A Project Report on

4-BIT ALU USING VERILOG

Submitted by

**SWAPNIL PHALKE (ROLL NO. 13)**

**MIKHAIL PINTO (ROLL NO. 14)**

**SVILZOR PINTO (ROLL NO. 15)**

**JANAK PISHARODY (ROLL NO. 16)**

In fulfillment of

**MINI-PROJECT 2**

in

**Electronics & Telecommunication Engineering**

Under the Guidance of

**Ms. ANJALI CHOUDHARI**



Department of Electronics and Telecommunication Engineering

St. Francis Institute of Technology, Mumbai

University of Mumbai

(2016-2017)

**ABSTRACT**

The main purpose of this project was to study the designing of a 4-bit ALU on hardware description language (HDL). This was implemented on Verilog using structural method. Using k-map technique, 15 operations were selected including arithmetic, logical and shifting operations. And the code was developed such that using multiplexing, blocks were made for each type of operation which facilitated easy selection.

The tasks undertaken were:

* Study of Verilog software.
* Block design of the ALU structure.
* Selection and grouping of the required operations.
* Systematic implementation of the code using available functions and function call provided in the software.
* Verification of results and rectification of possible issues in the code.

**CERTIFICATE**

This is to certify that the project entitled **“4-Bit Arithmatic and Logical unit usin Verilog Software”** is a bonafide work of **“Swapnil Phalke (Roll No.13), Mikhail Pinto(Roll No.14), Svilzor Pinto(Roll No.15), Janak Pisharody(Roll No.16)”** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering** in **Electronics and Telecommunication Engineering**.

|  |  |
| --- | --- |
| Ms. Anjali Choudhari |  |
| Internal Guide |  |

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Internal Examiner | External Examiner |

**ACKNOWLEDGEMENT**

This project was the outcome of efforts of many people including the students, the guide, and some more teachers who helped us throughout the project for it to have the desired outcome. The project fostered learning and application of concepts that were learned in the lectures. Selection of an appropriate topic was another area where the knowledge in hand and analysis by the guide helped. In difficulties it was the friends who gave a helping hand where we fell short.

First of all at completion of our project we articulate our deep appreciation to:

* Our Guide Ms. Anjal Choudhari, who supported us and mentored us for the planning and execution of the project and provided all necessary resources in terms of permissions.
* Our Teachers who taught us the pre-requisites for structural design of different systems and digital electronics also the softwares.
* Nikita Ma’am for helping us when we were stuck in between with no clue to move ahead.
* Ravindra Sir for helping us from the inception of the project with the idea to introducing us to Verilog and giving a direction.
* And our friends who helped us through discussions when we fell in mid-time crisis.
* The digital electronics lab incharge and assistant who allowed us resources whenever we needed it owing to our limitations in the available software etc.

|  |  |  |
| --- | --- | --- |
| **Contents** | | |
| **Chapter 1** | **Introduction** | **1** |
| * 1. Motivation | 1 |
| * 1. Scope of the Project | 2 |
| 1. Methodology | 3 |
| 1. Organization of Project Report |
| **Chapter 2** | **Design Methodology** | **4** |
| 2.1 |  |
| 2.2 K-map |  |
| 2.3 XILIN |
| 2.4 Verilog v/s VHDL |  |
| **Chapter 3** | **Software and Hardware Support** | **13** |
| 1. DESIGN | 14 |
| 1. ALU Structure | 15 |
| 1. Selection K-map | 16 |
| 1. Code | 23 |
| 1. Test Bench | 24 |
| **Chapter 4** | **Simulation and Results** | **35** |
|  |  | 35 |
|  |  | 36 |
| **Chapter 5** | **Conclusion** | **60** |
|  | 1. Conclusion | 60 |
|  | 1. Future Scope | 60 |
| **References** |  | **62** |
| **Appendix** |  | **63** |

List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Captions** | **Page No.** |
| 1 | K-map for Arithmetic Operations |  |
| 2 | K-map for Incrementer and Decrement |  |
| 3 | 4-Bit ALU Design |  |
| 4 | Output of Increment |  |
| 5 | Output of Decrement |  |
| 6 | Output of Add |  |
| 7 | Output of Add+1 |  |
| 8 | Output of Subtractor |  |
| 9 | Output of Sub-1 |  |
| 10 | Output of Left shift |  |
| 11 | Output of left rotate |  |
| 12 | Output of Right Shift |  |
| 13 | Output of Right Rotate |  |
| 14 | Output of 1's Compliment |  |
| 15 | Output of X-OR |  |
| 16 | Output of OR |  |
| 17 | Output of AND |  |

List of Abbreviations

|  |  |
| --- | --- |
| ALU | Arithmetic and Logical Unit |
| CPU | Central Processing Unit |
| ISE | Integrated Synthesis Environment |
| FGPA | Field Programmable Gate Array |

ASIC Application Specific Integrated Circuit

**Chapter 1**

**Introduction**

ALU or Arithmetic and Logical Unit is the heart of a system which does all the data modifications and gives the result according to the given data. A 4-bit ALU as seen here will take 2 inputs of 4 bit each and then do the required operation on them and give the 4-bit output with the carry if available. The required operations are selected using some select lines. This complete ALU structure is being implemented here with the help of a HDL code simulated in a Verilog coder.

* 1. **Motivation**

This project facilitated a platform to study and know about a new programing language and about how to implement a very common but complex system component using the program. HDL language helps to design the structure very efficiently as per the personal requirement, so there was also some scope of putting in our creativity.

