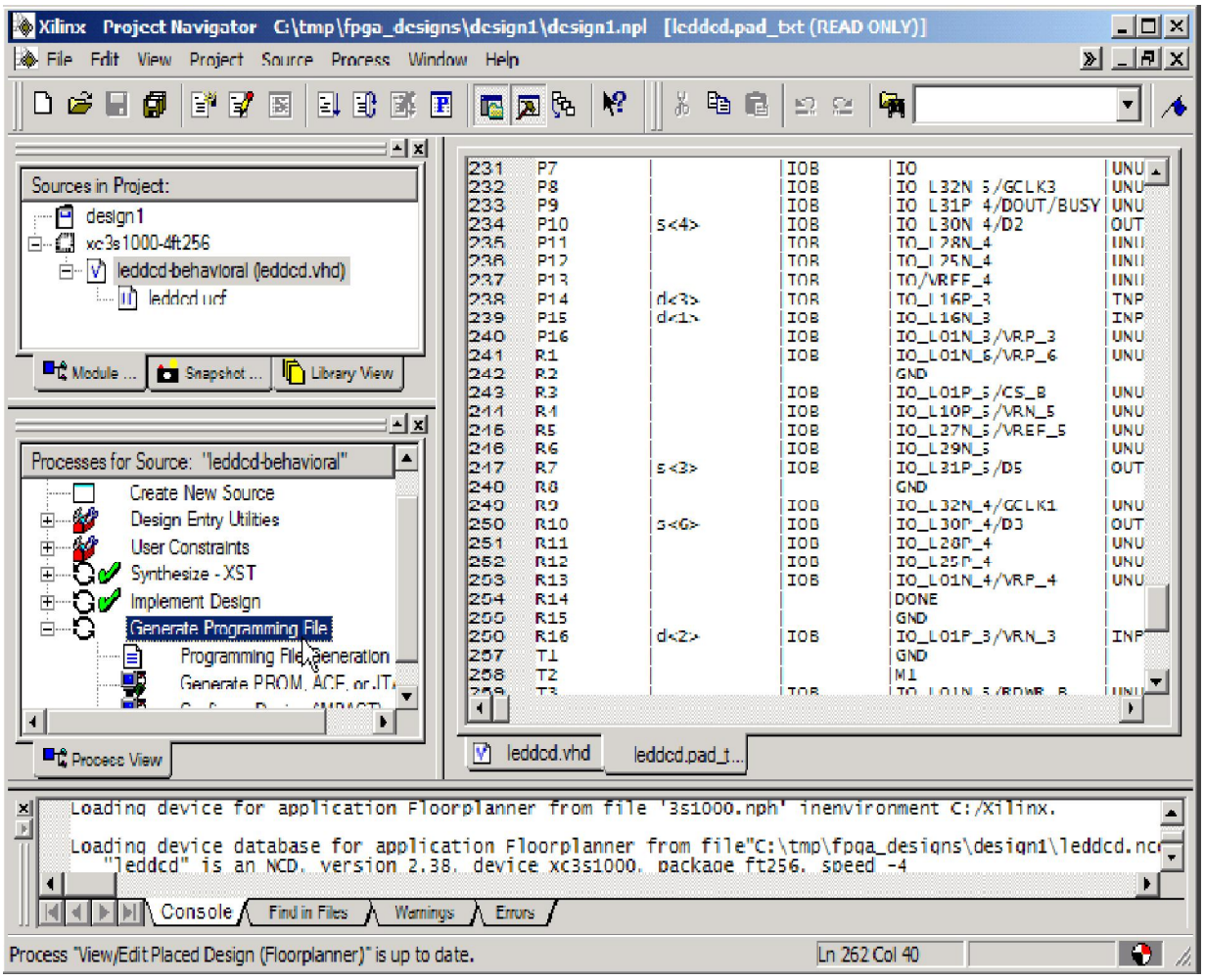

Fig. 2. Generating Programming File

4. Spartan-3E FPGA Starter Kit Board

Fig. 3. shows the actual Spartan-3E FPGA Starter Kit Board which includes XC3S500E device (Package: FG320, Speed: -4) of Spartan-3E family.

