A PROJECT SYNOPSIS ON

DESIGN AND IMPLEMENTATION OF A 8-BIT ALU ON XILINX FPGA USING VHDL

SUBMITTED BY
Dipanwita De
Diptanu Majumder
Jayashree Barman
Rajkumar Manna
Sanu Samanta
Sayani Dutta

Guided By
Mr. Amrik Basak
(Asst. Professor)
Dept. of ECE, SMIT

SAROJ MOHAN INSTITUTE OF TECHNOLOGY
(DEGREE DIVISION)
GUPTIPARA, HOOGHLY

OBJECTIVE

The main objective of our project is to design a 8 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.

INTRODUCTION

This paper deals with the design methodology of a FPGA Arithmetic Processor using VHDL to enhance the description, simulation and hardware realization. The design and implementation of FPGA based Arithmetic Logic Unit is of core significance in digital technologies as it is being an integral part of all microprocessors. As the name suggests, this is a system which is capable of performing not only arithmetic operations but also computes logic functions and provides the output through gating circuitry. All the modules described in the design are coded using VHDL which is a very useful tool with its degree of concurrency to cope with the parallelism of digital hardware. The top level module connects all the stages into a higher level. Once identifying the individual approaches for input, output and other modules, the VHDL descriptions are run through a VHDL simulator, followed by the timing diagrams for the verification, working and performance of the above design along with the hardware implementation that shows the appropriateness of the design.

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 two parts namely, Entity and Architecture as Fig. 1.

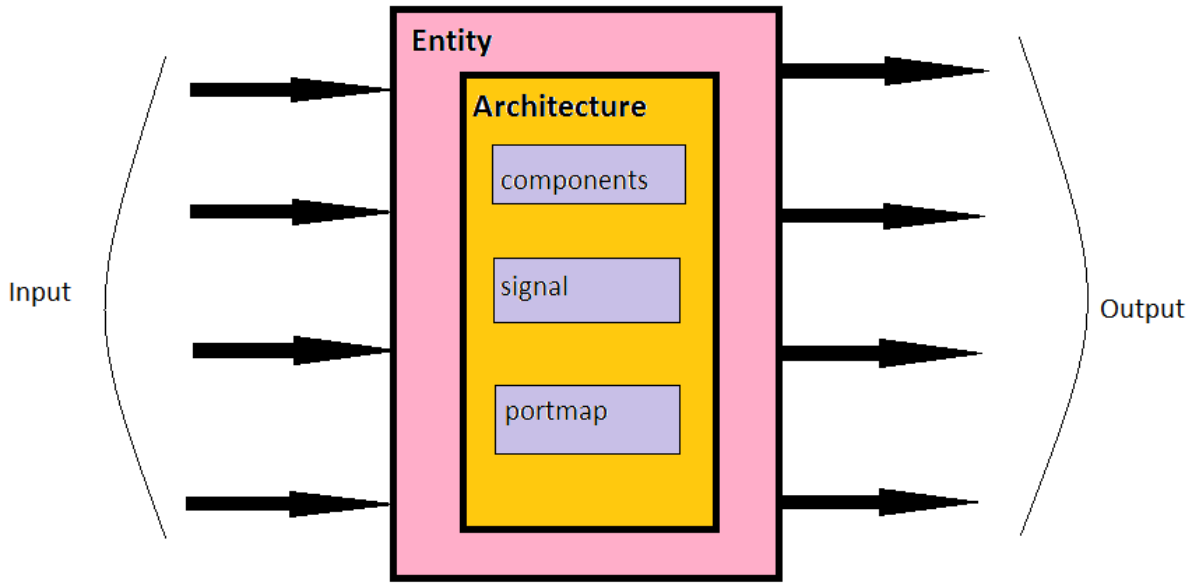


Fig. 1. VHDL Module

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.

Programming in VHDL

- Entity Declaration:

It defines the names, input output signals and modes of a hardware module.
An entity declaration should starts with "entity" and ends with" end" keywords. Ports are interfaced 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.

- Syntax:

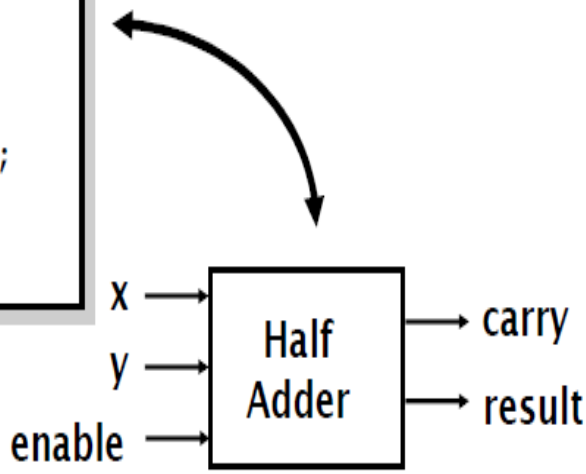
```
entity entity_name is
Port declaration;
end entity_name;
```

- Example:

```
ENTITY half_adder IS

   PORT( x, y, enable: IN bit;
         carry, result: OUT bit);

END half_adder;
```



• Architecture Declaration

It describes the internal description of design or it tells what is there inside design. Each entity has at least 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.