* 1. **Scope of the Project**

Implementation of 4-bit ALU which does up to 14 functions based on the selection of the user.

**1.3 Organization of Project Report**

This project report is organized as follows:

Chapter 2 Methodology

Chapter 3: Designing of a ALU structure (block diagram) with selected operations. Coding on paper

Chapter 4: Simulation of the code in Structural format in Verilog software

Chapter 5: Conclusion and result of project and tracing the future scope.

**Chapter 2**

**Design Methodology**

The ALU can be designed by implementing the required arithmetic and logical operations and then designing a block using the multiplexer logic. Selection of the block and the operation is done by using the select lines at each stage which comes up to 5 lines in the structure shown over here. That gives up to 2^5=32 options. Of these, 15 operations are decided and using K-map logic the relevant select logic is defined.

After the block design stage, a new stage that is the shifting block is also implemented. The complete block structure is then connected using the multiplexer logic.

Based on this design, the Verliog code is first designed. This code is first written on the notepad with ‘.v’ extension. And added as a new source in the Verilog file. Once the syntax and code is checked and errors are rectified if any. By implementing the design, we obtain a schematic of the complete structure with the input and output modules properly defined. When the code is implemented without any errors, test bench module is made which controls the input variable data. On executing the behavioral model, we get the output waveforms for each operation and the input data which is given to the test bench fixture.

Verilog software provides us with the design schematic and the waveforms. There is flexibility in implementing functions as any new source can be either added of a new one can be created and the software automatically links it with the main source and the changes to the schematic is made.

**2.1 ALU**

An Arithmetic and Logical Unit(ALU) is a digital circuit, used to perform Arithmetic and logical operations. It represents the fundamental building block of Central Processing Unit(CPU) of a computer. Modern CPUs contain very powerful and complex ALUs.

**2.2 K-MAP**

The **Karnaugh map** (**KM** or **K-map**) is a method of simplifying [Boolean algebra](https://en.wikipedia.org/wiki/Boolean_algebra) expressions. [Maurice Karnaugh](https://en.wikipedia.org/wiki/Maurice_Karnaugh) introduced it in 1953[[1]](https://en.wikipedia.org/wiki/Karnaugh_map#cite_note-Karnaugh_1953-1) as a refinement of [Edward Veitch](https://en.wikipedia.org/wiki/Edward_Veitch)'s 1952 **Veitch chart**.[[2]](https://en.wikipedia.org/wiki/Karnaugh_map#cite_note-Brown_2012-2) The Karnaugh map reduces the need for extensive calculations by taking advantage of humans' pattern-recognition capability.[[1]](https://en.wikipedia.org/wiki/Karnaugh_map#cite_note-Karnaugh_1953-1) It also permits the rapid identification and elimination of potential [race conditions](https://en.wikipedia.org/wiki/Race_condition).

The required Boolean results are transferred from a [truth table](https://en.wikipedia.org/wiki/Truth_table) onto a two-dimensional grid where the cells are ordered in [Gray code](https://en.wikipedia.org/wiki/Gray_code), and each cell position represents one combination of input conditions, while each cell value represents the corresponding output value. Optimal groups of 1s or 0s are identified, which represent the terms of a [canonical form](https://en.wikipedia.org/wiki/Canonical_form_(Boolean_algebra)) of the logic in the original truth table.[[3]](https://en.wikipedia.org/wiki/Karnaugh_map#cite_note-Belton_1998-3) These terms can be used to write a minimal Boolean expression representing the required logic.

Karnaugh maps are used to simplify real-world logic requirements so that they can be implemented using a minimum number of physical logic gates. A [sum-of-products expression](https://en.wikipedia.org/wiki/Conjunctive_normal_form) can always be implemented using [AND gates](https://en.wikipedia.org/wiki/AND_gate) feeding into an [OR gate](https://en.wikipedia.org/wiki/OR_gate), and a [product-of-sums expression](https://en.wikipedia.org/wiki/Disjunctive_normal_form) leads to OR gates feeding an AND gate.[[4]](https://en.wikipedia.org/wiki/Karnaugh_map#cite_note-Dodge_2016-4) Karnaugh maps can also be used to simplify logic expressions in software design. Boolean conditions, as used for example in [conditional statements](https://en.wikipedia.org/wiki/Conditional_(programming)), can get very complicated, which makes the code difficult to read and to maintain. Once minimised, canonical sum-of-products and product-of-sums expressions can be implemented directly using AND and OR logic operators.[1]

**2.3 Xilin**

Xilin ISE (Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.  
The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes).

The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. For single-chip designs there may be one main module, with other modules included by the main module, similar to the main() subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping.

The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each function.

The Transcript window provides status of currently running operations, and informs engineers on design issues. Such issues may be filtered to show Warnings, Errors, or both.

SimulationEdit

System-level testing may be performed with ISIM or the ModelSim logic simulator, and such test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test.

ModelSim or ISIM may be used to perform the following types of simulations:

* Logical verification, to ensure the module produces expected results.
* Behavioural verification, to verify logical and timing issues.
* Post-place & route simulation, to verify behaviour after placement of the module within the reconfigurable logic of the FPGA

SynthesisEdit

Xilinx's patented algorithms for synthesis allow designs to run up to 30% faster than competing programs, and allows greater logic density which reduces project time and costs.[2]

**2.4 Why we took VERILOG than VHDL?**

If you look at the synthesizable subset of both languages, they are very similar in capabilities, but code written in Verilog will have much better performance than the same code written in VHDL. Verilog was much more popular for larger ASIC designs, which had performance and capacity demands on their simulations that FPGA designs did not. So  geographical areas where ASIC design was more popular will also be  where Verilog will be more popular.