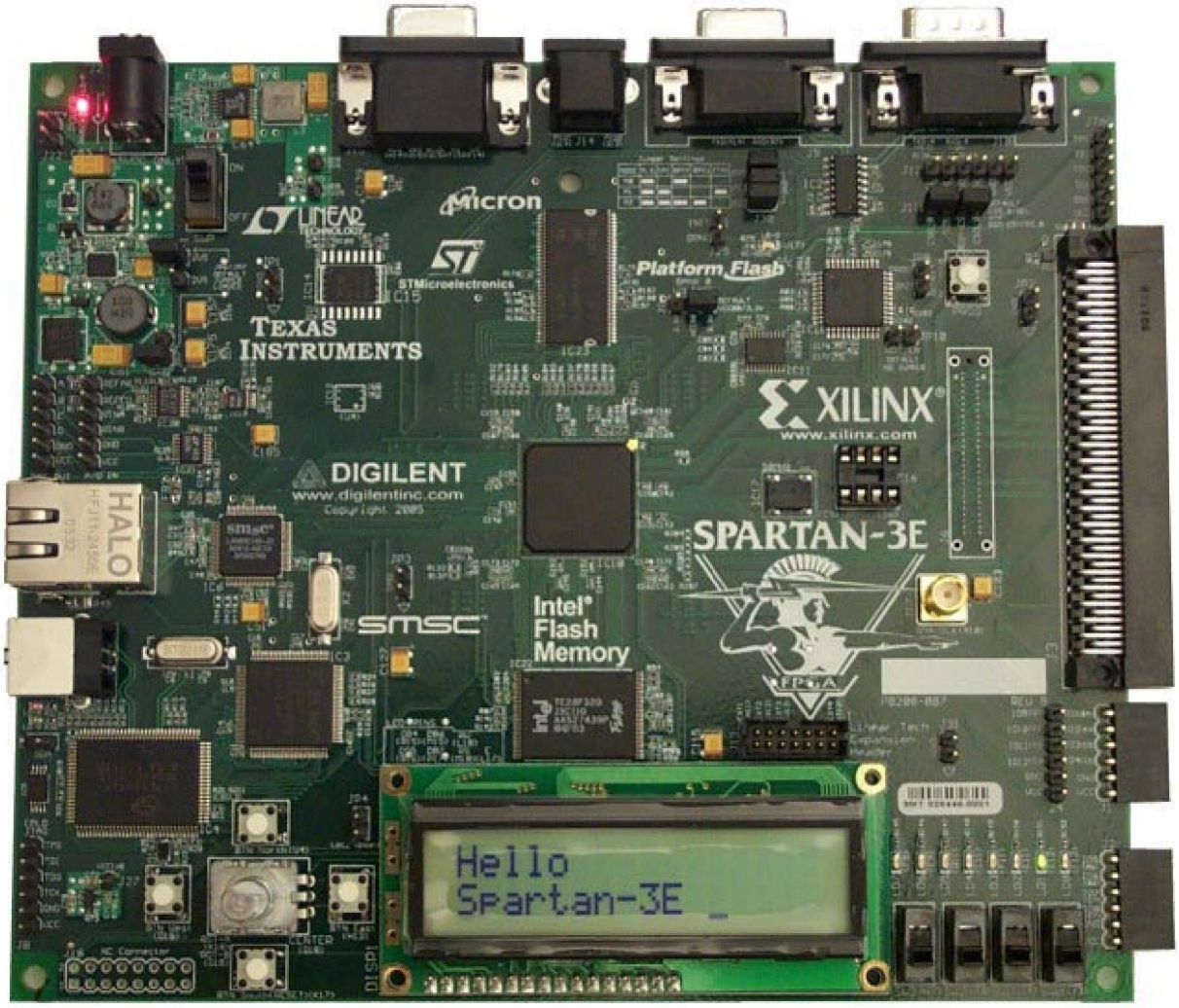

Fig. 3. Spartan-3E FPGA Starter Kit Board

4.1. Spartan-3E FPGA Features and Embedded Processing Functions

The Spartan-3E Starter Kit board highlights the unique features of the Spartan-3E FPGA family and provides a convenient development board for embedded processing applications. The board highlights these features:

- Spartan-3E FPGA specific features
  - ♦ Parallel NOR Flash configuration
  - ♦ Multi-Boot FPGA configuration from Parallel NOR Flash PROM
  - ♦ SPI serial Flash configuration

- Embedded development
  - ♦ MicroBlaze™ 32-bit embedded RISC processor
  - ♦ PicoBlaze™ 8-bit embedded controller
  - ♦ DDR memory interfaces

4.2. Key Components and Features

The key features of the Spartan-3E Starter Kit board are:

- Xilinx XC3S500E Spartan-3E FPGA
  - ♦ Up to 232 user-I/O pins
  - ♦ 320-pin FBGA package
  - ♦ Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner™ CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz
- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
  - ♦ FPGA configuration storage
  - ♦ MicroBlaze code storage/shadowing
- 16 Mbits of SPI serial Flash (STMicro)
  - ♦ FPGA configuration storage
  - ♦ MicroBlaze code shadowing
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors

- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- SMA clock input
- 8-pin DIP socket for auxiliary clock oscillator.

4.3. FPGA – Field Programmable Gate Array

A Field-programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL).

The Field Programmable Gate Arrays (FPGAs) devices, with their reconfigurable logic, practicality, portability, low consumption of energy, high operation speedy and large data-storage capacity, are a great choice for embedded systems project development and prototype.

4.3.1. FPGA – Architecture

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow.

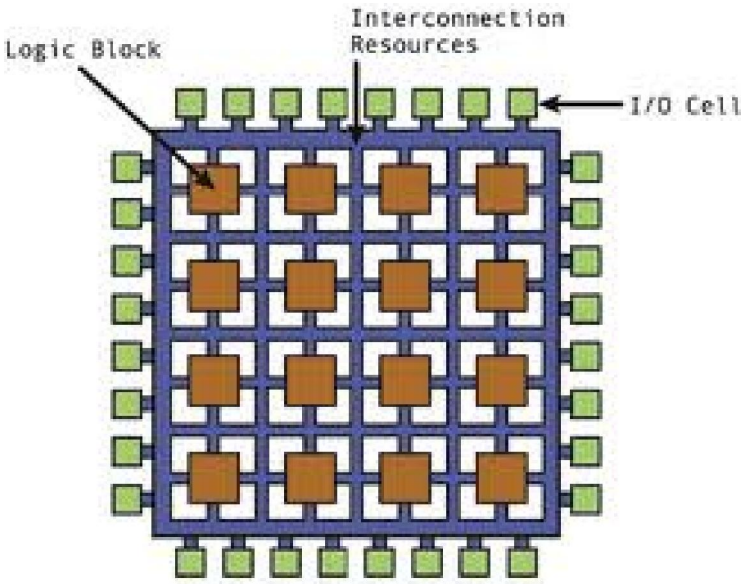

Fig. 4. FPGA Structure

Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip. Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

Xilinx Co-Founders, Ross Freeman and Bernard Vonderschmitt, invented the first commercially viable field programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 boasted a mere 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs).

4.4. VHDL

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. It became IEEE standard 1076 in 1987. It was updated in 1993 and is known today as "IEEE standard 1076 1993.

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 testbench.

There are two aspects to modeling hardware that any hardware description language facilitates; true abstract behavior and hardware structure. A multitude of language or user defined data types can be used. This will make models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code.

One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE) to produce the RTL schematic 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.

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

5. Tools and Environment Used

5.1. Minimum Hardware Requirement:

1. Computer : IBM or compatible
2. Hard disk : 20 GB or higher
3. Processor : PENTIUM– IV 2 GHz or above
4. Ram : 512 Mb and above
5. VDU : VGA
6. Xilinx Spartan–3E FPGA Starter Kit Board

5.2. Minimum Software Requirement:

1. Operating System : Windows XP
2. Development Software : Xilinx ISE 8.2i

6. Project Planning

Once a project is found to be feasible, project planning is undertaken and completed before any development activity starts.

Project planning consists of the following essential activities:-

Estimating some basic attributes of the project-

- **Cost**: How much it will cost to develop the project?
- **Duration**: How long will it take to complete the development?
- **Effort**: How much effort will be required?

The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.