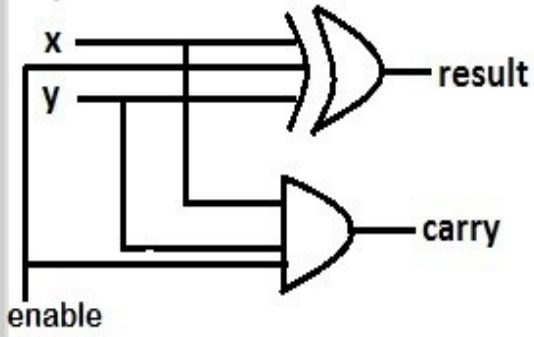
- Syntax:

```
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.

- Example:

```
Architecture behave of half_adder is
BEGIN
    PROCESS (enable, x, y)
    BEGIN
      IF (enable ='1') THEN
         result <= x XOR y ;
         carry <= x AND y ;
      ELSE
         carry ='0' ;
         result ='0' ;
      END IF ;
      END PROCESS ;
END behave ;
```



Spartan-3E FPGA Starter Kit Board

Figure shows the actual Spartan-3E FPGA Starter Kit Board which includes XC3S100E device of Spartan-3E family.

- Main components:

• 100,000-gate Xilinx Spartan-3E XC3S100E FPGA
• A 40-pin expansion connection port to gain access to the Spartan-3E FPGA
• Four 6-pin expansion connector ports to extend and enhance the Spartan-3E Sample Pack
• 7 Light Emitting Diodes (LEDs)
• Power Regulators

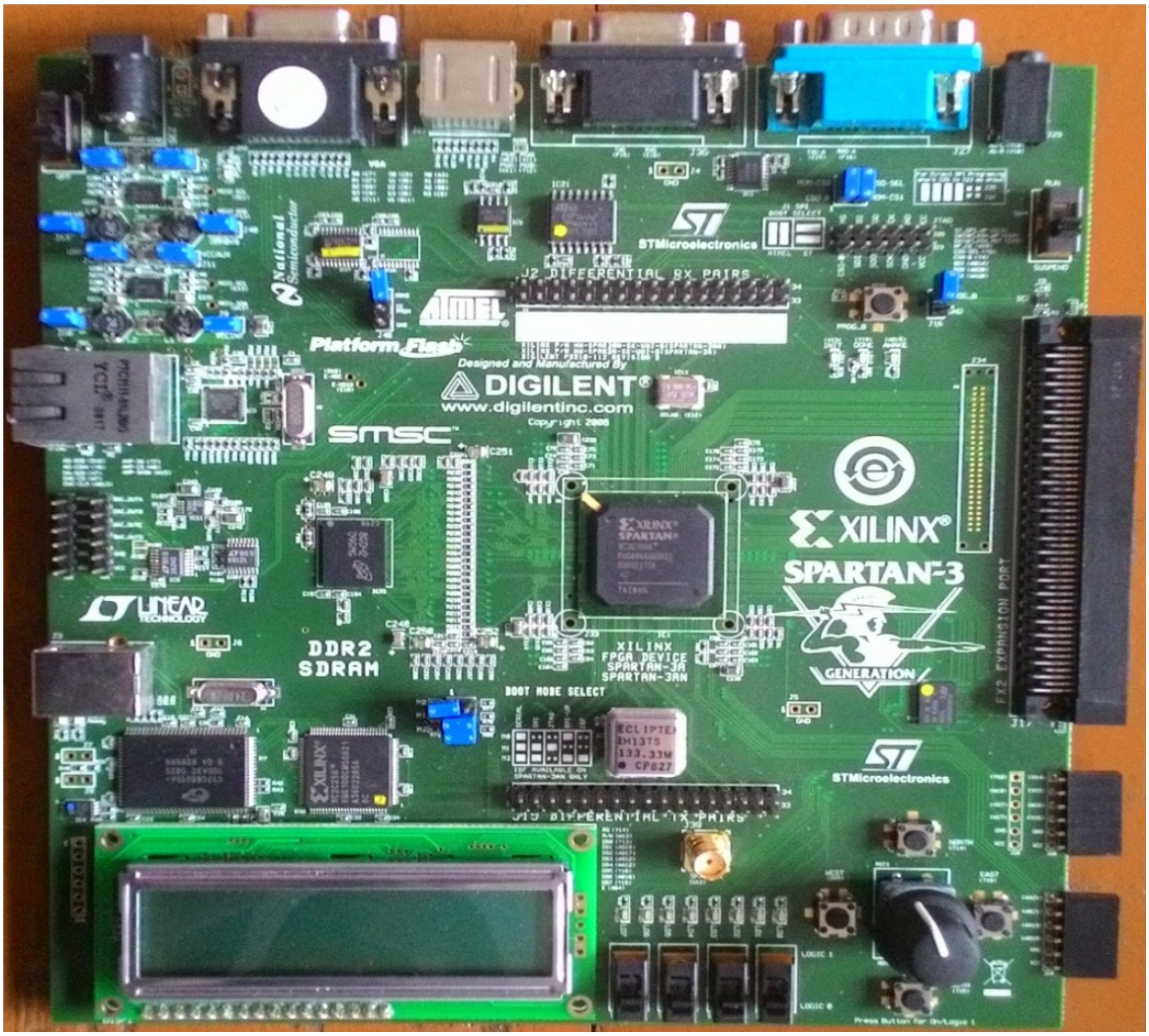


Fig.3:- Spartan-3E FPGA Starter Kit Board

Field Programmable Gate Array (FPGA)

FPGAs combine the programmability of processors with the performance of custom hardware. As they become more common in critical embedded systems, new techniques are necessary to manage security in FPGA designs. This article discusses FPGA security problems and current research on reconfigurable devices and security, and presents security primitives and a component architecture for building highly secure systems on FPGAs. Because FPGAs can provide a useful balance between performance, rapid time to market, and flexibility, they have become the primary source of computation in many critical embedded systems.1,2 The aerospace industry, for example, relies on FPGAs to control everything from the Joint Strike Fighter to the Mars Rover. Face recognition systems, wireless networks, intrusion detection systems, and supercomputers, all of which are employed in large security applications, also use FPGAs. In fact, in 2005 alone, an estimated 80,000 different commercial FPGA design projects began. Because major IC manufacturers outsource most of their operations,4 IP theft from a foundry is a serious concern. FPGAs provide a viable solution to this problem because the sensitive IP is not loaded onto the device until after it has been manufactured and delivered, making it