For the rest of the non-synthesizable portion of the two languages, VHDL certainly had more capabilities than Verilog, but SystemVerilog borrowed a lot from VHDL and is now the predominant language for writing test benches.

If you work at the synthesis level (RTL) there isn't much difference, however Verilog has support for bidirectional primitives that make it better for ASIC verification.

VHDL seems better to academics who don't actually understand electronics but have a computer science background. However most HDLs are a syntactic and semantic mess that are inadequate for the job of designing chips, which is why the first-spin success rate on Silicon is something like 30% (a failing grade).[3]

**Chapter 3**

**Design And Coding Methodology**

Working

1) Arithmetic Operations

The 4-bit adder subtractor circuit using Full Adder block is  used the perform the arithmetic operations which can be selected using the select lines. The operations included in this block are:

Increment

Decrement

Addition

Add +1

Subtraction

Subtract -1

For performing this operation, we use the basic circuit of 4-bit adder/subtractor using full adder. For the modification like increment, decrement, add+1 and subtract-1, we made small changes using K-map. As shown in the figure, when m=0, the output of XOR, is complement of B and when m=1, then B will be as it is at the output. For this purpose, the designed K-map is as shown below.

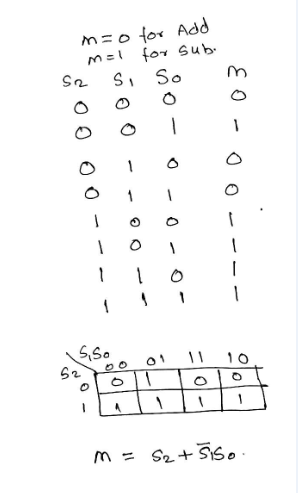


Figure 1:K-map for Arithmetic Operations

The carry of the 1st full adder should be equal to 1 for operations like increment, add +1 and subtract. For this purpose, the designed K-map is as shown below.

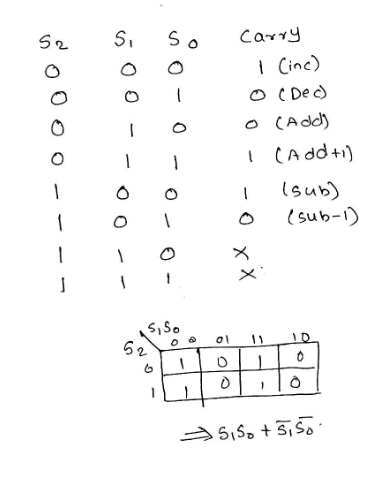


Figure 2:K-map for Incrementer and Decrement

For the special operation like increment and decrement, B input should be equal to 0.

Therefore we use the OR gate which is 0 only in case where both inputs are 0.

2) Logical Operation

Quad 4:1 MUX:

   It is a MUX that takes 4 input and gives one output but the input given is each are of 4 bits.

Therefore here, normal gates XOR OR AND and NOT has been used. And output is given to Quad 4:1 MUX.

Therefore using S1 and S0, we perform specific operation.

3) Shift and Rotate Block

Quad 2:1 MUX:

   It is a MUX that takes 2 input and gives one output, the input given is each are of 1 bit.

Therefore, here 4, 2:1 MUX are used.

S1 decides whether to take upper bit or lower bit of the two inputs given to mux.

Left shift, rotate:

 Here S1=0 therefore, output is A2 A1 A0 AND(A3,S0)

Therefore when S0 =0 shift operation is performed and when S0=1, rotate operation is performed.

Right shift, rotate:

 Here S1=0 therefore, output is AND(S0,A0) A3 A2 A1

Therefore when S0 =0 shift operation is performed and when S0=1, rotate operation is performed.

Main  Block: Quad 4:1 MUX

Quad 4:1 MUX:

   It is a MUX that takes 4 input and gives one output but the input given is each are of 4 bits.

All the output that has be obtained from individual blocks has been applied as an input to this MUX.

And using S4 and S3, Arithmetic, Logical and Shifting/Rotating operation is performed[4]