- Scheduling manpower and other resources
- Staff organization and staffing plans
- Risk identification and analysis
- Miscellaneous plans such as quality assurance plan, configuration management plan etc. Developing a system requires planning and coordinating resources within a given time. More importantly, effective project management is required to organize the available resources, schedule the events and establish standards.

The following figure shows the order in which the important planning activities may be undertaken. Size estimation is the first of all activities. It is also the most fundamental parameter based on which all other planning activities are carried out. Other activities such as estimation of effort, cost, resources and project duration are also important components of project planning.

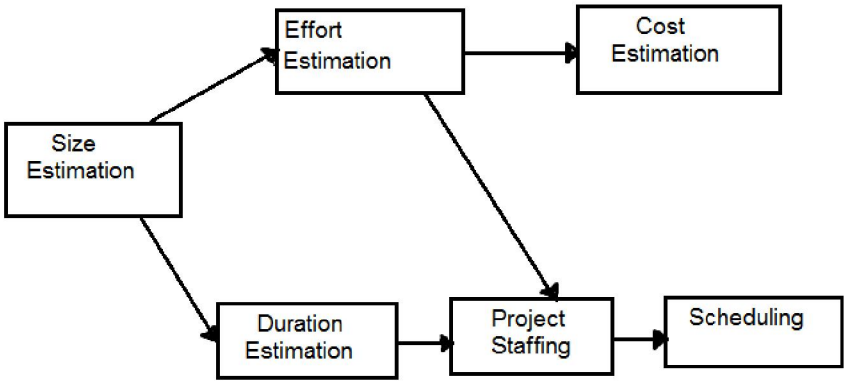

Fig. 5. Project Planning

Project planning involves plotting project activities against a time frame. One of the first steps in planning is development of the road map structure or a network based analysis of the tasks that must be performed to complete the project.

7. 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.

7.1. 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 + C_in

Where A is a 32-bit number, Y is the 32-bit output of multiplexers and C_in is the carry input bit to the parallel adder.

When S_1 S_0= 00:	if C_in=0 then RESULT= A+B i.e. Addition.
	if C_in=1 then RESULT= A+B+1 i.e. Addition with carry.
When S_1 S_0= 01:	if C_in=0 then RESULT= A+$\overline{B}$ i.e. Subtraction with borrow.
	if C_in=1 then RESULT= A+$\overline{B}$ +1 i.e. Subtraction.
When S_1 S_0= 10:	if C_in=0 then RESULT= A-1 i.e. Decrement.
	if C_in=1 then RESULT= A i.e. Transfer.
When S_1 S_0= 10:	if C_in=0 then RESULT= A i.e. Transfer.
	if C_in=1 then RESULT= A i.e. Increment.

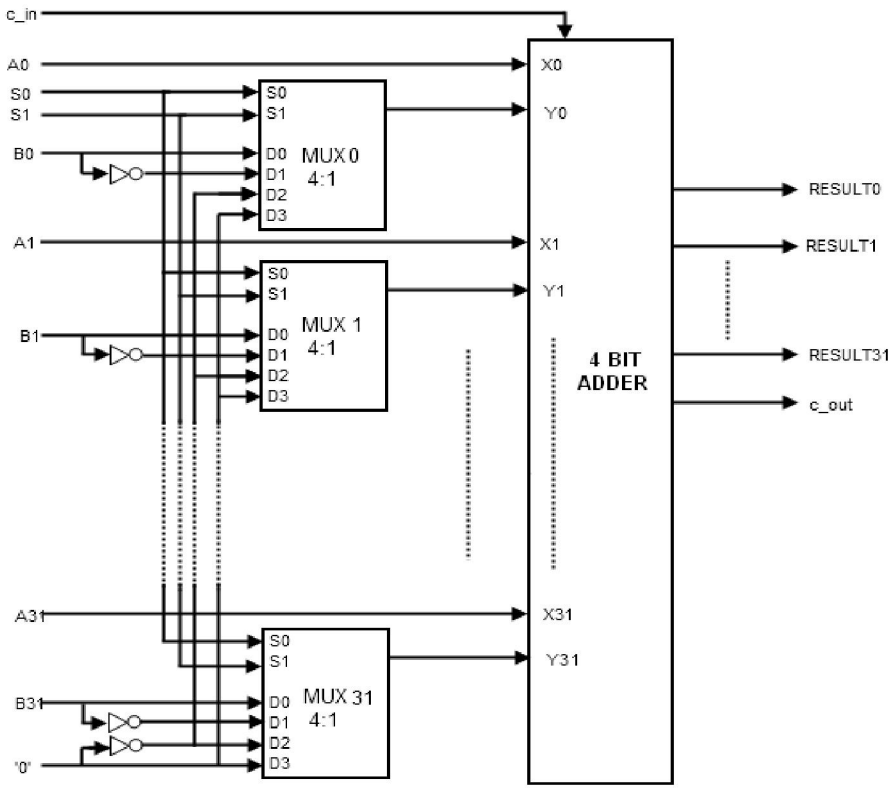

Fig. 6. 32-bit Arithmetic Unit

7.2. 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-impedanced. The common selection lines are applied to all the stages.

When S_1 S_0= 00: RESULT= A·B i.e. AND.
When S_1 S_0= 01: RESULT= A+B i.e. OR.
When S_1 S_0= 10: RESULT= A⊕B i.e. XOR.
When S_1 S_0= 10: RESULT= $\overline{A}$ i.e. NOT

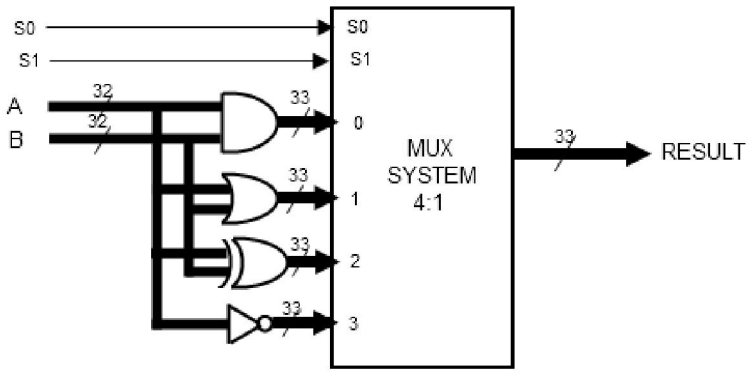

Fig. 7. 32-bit Logic Unit

7.3. 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. 8.