harder for adversaries to target a specific application or user. Furthermore, modern FPGAs use bit stream encryption and other methods to protect IP once it is loaded onto the FPGA or external memory. However, techniques beyond bit stream encryption are necessary to ensure FPGA design security. To save time and money, FPGA systems are typically cobbled together from a collection of existing computational cores, often obtained from third parties.

FPGAs are programmable semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected through programmable interconnects. As opposed to Application Specific Integrated Circuits (ASICs), where the device is custom built for the particular design, FPGAs can be programmed to the desired application or functionality requirements. Although One- Time Programmable (OTP) FPGAs are available, the dominant type are SRAM based which can be reprogrammed as the design evolves. FPGAs allow designers to change their designs very late in the design cycle– even after the end product has been manufactured and deployed in the field. In addition, Xilinx FPGAs allow for field upgrades to be completed remotely, eliminating the costs associated with re-designing or manually updating electronic systems.

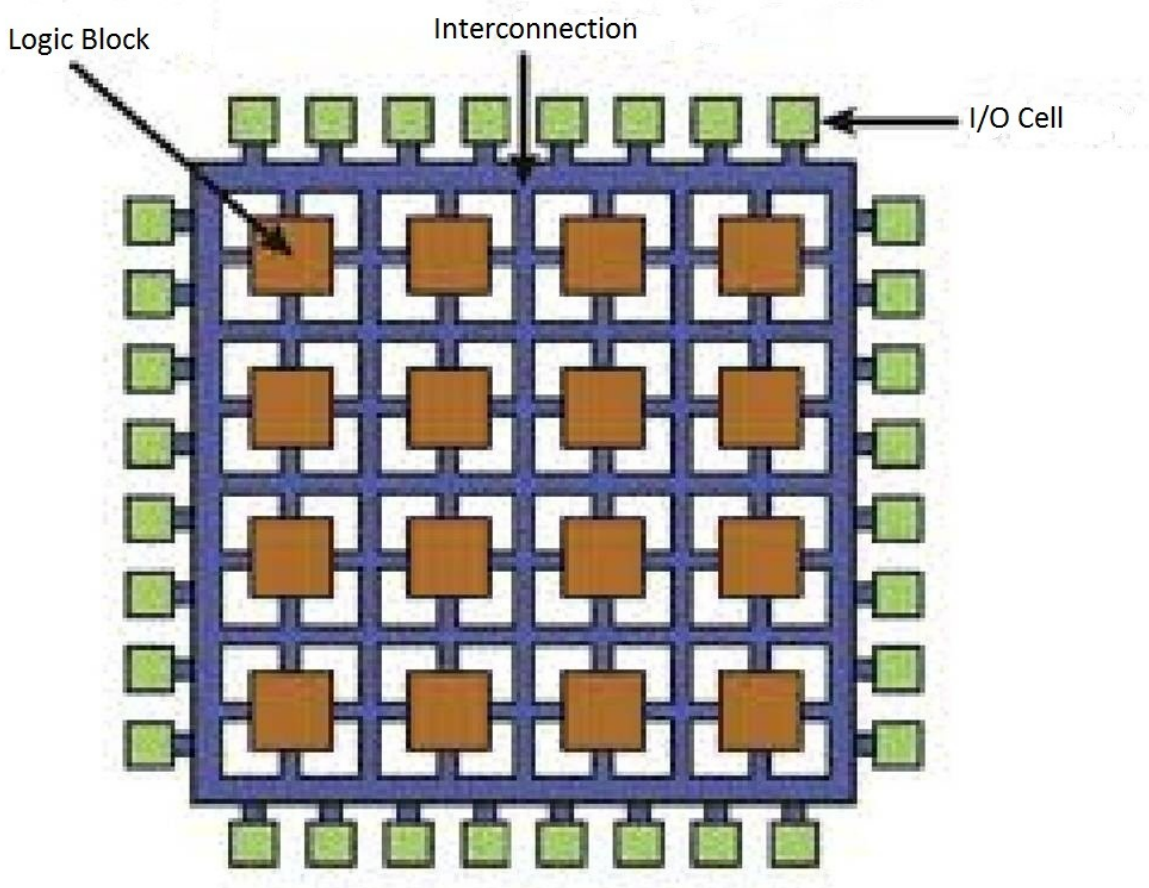


Fig.4. FPGA Block Structure

Common FPGA Features

FPGAs have evolved far beyond the basic capabilities present in their predecessors,

and incorporate hard (ASIC type) blocks of commonly used functionality such as RAM, clock management, and DSP. The following are the basic components in an FPGA:.

- Configurable Logic Blocks (CLBs)

The CLB is the basic logic unit in a FPGA. Exact numbers and features vary from device to device, but every CLB consists of a configurable switch matrix with 4 or 6 inputs, some selection circuitry (MUX, etc), and flip-flops. The switch matrix is highly flexible and can be configured to handle combinatorial logic, shift registers or RAM. More architectural details can be found in the applicable device‟s data sheet.

- Interconnect