**3.1 DESIGN**

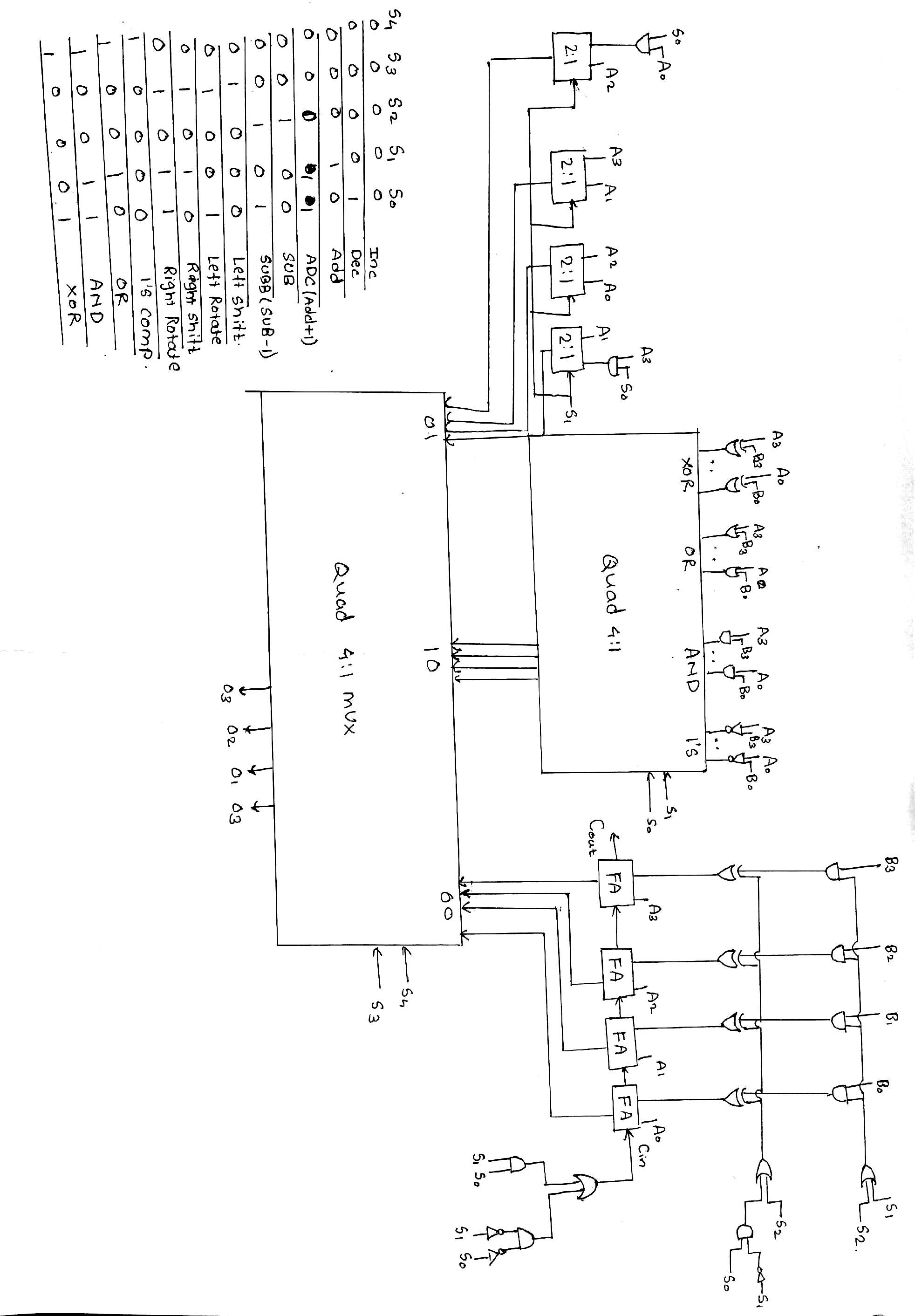


Figure 3: 4-Bit ALU Design

**3.2 Software Implementation and CODE**

//NOT-Logical

module not\_opration(v,a);

output [3:0]v;

input [3:0]a;

not(v[0],a[0]);

not(v[1],a[1]);

not(v[2],a[2]);

not(v[3],a[3]);

endmodule

//XOR-Logical

module xor\_opration(v,a,b);

input [3:0]a;

input [3:0]b;

output [3:0]v;

xor(v[0],a[0],b[0]);

xor(v[1],a[1],b[1]);

xor(v[2],a[2],b[2]);

xor(v[3],a[3],b[3]);

endmodule

//4-BIT ALU main code

module alu\_4bit(a,b,o,s4,s3,s2,s1,s0,cout1);

input [3:0]a ;

input [3:0]b;

input s4;

input s3;

input s2;

input s1;

input s0;

output [3:0]o;

output cout1;

wire [3:0]p;

wire [3:0]l;

wire [3:0]opl;

wire [3:0]x;

arithmatic\_opration fa1(s1,s2,s0,b,a,p,cout1);

multiple\_opration fa2(s0,s1,a,b,l);

logical\_opration fa3(s0,s1,a,b,opl);

mux\_41opration fa4(o,p,l,x,opl,s4,s3);

assign x=0;

endmodule

//Arithmetic

module full\_adder(x,y,z,l,m);

input z;

input l;

input m;

output x;

output y;

wire o[3:0];

xor(o[0],z,l);

xor(x,m,o[0]);

and(o[1],z,l);

and(o[2],z,m);

and(o[3],l,m);

or(y,o[1],o[2],o[3]);

endmodule

//Arithmetic operations

module arithmatic\_opration(s1,s2,s0,b,a,p,cout);

input [3:0]a;

input [3:0]b;

input s2;

input s1;

input s0;

output [3:0]p;

output cout;

wire [15:0]k;

wire c1,c2,c3;

or(k[0],s1,s2);

and(k[1],b[3],k[0]);

and(k[2],b[2],k[0]);

and(k[3],b[1],k[0]);

and(k[4],b[0],k[0]);

not(k[5],s1);

and(k[6],s0,k[5]);

or(k[7],s2,k[6]);

xor(k[8],k[7],k[1]);

xor(k[9],k[7],k[2]);

xor(k[10],k[7],k[3]);

xor(k[11],k[7],k[4]);

not(k[13],s0);

not(k[12],s1);

and(k[14],s0,s1);

and(k[15],k[12],k[13]);

or(cin,k[14],k[15]);

full\_adder sa1(p[0],c1,a[0],k[11],cin);

full\_adder sa2(p[1],c2,a[1],k[10],c1);

full\_adder sa3(p[2],c3,a[2],k[9],c2);