When S_1= 0: RESULT= Shift Right A.

When S_1= 1: RESULT= Shift Left A.

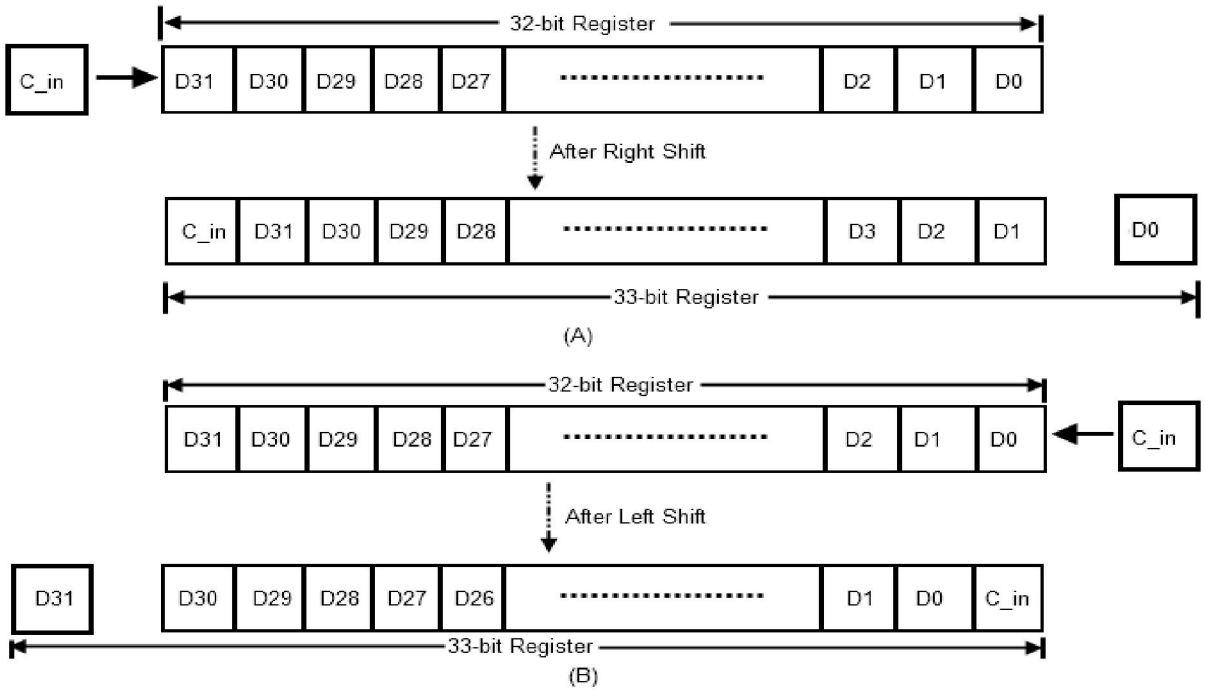

Fig. 7.A. 32-bit Right Shift Operation, 7.B. 32-bit Left Shift Operation.

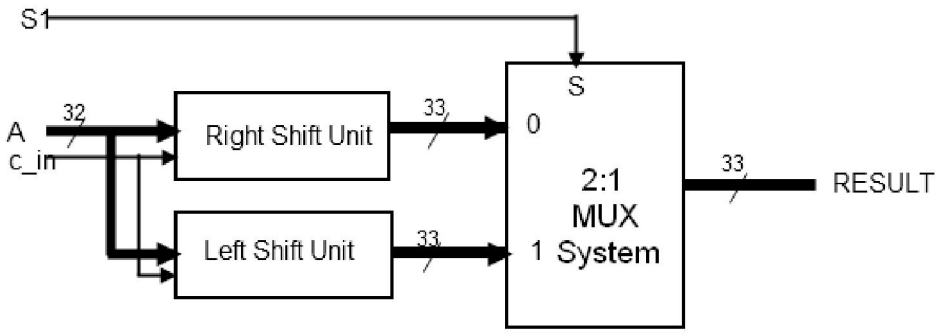

Fig. 8. 32-bit Shifter Unit

7.4. 32-bit Arithmetic and Logical 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.