While the CLB provides the logic capability, flexible interconnect routing routes the signals between CLBs and to and from I/Os. Routing comes in several flavors, from that designed to interconnect between CLBs to fast horizontal and vertical long lines spanning the device to global low-skew routing for Clocking and other global signals. The design software makes the interconnect routing task hidden to the user unless specified otherwise, thus significantly reducing design complexity.

- Select IO (IOBs)

Today‟s FPGAs provide support for dozens of I/O standards thus providing the ideal interface bridge in your system. I/O in FPGAs is grouped in banks with each bank independently able to support different I/O standards. Today‟s leading FPGAs provide over a dozen I/O banks, thus allowing flexibility in I/O support.

- Memory

Embedded Block RAM memory is available in most FPGAs, which allows for on-chip memory in your design. These allow for on-chip memory for your design. Xilinx FPGAs provide up to 10Mbits of on-chip memory in 36kbit blocks that can support true dual-port operation.

- Complete Clock Management

Digital clock management is provided by most FPGAs in the industry (all Xilinx FPGAs have this feature). The most advanced FPGAs from Xilinx offer both digital clock management and phase-looped locking that provide precision clock synthesis combined with jitter reduction and filtering.

ARITHMETIC LOGIC UNIT

• Hardware Overview:

The fundamental building block of ALU is shown in Fig. . Here bold dots (dip switches) indicate the inputs to an ALU which are selected by the user. The input can either be 1 or 0 indicating 5V and 0 V respectively. A LCD is used to display 16 bit output, whereas * indicates LEDs which are used to show the status of carry out, overflow, borrow and zero. The VHDL code which implies the hardware part of ALU is downloaded on FPGA processor using JTAG cable interfacing PC and the hardware element.

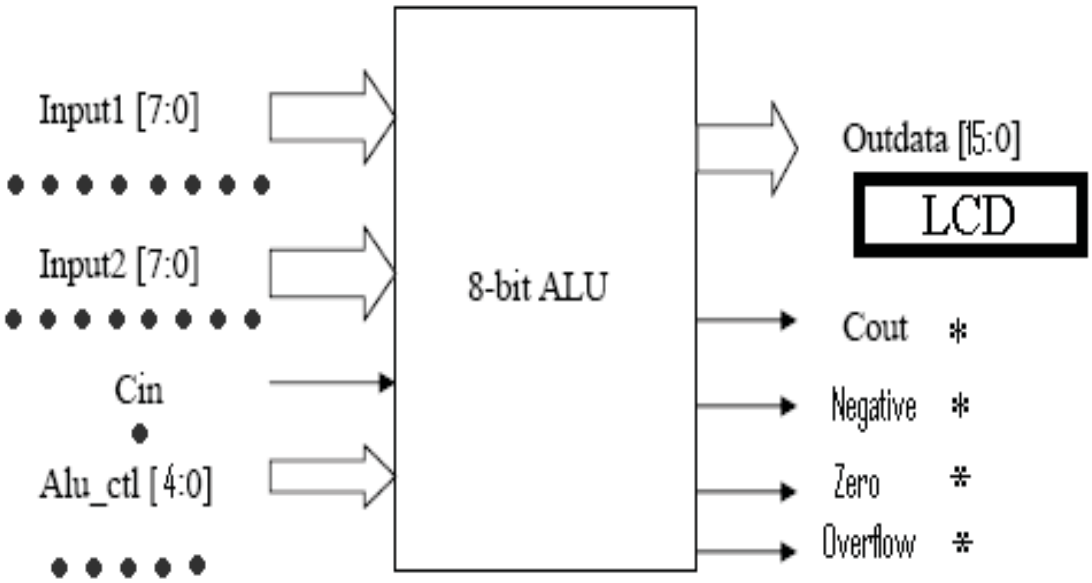


Fig 5. Hardware representation of an 8-bit ALU.

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, and then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip.

• Operational overview