full\_adder sa4(p[3],cout,a[3],k[8],c3);

endmodule

//OR-Logical

module or\_opration(v,a,b);

input [0:3]a;

input [0:3]b;

output[0:3]v;

or(v[0],a[0],b[0]);

or(v[1],a[1],b[1]);

or(v[2],a[2],b[2]);

or(v[3],a[3],b[3]);

endmodule

//AND-Logical

module and\_opration(v,a,b);

input [0:3]a;

input [0:3]b;

output [0:3]v;

and(v[0],a[0],b[0]);

and(v[1],a[1],b[1]);

and(v[2],a[2],b[2]);

and(v[3],a[3],b[3]);

endmodule

//SHIFT

module multiple\_opration(s0,s1,a,b,op1);

output [3:0]op1;

input [3:0]a;

input [3:0]b;

input s0;

input s1;

wire y1,y2;

and(y1,s0,a[3]);

and(y2,s0,a[0]);

mux\_21opration d1(op1[0],y1,a[1],s1);

mux\_21opration d2(op1[1],a[0],a[2],s1);

mux\_21opration d3(op1[2],a[1],a[3],s1);

mux\_21opration d4(op1[3],a[2],y2,s1);

endmodule

//Logical operations

module logical\_opration(s0,s1,a,b,opl);

input [3:0]a;

input [3:0]b;

input s0;

input s1;

output [3:0]opl;

wire [3:0]x;

wire [3:0]ko;

wire [3:0]oo;

wire [3:0]no;

mux\_41opration f0(opl,no,oo,ko,x,s0,s1);

xor\_opration f1(x,a,b);

and\_opration f2(ko,a,b);

not\_opration f3(no,a);

or\_opration f4(oo,a,b);

endmodule

//Selection MUX 2:1

module mux\_21opration(u,i,t,s1);

input i;

input t;

input s1;

output u;

assign

u=(s1==0?i:t);

endmodule

//MUX 4:1 Assignment

module mux\_41opration(l,v1,v2,v3,v4,s4,s3);

parameter WIDTH=4;

input [3:0]v1;

input [3:0]v2;

input [3:0]v3;

input [3:0]v4;

input s4;

input s3;

output[3:0]l;

assign l=(s4==0?(s3==0?v1:v2):(s3==1?v3:v4));

endmodule

**3.3 TEST BENCH For Simulation**

module kl\_v;

// Inputs

reg [3:0] a;

reg [3:0] b;

reg s4;

reg s3;

reg s2;

reg s1;

reg s0;

// Outputs

wire [3:0] o;

wire cout1;

// Instantiate the Unit Under Test (UUT)

alu\_4bit uut (

.a(a),

.b(b),

.o(o),

.s4(s4),

.s3(s3),

.s2(s2),

.s1(s1),

.s0(s0),

.cout1(cout1)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

s4 = 0;

s3 = 0;

s2 = 0;

s1 = 0;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

end

endmodule

**Chapter 4**

**Simulation and Experimental Results**

**4.1 Simulation and Results**

1. Increment

initial begin

// Initialize Inputs

a = 4;

b = 0;

s4 = 0;

s3 = 0;

s2 = 0;

s1 = 0;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

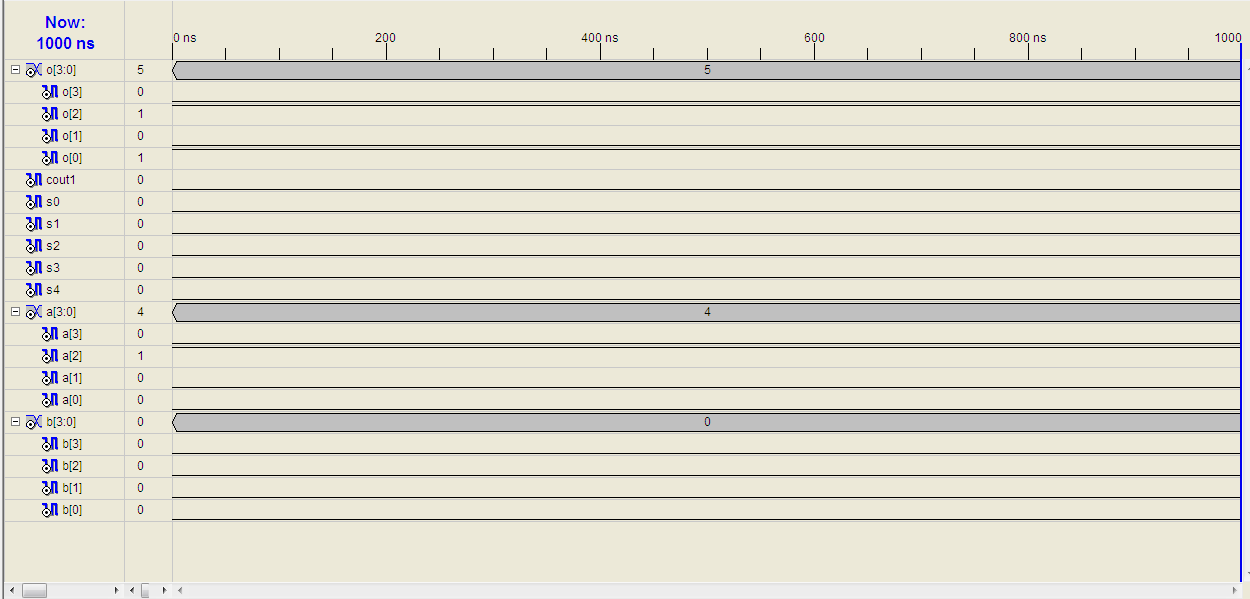


Figure 4-Output of Increment

1. Decrement

initial begin

// Initialize Inputs

a = 10;

b = 0;

s4 = 0;

s3 = 0;

s2 = 0;

s1 = 0;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 5:Output of Decrement