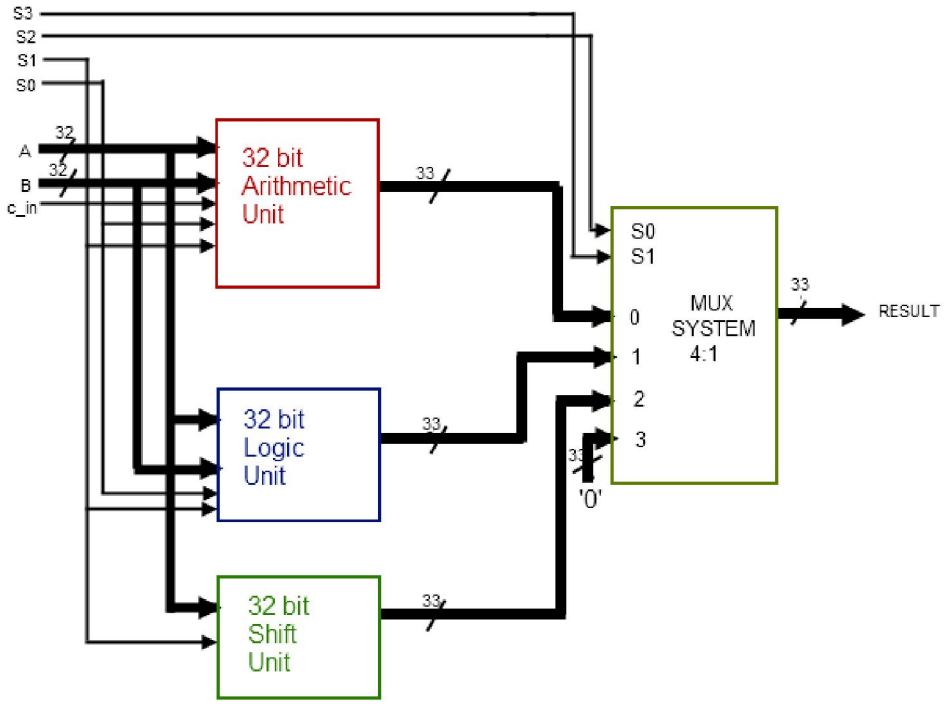

Fig. 9. 32-bit ALU

8. Functions of ALU

S_3	S_2	S_1	S_0	C_{in}	RESULT	Operation
0	0	0	0	0	A + B	Addition
0	0	0	0	1	A + B + 1	Addition with carry
0	0	0	1	0	$A + \overline{B}$	Subtraction with borrow
0	0	0	1	1	$A + \overline{B} + 1$	Subtraction
0	0	1	0	0	A – 1	Decrement
0	0	1	0	1	A	Transfer
0	0	1	1	0	A	Transfer
0	0	1	1	1	A + 1	Increment
0	1	0	0	x	A·B	AND
0	1	0	1	x	A+B	OR
0	1	1	0	x	A⊕B	XOR
0	1	1	1	x	$\overline{A}$	Complement
1	0	0	x	x	LSR A	Shift Right
1	0	1	x	x	LSL A	Shift Left