The operational overview deals with two kinds of operations which an ALU can perform. First part deals with arithmetic computations and is referred to as Arithmetic Unit. It is capable of addition, subtraction, multiplication, division, increment and decrement. The second part deals with the Gated results in the shape of AND, OR, XOR, inverter, rotate, left shift and right shift, which is referred to as Logic Unit. The functions are controlled and executed by selecting operation or control bits. A select input of 5 bit size that will accommodate up to 32 operations is sufficient to achieve the objectives. The operations selected by the Control Unit are shown in the Fig. . Arithmetic part is quite complex as compared to logic unit and involves an additional carry input. Multiplication and Division also increases the complexity of ALU. In the Logic Block, gates such as AND, OR, XOR and NOT operations are shown. Logic Block of ALU does not need as many gates as required in Arithmetic Unit and if done separately, the LOGIC unit can be implemented using

Complex Programmable Logic Devices (CPLD) or other Programmable Logic Device (PLD) technologies instead of using FPGA.

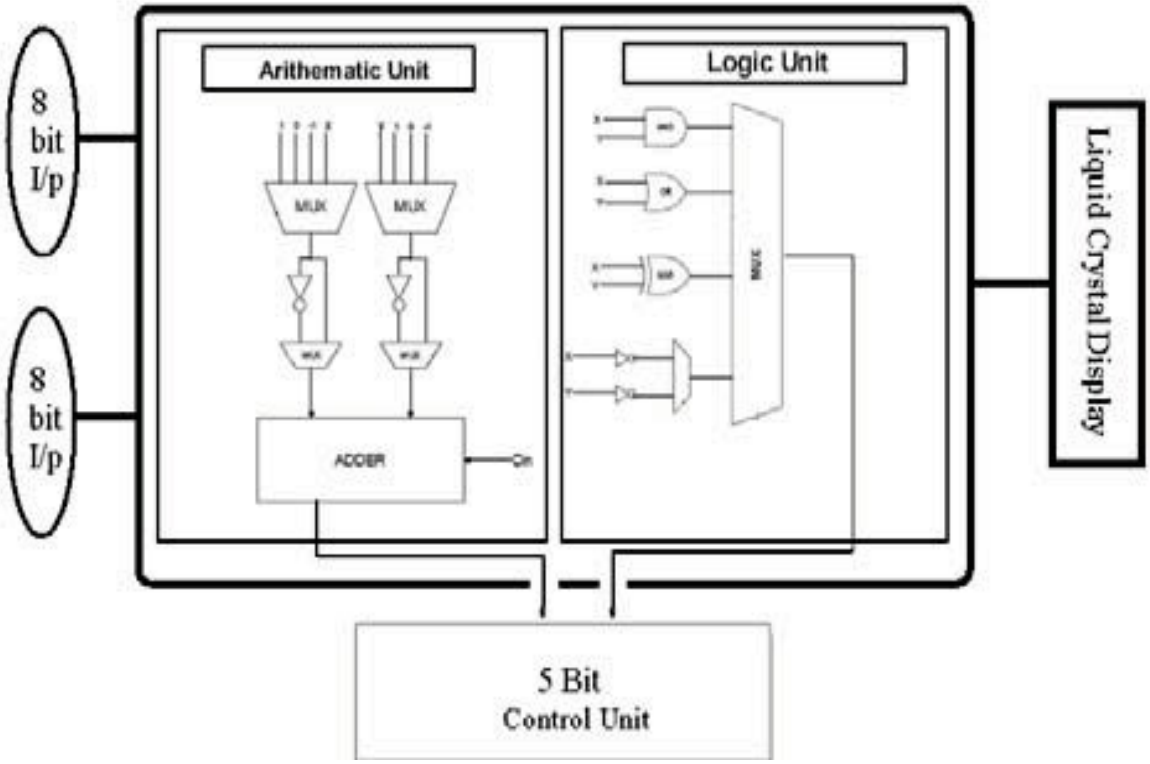


Fig.6 : General operation – ALU sub-blocks and control units using multiplexers

In this method, an ALU is chopped into several segments each incorporating its specific operations. A model of this technique containing Arithmetic, Logic, Multiplexer and Shift and Rotate Units are shown in Fig ..

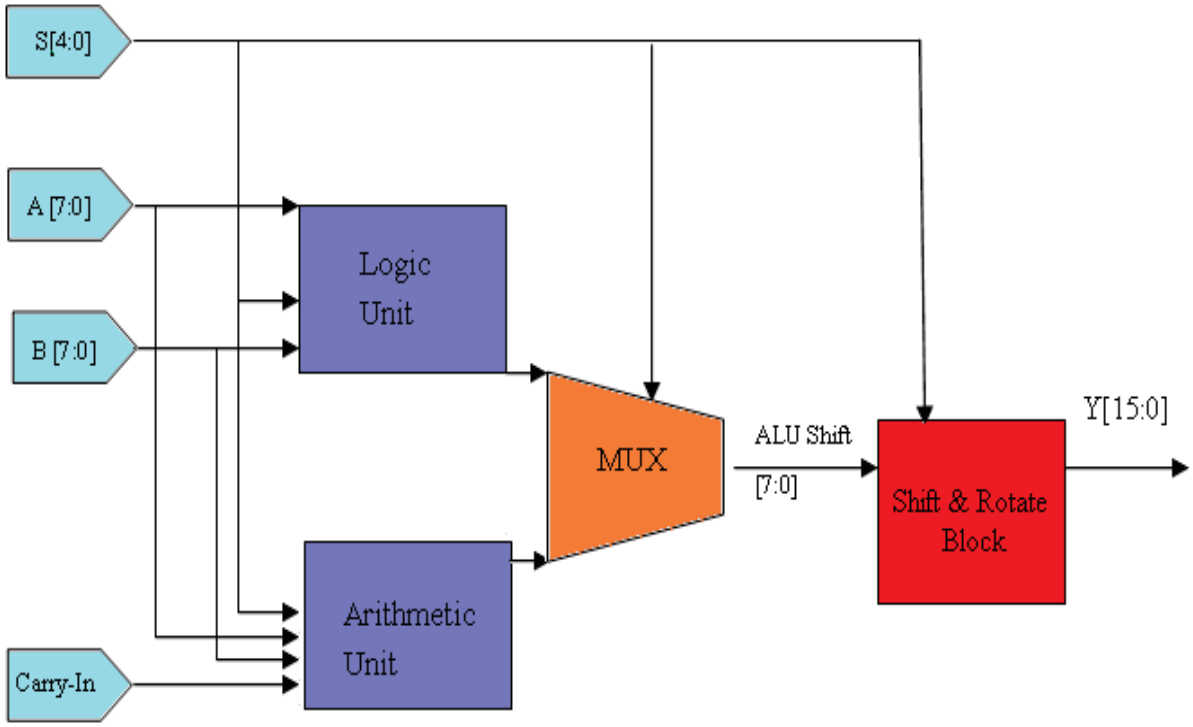


Fig.7. Block Diagram of ALU

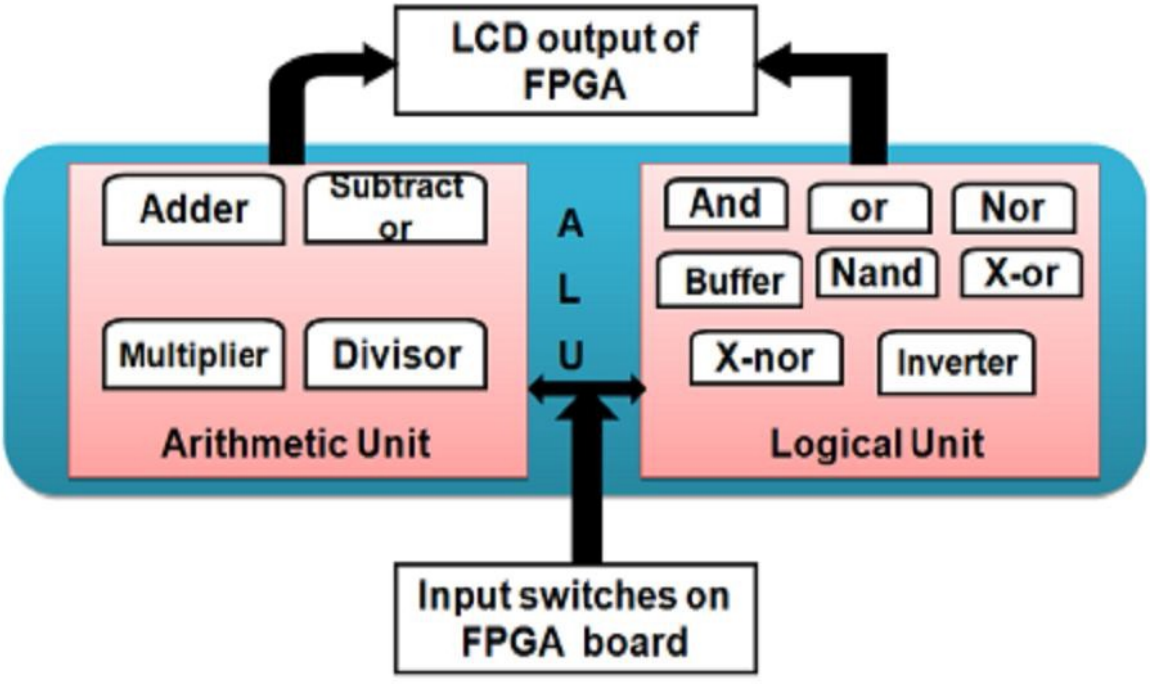


Fig.8: diagram of 8-bit ALU.

- Truth Table Design

Truth table design is an effective method compiling all those functions that are needed by the user on a single platform. Table 1 indicates the combination of five bit control input S [4 down to 0] with their operation and functions that are used in ALU. Arithmetic, Logic and Shifter units are separated in the table.

Table 1 : Combination of five bit control input S [4 down to 0] with their operation and functions that are used in ALU.

Decimal Value	S4	S3	S2	S1	S0	Operation	Function	Implementation