1. ADD

initial begin

// Initialize Inputs

a = 6;

b = 5;

s4 = 0;

s3 = 0;

s2 = 0;

s1 = 1;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 6-Output of Add

1. ADD+1

initial begin

// Initialize Inputs

a = 4;

b = 4;

s4 = 0;

s3 = 0;

s2 = 0;

s1 = 1;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

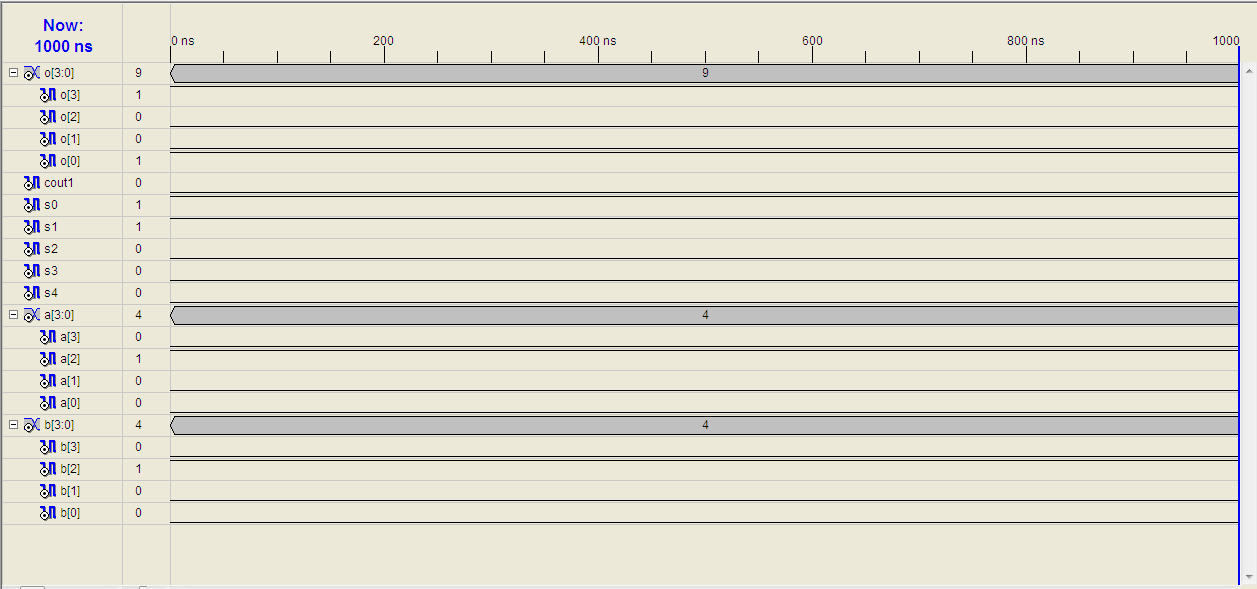


Figure 7-Output of Add+1

1. SUB

initial begin

// Initialize Inputs

a = 8;

b = 4;

s4 = 0;

s3 = 0;

s2 = 1;

s1 = 0;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 8-Output of Subtractor

1. SUB-1

initial begin

// Initialize Inputs

a = 8;

b = 4;

s4 = 0;

s3 = 0;

s2 = 1;

s1 = 0;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 9-Output of Sub-1

1. Left Shift

initial begin

// Initialize Inputs

a = 7;

b = 4;

s4 = 0;

s3 = 1;

s2 = 0;

s1 = 0;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 10-Output of Left shift

1. Left Rotate

initial begin

// Initialize Inputs

a = 9;

b = 4;

s4 = 0;

s3 = 1;

s2 = 0;

s1 = 0;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 11-Output of left rotate

1. Right Shift

initial begin

// Initialize Inputs

a = 8;

b = 4;

s4 = 0;

s3 = 1;

s2 = 0;

s1 = 1;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 12-Output of Right Shift

1. Right Rotate

initial begin

// Initialize Inputs

a = 5;

b = 4;

s4 = 0;

s3 = 1;

s2 = 0;