Table 1. Functions of ALU

9. VHDL Coding

```
----------------------------------------------------------------------------------
-- Company:             ARDENT
-- Engineer:            Arindam, Tanaya, Kausik, Monigingir, Anushka
-- Create Date:         08:02:14 06/24/2011
-- Design Name:         32 bit ALU
-- Module Name:         ALU_32bit - Bhv_alu_32bit
-- Project Name:        32 bit ALU
-- Target Devices:      Spartan3E:xc3s500e-4-fg320
-- Tool versions:
-- Description:
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments: Version 1.0
----------------------------------------------------------------------------------
------------------------------------ CONTROLS ------------------------------------
--  s(3)   s(2)    s(1)  s(0)  c_in      result          Operation
----------------------------------------------------------------------------------
--  0      0       0     0     0         i_a + i_b       Addition
--  0      0       0     0     1         i_a + i_b + 1   Addition with carry
--  0      0       0     1     0         i_a + i_b'      Subtraction with borrow
--  0      0       0     1     1         i_a + i_b' + 1  Subtraction
--  0      0       1     0     0         i_a - 1         Decrement
--  0      0       1     0     1         i_a             Transfer
--  0      0       1     1     0         i_a             Transfer
--  0      0       1     1     1         i_a + 1         Increment
--  0      1       0     0     x         i_a and i_b     AND
--  0      1       0     1     x         i_a or i_b      OR
--  0      1       1     0     x         i_a xor i_b     XOR
--  0      1       1     1     x         i_a'            Complement
--  1      0       0     x     x         LSR i_a         Shift Right
--  1      0       1     x     x         LSL i_a         Shift Left
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
------- 32 BIT ALU BEGINS [i_a(31-0), i_b(31-0), c_in, s(3-0), result(32-0)]----
----------------------------------------------------------------------------------
------------------------ SELECTION MUX 4 TO 1 (1 BIT I/O) ----------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux1_4to1 is
    Port ( a,b,c,d:in STD_LOGIC;
              s: in STD_LOGIC_VECTOR(1 downto 0);
              y: out STD_LOGIC);
end mux1_4to1;
```

```
architecture bhv_mux1_4to1 of mux1_4to1 is

begin
    process(s,a,b,c,d)
    begin
        case s is
             when "00" => y<=a;
             when "01" => y<=b;
             when "10" => y<=c;
             when "11" => y<=d;
             when others => y<='Z';
        end case;
    end process;
end bhv_mux1_4to1;

----------------------------------------------------------------------------------
---------------------- SELECTION MUX 4 TO 1 (33 BIT I/O) ----------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux33_4to1 is
    Port ( a,b,c,d:in STD_LOGIC_VECTOR(32 downto 0);
              s: in STD_LOGIC_VECTOR(1 downto 0);
              y: out STD_LOGIC_VECTOR(32 downto 0));
end mux33_4to1;

architecture bhv_mux33_4to1 of mux33_4to1 is

begin
    process(s,a,b,c,d)
    begin
        case s is
             when "00" => y<=a;
             when "01" => y<=b;
             when "10" => y<=c;
             when "11" => y<=d;
             when others => y<="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        end case;
    end process;
end bhv_mux33_4to1;

----------------------------------------------------------------------------------
---------------------- SELECTION MUX 2 TO 1 (33 BIT I/O) ----------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity mux33_2to1 is
    Port ( a:in STD_LOGIC_VECTOR (32 downto 0);
             b:in STD_LOGIC_VECTOR (32 downto 0);
             s: in STD_LOGIC;
             y: out STD_LOGIC_VECTOR (32 downto 0));
end mux33_2to1;

architecture bhv_mux33_2to1 of mux33_2to1 is

begin
    process(s,a,b)
    begin
        case s is
             when '0' => y<=a;
             when '1' => y<=b;
             when others => y<="ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        end case;
    end process;
end bhv_mux33_2to1;

----------------------------------------------------------------------------------
---------------------------------- FULL ADDER ------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity full_adder is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end full_adder;

architecture bhv_full of full_adder is

begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or ((a xor b) and cin);
end bhv_full;

----------------------------------------------------------------------------------
--------------------------------- 4BIT ADDER -------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity adder_4bit is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           b : in  STD_LOGIC_VECTOR (3 downto 0);
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC_VECTOR (3 downto 0);
           carry : out  STD_LOGIC);
end adder_4bit;

architecture bhv_4bit of adder_4bit is

component full_adder is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end component;

signal s:STD_LOGIC_VECTOR (2 downto 0);

begin
    p1:full_adder
        port map(a(0),b(0),cin,sum(0),s(0));
    p2:full_adder
        port map(a(1),b(1),s(0),sum(1),s(1));
    p3:full_adder
        port map(a(2),b(2),s(1),sum(2),s(2));
    p4:full_adder
        port map(a(3),b(3),s(2),sum(3),carry);
end bhv_4bit;

--------------------------------------------------------------------------------
---------------------------------- 32BIT ADDER ---------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Adder_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end Adder_32bit;

architecture bhv_adder_32 of Adder_32bit is

component adder_4bit is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           b : in  STD_LOGIC_VECTOR (3 downto 0);
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC_VECTOR (3 downto 0);
```

```
           carry : out  STD_LOGIC);
end component;

signal s:STD_LOGIC_VECTOR (6 downto 0);

begin
    s1:adder_4bit
        port map(i_a(3 downto 0),i_b(3 downto 0),c_in,result(3 downto 0),s(0));
    s2:adder_4bit
        port map(i_a(7 downto 4),i_b(7 downto 4),s(0),result(7 downto 4),s(1));
    s3:adder_4bit
        port map(i_a(11 downto 8),i_b(11 downto 8),s(1),result(11 downto
8),s(2));
    s4:adder_4bit
        port map(i_a(15 downto 12),i_b(15 downto 12),s(2),result(15 downto
12),s(3));
    s5:adder_4bit
        port map(i_a(19 downto 16),i_b(19 downto 16),s(3),result(19 downto
16),s(4));
    s6:adder_4bit
        port map(i_a(23 downto 20),i_b(23 downto 20),s(4),result(23 downto
20),s(5));
    s7:adder_4bit
        port map(i_a(27 downto 24),i_b(27 downto 24),s(5),result(27 downto
24),s(6));
    s8:adder_4bit
        port map(i_a(31 downto 28),i_b(31 downto 28),s(6),result(31 downto
28),result(32));
end bhv_adder_32;

--------------------------------------------------------------------------------
---------------------------------- 32BIT AND -----------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity AND_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end AND_32bit;

architecture Bhv_and_32bit of AND_32bit is

begin
    result(31 downto 0)<=i_a(31 downto 0) and i_b(31 downto 0);
    result(32)<='Z';
end Bhv_and_32bit;

--------------------------------------------------------------------------------
```

```
---------------------------------- 32BIT OR ---------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OR_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end OR_32bit;

architecture Bhv_or_32bit of OR_32bit is

begin
    result(31 downto 0)<=i_a(31 downto 0) or i_b(31 downto 0);
    result(32)<='Z';
end Bhv_or_32bit;

-------------------------------------------------------------------------------
---------------------------------- 32BIT XOR --------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity XOR_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end XOR_32bit;

architecture Bhv_xor_32bit of XOR_32bit is

begin
    result(31 downto 0)<=i_a(31 downto 0) xor i_b(31 downto 0);
    result(32)<='Z';
end Bhv_xor_32bit;

-------------------------------------------------------------------------------
---------------------------------- 32BIT NOT --------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity NOT_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end NOT_32bit;
```

```
architecture Bhv_not_32bit of NOT_32bit is

begin
    result(31 downto 0)<=not i_a(31 downto 0);