0	0	0	0	0	0	Y<=a+b	Addition	Arithmetic
1	0	0	0	0	1	Y<=a+ 2's C of b	Subtraction	Arithmetic
2	0	0	0	1	0	Y<=a*b	Multiplication	Arithmetic
3	0	0	0	1	1	Y<=a/b	Division	Arithmetic
4	0	0	1	0	0	Y<=a AND b	AND	Logical
5	0	0	1	0	1	Y<=a OR b	OR	Logical
6	0	0	1	1	0	Y<=a NAND b	NAND	Logical
7	0	0	1	1	1	Y<=a NOR b	NOR	Logical
8	0	1	0	0	0	Y<= a NOT b	NOT	Logical
9	0	1	0	0	1	Y<=a XOR b	XOR	Logical

Examples for arithmetic operations in ALU

- Binary Adder-Subtractor:

The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations. 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 10. The first three operations produce sum of one digit, but when the both augends and addend bits are equal 1, the binary sum consists of two digits. The higher significant bit of the result is called carry .When the augends and addend number contains more significant digits, the carry obtained from the addition of the two bits is called half adder. One that performs the addition of three bits (two significant bits and a previous carry) is called half adder. The name of circuit is from the fact that two half adders can be employed to implement a full adder.A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers. Connecting n full adders in cascade produces a binary adder for two n-bit numbers.

❖ Binary Adder:

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adder connected in cascade, the output carry from each full adder connected to the input carry of the next full adder in the chain. Fig. 4 shows the interconnection of four full adder (FA) circuits to provide a 4 bit binary ripple carry adder. The augends bits of A and addend bits of B are designated by subscript

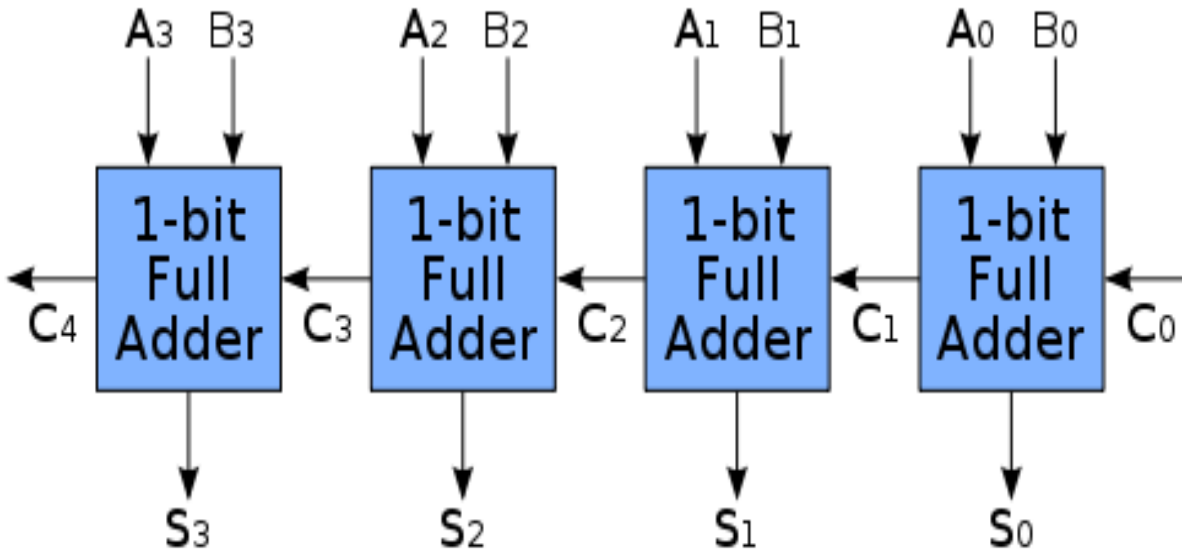


Fig.9. Binary Adder

numbers from right to left, with subscript 0 denoting the least significant bit. The carries are connected in the chain through the full adders. The input carry to the adder is C0 and it ripples through the full adder to the output carry C4.The S output generate the required sum bits. An n-bit adder requires n full adders with each output connected to the input carry of the next higher order full adder.

❖ Binary Subtractor:

The subtraction of unsigned binary numbers can be done most conveniently by means of complement. Subtraction A–B can be done by taking the 2‟s complement of B and adding it to A. The 2‟s complement can be obtained by taking the 1‟s complement and adding one to the least significant pair of bits. The 1‟s complement can be implemented with the inverters and a one can be added to the sum through the input carry. The circuit for subtracting, A–B, consists of an adder with inverter placed between each data input B and the corresponding input of the full adder. The input carry C0 must be equal to 1when performing subtraction. The operation thus performed becomes A, plus the 1‟s complement of B, plus 1.This is equal to A plus 2‟s complement of B. For unsigned numbers this gives A–B if A ≥ B or the 2‟s complement of (B–A) if A < B. for signed numbers, the result is A – B, provided that there is no overflow.

Examples for Logical operations in ALU

In a 4-bit Arithmetic Logic Unit, logical operations are performed on individual bits.

❖ EX-OR Gate:

In a 4 bit ALU, the inputs given are A0, A1, A2, A3 and B0, B1, B2, B3. Operations are performed on individual bits. Thus, as shown in a fig., inputs, A0 and B0 will give output F0.

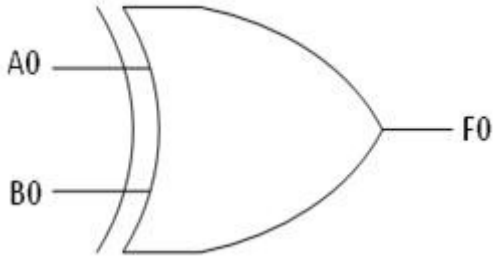


Fig.10 Ex-OR Gate

Inputs		Output
A0	B0	F0
0	0	0
0	1	1
1	0	1
1	1	0

Table: Truth table for Ex-OR Gate

Implementation

The VHDL coding of this paper design is compiled and simulated using FPGA using Xilinx Spartan XC3S100E kit shown in Fig.11 . The data is updated to the kit using two separate select inputs A and B each carrying 8 bits. The function of FPGA is embedded on the kit along with PROM, LCD, LEDs and DIP switches. A Joint Test Action Group (JTAG) interface connects the FPGA chip with PROM and leads to PC through a serial interface. The structure of such a PROM assembly XC10S is shown in Fig12.. Since FPGA is user programmable, therefore JTAG is of core significance. PROM has several postulates in the shape of data storage and debugging, permanent storage of data, consistency of operation, low cost, high speed and compactness. PROM used in this design of ALU is “XC10S”, which is equipped with the inbuilt circuitry to support and store complex functions. It supports both mode of Master and Slave serial Field Programmable Gate Array.