s1 = 1;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

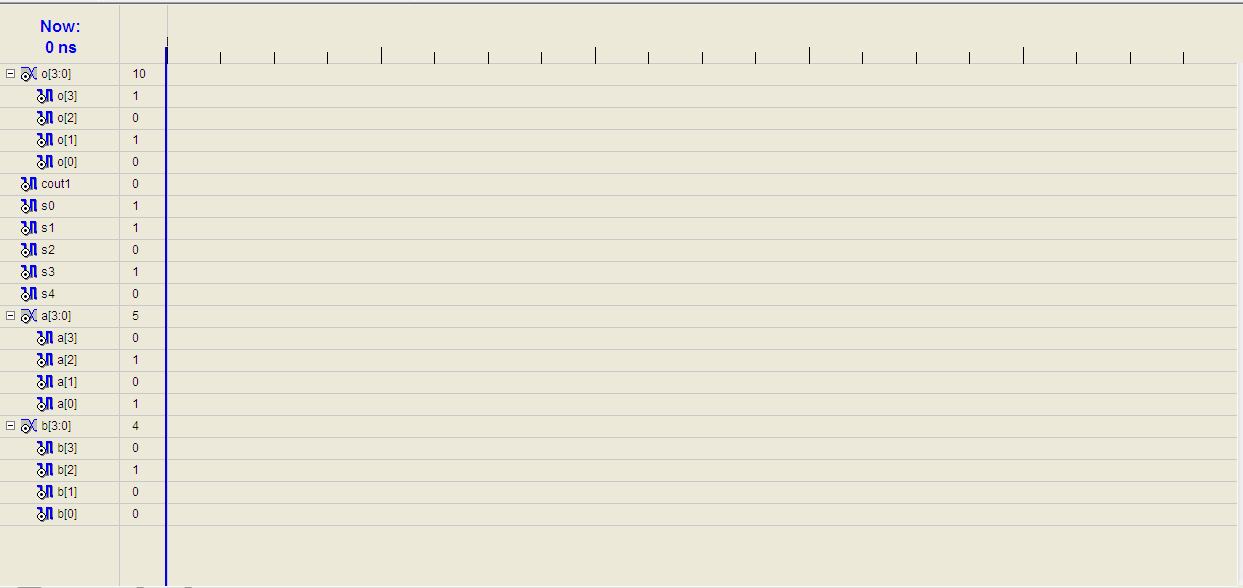


Figure 13-Output of Right Rotate

1. 1’s Compliment

initial begin

// Initialize Inputs

a = 11;

b = 9;

s4 = 1;

s3 = 0;

s2 = 0;

s1 = 0;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

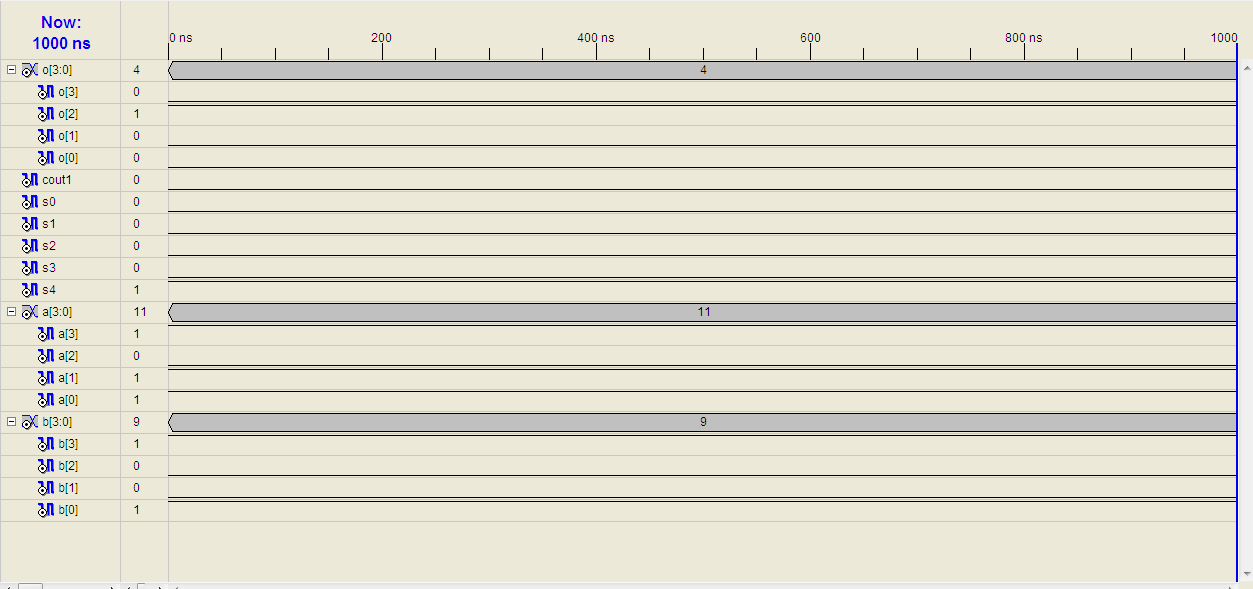


Figure 14-Output of 1's Compliment

1. XOR

initial begin

// Initialize Inputs

a = 6;

b = 9;

s4 = 1;

s3 = 0;

s2 = 0;