    result(32)<='Z';
end Bhv_not_32bit;

----------------------------------------------------------------------------------
---------------------------------- RIGHT SHIFT -----------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity R_Shift_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end R_Shift_32bit;

architecture bhv_rshift_32 of R_Shift_32bit is

begin
    result(30 downto 0)<=i_a(31 downto 1);
    result(31)<=c_in;
    result(32)<=i_a(0);
end bhv_rshift_32;

----------------------------------------------------------------------------------
---------------------------------- LEFT SHIFT ------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity L_Shift_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end L_Shift_32bit;

architecture bhv_lshift_32 of L_Shift_32bit is

begin
    result(31 downto 1)<=i_a(30 downto 0);
    result(0)<=c_in;
    result(32)<=i_a(31);
end bhv_lshift_32;

----------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
---- ARITHMETIC UNIT BEGINS [i_a(31-0), i_b(31-0), c_in, s(1-0), result(32-0)]--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Arithmetic is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           s : in  STD_LOGIC_VECTOR (1 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end Arithmetic;

architecture Bhv_Arithmetic of Arithmetic is

component mux1_4to1 is
    Port ( a,b,c,d:in STD_LOGIC;
             s: in STD_LOGIC_VECTOR(1 downto 0);
             y: out STD_LOGIC);
end component;

component Adder_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

signal sg:STD_LOGIC_VECTOR(31 downto 0);

begin
    p0:mux1_4to1
        Port map(i_b(0),not i_b(0),'1','0',s,sg(0));
    p1:mux1_4to1
        Port map(i_b(1),not i_b(1),'1','0',s,sg(1));
    p2:mux1_4to1
        Port map(i_b(2),not i_b(2),'1','0',s,sg(2));
    p3:mux1_4to1
        Port map(i_b(3),not i_b(3),'1','0',s,sg(3));
    p4:mux1_4to1
        Port map(i_b(4),not i_b(4),'1','0',s,sg(4));
    p5:mux1_4to1
        Port map(i_b(5),not i_b(5),'1','0',s,sg(5));
    p6:mux1_4to1
        Port map(i_b(6),not i_b(6),'1','0',s,sg(6));
    p7:mux1_4to1
        Port map(i_b(7),not i_b(7),'1','0',s,sg(7));
    p8:mux1_4to1
        Port map(i_b(8),not i_b(8),'1','0',s,sg(8));
    p9:mux1_4to1
```

```
        Port map(i_b(9),not i_b(9),'1','0',s,sg(9));
    p10:mux1_4to1
        Port map(i_b(10),not i_b(10),'1','0',s,sg(10));
    p11:mux1_4to1
        Port map(i_b(11),not i_b(11),'1','0',s,sg(11));
    p12:mux1_4to1
        Port map(i_b(12),not i_b(12),'1','0',s,sg(12));
    p13:mux1_4to1
        Port map(i_b(13),not i_b(13),'1','0',s,sg(13));
    p14:mux1_4to1
        Port map(i_b(14),not i_b(14),'1','0',s,sg(14));
    p15:mux1_4to1
        Port map(i_b(15),not i_b(15),'1','0',s,sg(15));
    p16:mux1_4to1
        Port map(i_b(16),not i_b(16),'1','0',s,sg(16));
    p17:mux1_4to1
        Port map(i_b(17),not i_b(17),'1','0',s,sg(17));
    p18:mux1_4to1
        Port map(i_b(18),not i_b(18),'1','0',s,sg(18));
    p19:mux1_4to1
        Port map(i_b(19),not i_b(19),'1','0',s,sg(19));
    p20:mux1_4to1
        Port map(i_b(20),not i_b(20),'1','0',s,sg(20));
    p21:mux1_4to1
        Port map(i_b(21),not i_b(21),'1','0',s,sg(21));
    p22:mux1_4to1
        Port map(i_b(22),not i_b(22),'1','0',s,sg(22));
    p23:mux1_4to1
        Port map(i_b(23),not i_b(23),'1','0',s,sg(23));
    p24:mux1_4to1
        Port map(i_b(24),not i_b(24),'1','0',s,sg(24));
    p25:mux1_4to1
        Port map(i_b(25),not i_b(25),'1','0',s,sg(25));
    p26:mux1_4to1
        Port map(i_b(26),not i_b(26),'1','0',s,sg(26));
    p27:mux1_4to1
        Port map(i_b(27),not i_b(27),'1','0',s,sg(27));
    p28:mux1_4to1
        Port map(i_b(28),not i_b(28),'1','0',s,sg(28));
    p29:mux1_4to1
        Port map(i_b(29),not i_b(29),'1','0',s,sg(29));
    p30:mux1_4to1
        Port map(i_b(30),not i_b(30),'1','0',s,sg(30));
    p31:mux1_4to1
        Port map(i_b(31),not i_b(31),'1','0',s,sg(31));

    p32:Adder_32bit
        Port map(i_a,sg,c_in,result);

end Bhv_Arithmetic;

------------------------------ ARITHMETIC UNIT ENDS ----------------------------
--------------------------------------------------------------------------------
```

```
---------- LOGIC UNIT BEGINS [i_a(31-0), i_b(31-0), s(1-0), result(32-0)]-------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Logic is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           s : in  STD_LOGIC_VECTOR (1 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end Logic;

architecture Bhv_logic of Logic is

component AND_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component OR_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component XOR_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component NOT_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component mux33_4to1 is
    Port ( a,b,c,d:in STD_LOGIC_VECTOR(32 downto 0);
             s: in STD_LOGIC_VECTOR(1 downto 0);
             y: out STD_LOGIC_VECTOR(32 downto 0));
end component;

signal sg_and: STD_LOGIC_VECTOR(32 downto 0);
signal sg_or: STD_LOGIC_VECTOR(32 downto 0);
signal sg_xor: STD_LOGIC_VECTOR(32 downto 0);
signal sg_not: STD_LOGIC_VECTOR(32 downto 0);

begin
    p0:AND_32bit
        Port map(i_a,i_b,sg_and);
```

```
    p1:OR_32bit
        Port map(i_a,i_b,sg_or);
    p2:XOR_32bit
        Port map(i_a,i_b,sg_xor);
    p3:NOT_32bit
        Port map(i_a,sg_not);

    p4:mux33_4to1
        Port map(sg_and,sg_or,sg_xor,sg_not,s,result);

end Bhv_logic;

--------------------------------- LOGIC UNIT ENDS -----------------------------
-------------------------------------------------------------------------------
------------ SHIFT UNIT BEGINS [i_a(31-0), c_in, s(1-0), result(32-0)]---------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Shift is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           s : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end Shift;

architecture Bhv_shift of Shift is

component R_Shift_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component L_Shift_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component mux33_2to1 is
    Port ( a:in STD_LOGIC_VECTOR (32 downto 0);
             b:in STD_LOGIC_VECTOR (32 downto 0);
             s: in STD_LOGIC;
             y: out STD_LOGIC_VECTOR (32 downto 0));
end component;

signal sg_rshift: STD_LOGIC_VECTOR (32 downto 0);
signal sg_lshift: STD_LOGIC_VECTOR (32 downto 0);

begin
```

```
    p0:R_Shift_32bit
        Port map(i_a,c_in,sg_rshift);
    p1:L_Shift_32bit
        Port map(i_a,c_in,sg_lshift);

    p2:mux33_2to1
        Port map(sg_rshift,sg_lshift,s,result);

end Bhv_shift;

-------------------------------- SHIFT UNIT ENDS -----------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
---------------------------------- 32 BIT ALU --------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU_32bit is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           s : in  STD_LOGIC_VECTOR (3 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end ALU_32bit;

architecture Bhv_alu_32bit of ALU_32bit is

component Arithmetic is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           s : in  STD_LOGIC_VECTOR (1 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component Logic is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           i_b : in  STD_LOGIC_VECTOR (31 downto 0);
           s : in  STD_LOGIC_VECTOR (1 downto 0);
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component Shift is
    Port ( i_a : in  STD_LOGIC_VECTOR (31 downto 0);
           c_in : in  STD_LOGIC;
           s : in  STD_LOGIC;
           result : out  STD_LOGIC_VECTOR (32 downto 0));
end component;

component mux33_4to1 is
```

```
    Port ( a,b,c,d:in STD_LOGIC_VECTOR(32 downto 0);
             s: in STD_LOGIC_VECTOR(1 downto 0);
             y: out STD_LOGIC_VECTOR(32 downto 0));
end component;

signal sg_arith :STD_LOGIC_VECTOR(32 downto 0);
signal sg_logic :STD_LOGIC_VECTOR(32 downto 0);
signal sg_shift :STD_LOGIC_VECTOR(32 downto 0);

begin
    p0:Arithmetic
        Port map(i_a,i_b,c_in,s(1 downto 0),sg_arith);
    p1:Logic
        Port map(i_a,i_b,s(1 downto 0),sg_logic);
    p2:Shift
        Port map(i_a,c_in,s(1),sg_shift);

    p3:mux33_4to1
        Port
map(sg_arith,sg_logic,sg_shift,"ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ",s(3 downto
2),result);

end Bhv_alu_32bit;
-------------------------------- 32 BIT ALU ENDS ------------------------------
```