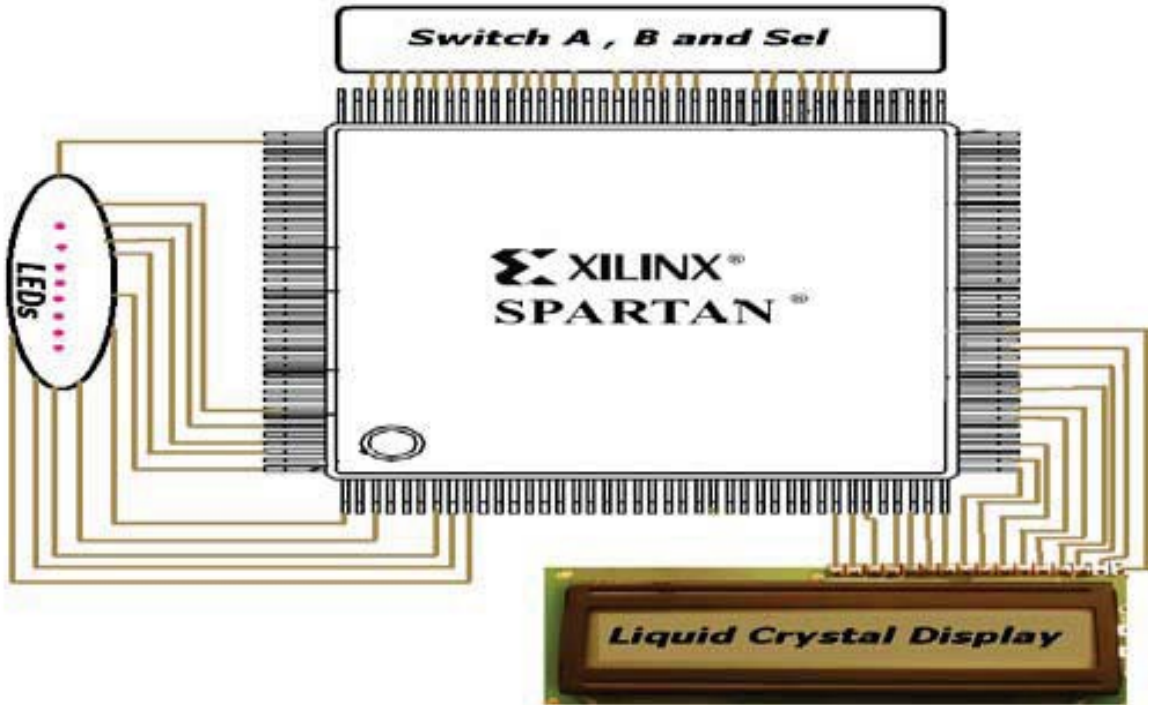


Fig.11:Circuit design of FPGA kit.

In real time application, after the process of compilation and simulation of the VHDL design, the hardware realization is constructed and tested as shown in Fig. 12. Here the 8-bit inputs are given by means of two sets of DIP switches and the 16-bit output can be displayed on a LCD panel and the result can be verified with the simulated output. The status of the flag register is indicated by a series of 8-bit LEDs. The provision of a select switch used in this hardware enables the user to perform the required operation on the FPGA processor.

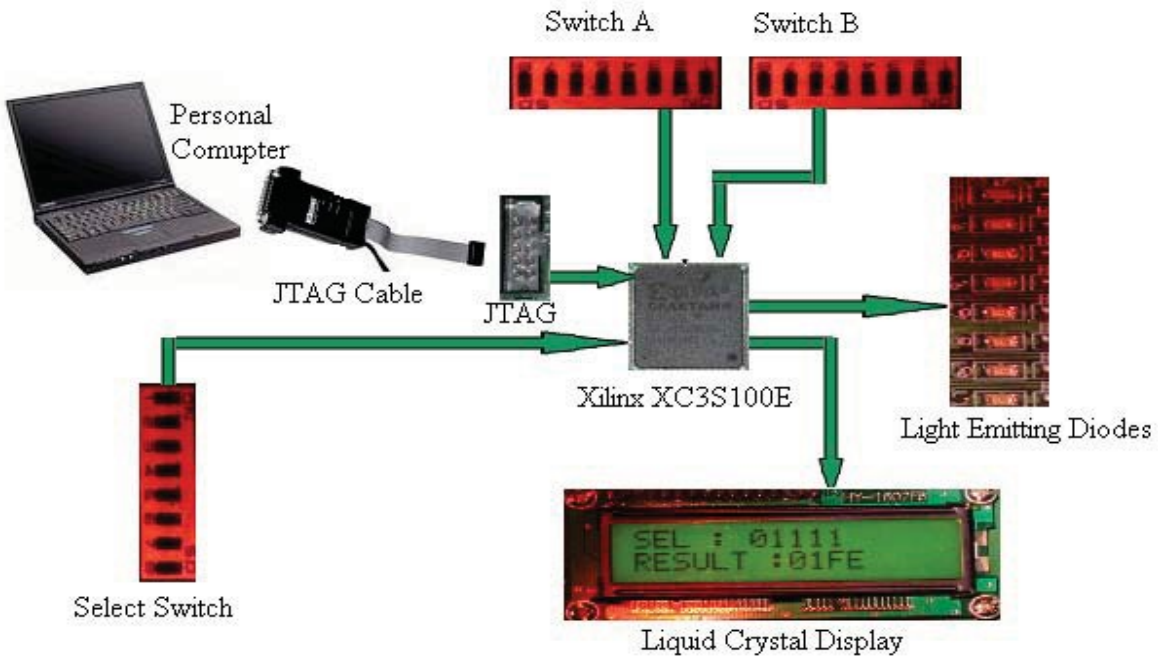


Fig.12 : Real time hardware implementation

CONCLUSION

VHDL implementation of 8-bit arithmetic logic unit (ALU) is presented. The design was implemented using VHDL Xilinx Synthesis tool and targeted for Spartan device. ALU was designed to perform arithmetic operations such as addition and subtraction using 8-bit fast adder, logical operations such as AND, OR, XOR and NOT operations, 1's and 2's complement operations and compare and many more operations.

High level design methodology allows managing the design complexity in a better way and reduces the design cycle. A high-level model makes the description and evaluation of the complex systems easier. RTL description specifies all the registers in a design, and the combinational logic between them. The registers are described either explicitly through component instantiation or implicitly through inference. The combinational logic is described by logical equations, sequential control statements subprograms, or through concurrent statements. Designing at a higher level of abstraction delivers the following benefits:

- • **Manages complexity:** Fewer lines of code improves productivity and reduces error.
- • **Increases design reuse:** Implementation of independent designs as cell library & reuse in various models.
- • **Improves verification:** Helps to run process faster