s1 = 0;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

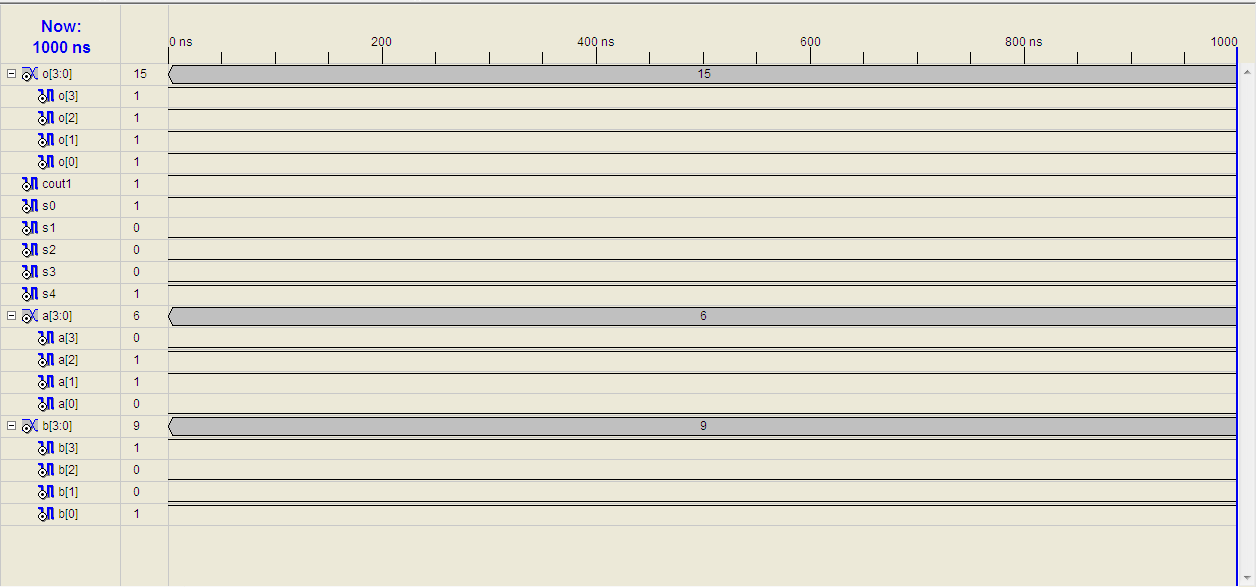


Figure 15-Output of X-OR

1. OR

initial begin

// Initialize Inputs

a = 5;

b = 3;

s4 = 1;

s3 = 0;

s2 = 0;

s1 = 1;

s0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End



Figure 16-Output of OR

1. AND

initial begin

// Initialize Inputs

a = 6;

b = 14;

s4 = 1;

s3 = 0;

s2 = 0;

s1 = 1;

s0 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

End

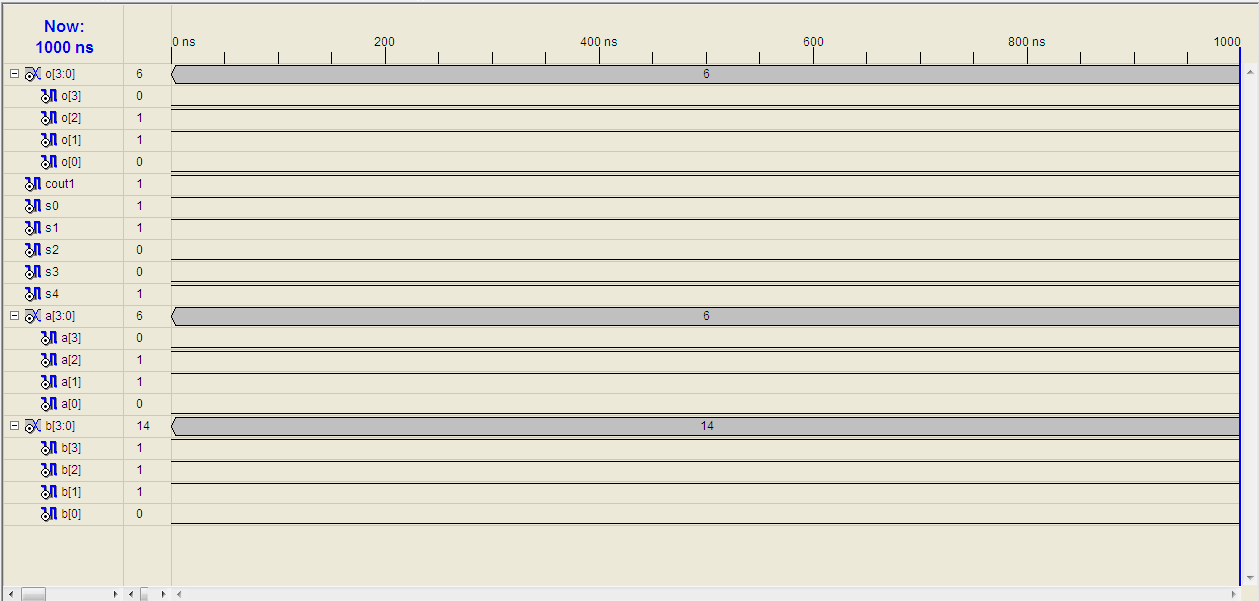


Figure 17-Output of AND

**Chapter 5**

**5.1 Conclusions**

The ALU design depends on the need of the developer such that he can add as many modules as required and create such a selection and operation logic which can connect to each other and thus give a complete schematic which can provide output for any selected functions mentioned in the list. As per requirement the K-map can be designed and coding of each function can be done which is then interlaced by the VERILOG software on its own. This project demanded immense study of the language and logic skills as well as designing capability. The outcome of this project is only on the software front as hardware implementation is not feasible for a small scale like the mini project.

**5.2 Future Scope**

The advancement which is possible in this project are:

* N-bit ALU which can take up to M numbers and operate on them.
* Better and efficient designing.
* Better design of ALU with more operations.
* Options for floating point numbers.

Adding advanced block for log, trigonometric and power based calculations can be other level of improvement to the project.

References

|  |  |
| --- | --- |
| [1] | https://en.wikipedia.org/wiki/Karnaugh\_map |
| [2] | <https://sites.google.com/site/nikitapatni04/se-extc>  https://sites.google.com/site/nikitapatni04/digital-vlsi-design |
| [3] | https://www.quora.com/Whats-the-difference-between-Verilog-and-VHDL |
| [4] | Modern Digital Electronics Paperback – 27 Jul 2009 by [R Jain](http://www.amazon.in/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=R+Jain&search-alias=stripbooks) (Author) |