10. Waveforms of Different Units of ALU

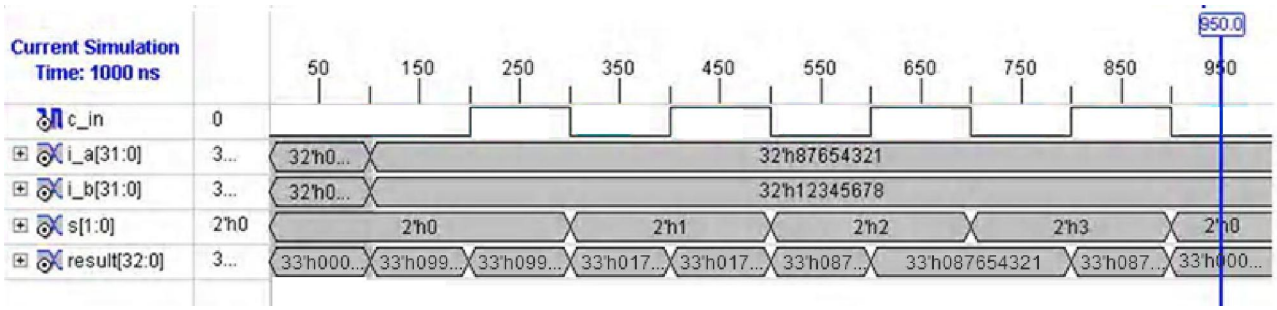

Fig. 10. Waveforms of Arithmetic Unit

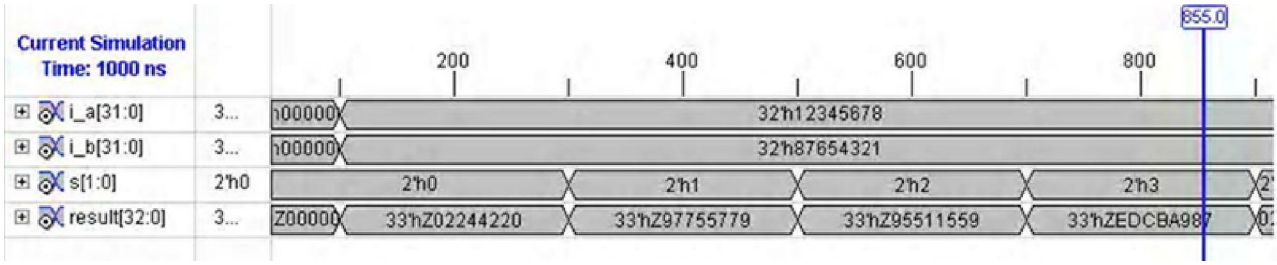

Fig. 11. Waveforms of Logic Unit

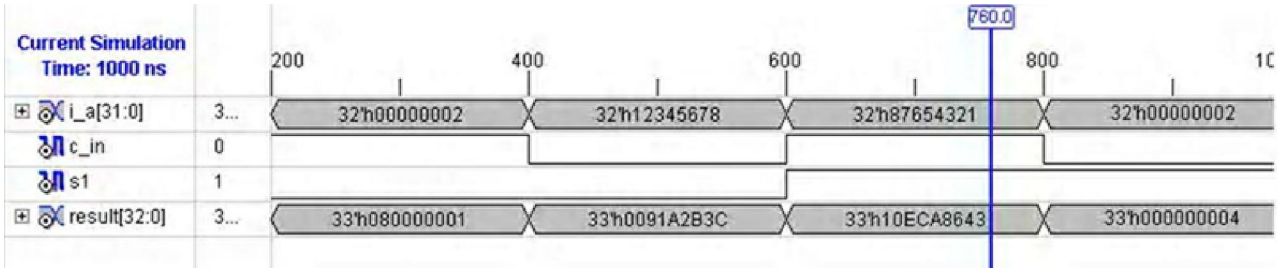

Fig. 12. Waveforms of Shifter Unit

11. Limitations of the Project

11.1. Limitations of VHDL

Although VHDL currently enjoys a healthy market share, there are several limitations and drawbacks in the language:

- VHDL syntax is verbose, extremely cumbersome, and requires several lines of code to describe even simple logic elements (e.g., a register typically requires four to ten lines of code).
- Hardware needs to be described at a very low level of abstraction (i.e., RTL). The programmer is responsible for specifying the logic that goes between each register stage, which can become a significant programming challenge for large irregular designs with thousands of registers and unique logic between register stages.
- As technology and FPGA architectures evolve, the optimal amount of pipelining required to meet the desired cycle time changes. Because RTL is written for a specific number of registers in the logic path, it needs to be rewritten when the number of register stages changes. In other words, the amount of logic between register stages must be modified accordingly.
- Low-level descriptions also make it hard for synthesis tools to optimize and schedule logic. Programmer bias disallows optimizations that might have otherwise been possible in a more flexible description.
- Hardware described in VHDL suffers from the additional drawback of significantly long verification times. It is known that equivalent simulation-specific, cycle-accurate models written in C, C++, Java, or other higher-level language can be simulated 10 to 100 times faster than in VHDL. Verification is a significant portion of the design cycle, and there is demand to contain the time spent on it.

11.2. Limitations of FPGA

- Need for internal memory
- Faster FPGAs : limited by reconfiguration switches
- Larger FPGAs : reduce the need for interconnection chips like the Aptix FPICs, further reducing the delays
- Higher level building blocks on the FPGA: build faster special purpose ALUs.

12. Conclusion

In our project **"**Design and Implementation of a 32-bit ALU on Xilinx FPGA using VHDL**"** we have designed and implemented a 32 bit ALU. Arithmetic Logic Unit is the part of a computer that performs all arithmetic computations, such as addition and subtraction, increment, decrement, shifting and all sorts of basic logical operations. The ALU is one component of the CPU (Central Processing Unit).

Here, using VHDL we have designed a 32 bit ALU which can perform the various arithmetic operations of Addition, Subtraction, Increment, Decrement, Transfer, logical operations such as AND, OR, XOR, NOT and also the shift operation.

All the above mentioned operations are then verified to see whether they match theoretically or not. The above given waveforms show that they match completely thereby verifying our results.

RERERENCES

[1]. T. K. Ghosh and A. J. Pal, *Computer Organization and Architecture,* Tata McGraw-Hill Publishing Company Limited, New Delhi, 2009.

[2]. Douglas L. Perry, VHDL Programming by Example, 4th ed., Tata McGraw-Hill Publishing Company Limited, New Delhi, 2002.

[3]. Xilinx, *Spartan-3E FPGA Family: Data Sheet,* DS312 (v3.8) August 26, 2009.

[4]. Xilinx*, Spartan-3E Starter Kit Board User Guide,* UG230 (v1.0) March 9, 2006.

[5]. *VHDL Tutorial,* Ardent Computech, PVT. LTD., 2011.

[6]. http://www.google.com.

[7]. http://www.wikipedia.org.