VHDL Implementation of Arithmetic Logic Unit

[1]Saumyakanta Sarangi
Associate Professor
Electronics & Telecommunication Engg
Eastern Academy of Science & Technology
Bhubaneswar, India

[2]Sangita Swain
Associate Professor
Electronics & Telecommunication Engg
Eastern Academy of Science & Technology
Bhubaneswar, India

[3]Swagatika Dash
Assistant Professor
Electronics & Telecommunication Engg
Eastern Academy of Science & Technology
Bhubaneswar, India

[4]Manas Ranjan Mohanta
Assistant Professor
Electronics & Telecommunication Engg
Eastern Academy of Science & Technology
Bhubaneswar, India

Abstract—**Digital design is an amazing and very broad field. The applications of digital design are present in our daily life, including computers, calculators, video cameras etc. The VHDL (VHSIC Hardware Description Language) has become an essential tool for designers in the world of digital design. This paper presents implementation of a 4-bit Arithmetic Logic Unit (ALU) using VHDL. ALU of digital computers is an aspect of logic design with the objective of developing appropriate algorithms in order to achieve an efficient utilization of the available hardware. Here the mixed VHDL model of ALU is designed to perform 11 operations which includes both logical and arithmetic operations. The VHDL implementation and functionality test of the 4-bit ALU is done by using the Xilinx ISE 9.2i tool.**

Keywords—*ALU, VHDL, XILINX*

I. INTRODUCTION

An arithmetic logic unit (ALU) is a multi operation, combinational-logic digital function. It can perform a set of basic arithmetic operations and a set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that K selection variables can specify up to 2^k distinct operations.
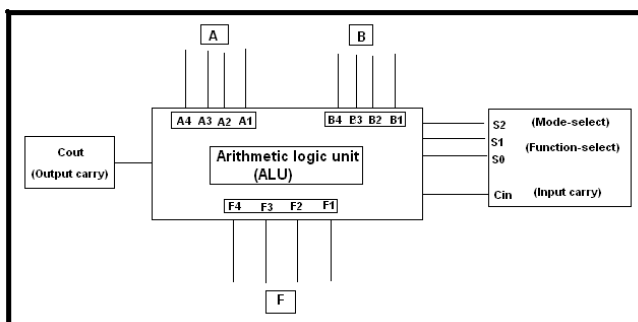

Figure:1. (Block diagram of a 4-bit ALU)

Figure 1 shows the block diagram of a 4-bit ALU. The four data inputs from A are combined with the four inputs from B to generate an operation at the F outputs. The mode-select input s2 distinguishes between arithmetic and logic operations. The two function-select input s1 and s0 specify the particular arithmetic or logic operation to be generated. With three selection variables, it is possible to specify four arithmetic operations (with s2 in one state) and four logic operations (with s2 in the other state). The input and output carries have meaning only during an arithmetic operation. The input carry in the least significant position of an ALU is quite often used as a fourth selection variable that can double the number of arithmetic operations. In this way, it is possible to generate four more operations, for a total of eight arithmetic operations.

The design of a typical ALU will be carried out in three stages. First, the design of arithmetic section will be undertaken. Second, the design of logic section will be considered. Finally, the arithmetic section will be modified so that it can perform both arithmetic and logic operations.

II. DESIGN OF ALU

We design an ALU with seven arithmetic operations and four logic operations. Three selection variables s2, s1, and s0 select eight different operations, and the input carry Cin is used to select four additional arithmetic operations. With s2=0, selection variables s1 and s0 together with Cin will select eight arithmetic operations in arithmetic table above. With s2 = 1 variables s1 and s0 will select the four logic operations OR, XOR, AND, and NOT.

The design of an ALU is a combinational logic. Because the unit has a regular pattern, it can be broken into identical stages connected in cascade through the carries. We can design one stage of ALU and then duplicate it for the number of stages required. There are six inputs to each stage: Ai, Bi, Ci, s2, s1, and s0. There are two outputs in each stage: output Fi and the carry out Ci+1. One can formulate a truth table with 64 entries and simplify the two output function.Here we choose to employ an alternative procedure that uses the

availability of a parallel adder. The steps involved in the design of an ALU are as follows:

Design the arithmetic section independent of the logic section.Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.Modify the arithmetic circuit to obtain the required logic operations. The final ALU is shown below. The inputs to each full-adder circuit are specified by the Boolean function:

$$X_i = A_i + s_2.s_1.'s_0.'B_i + s_2.s_1.s_0'.B_i'$$
$$Y_i = s_0.B_i + s_1.B_i'$$
$$Z_i = s_2'.C_i$$

When $s_2 = 0$, three functions reduce to:

$$X_i = A_i$$
$$Y_i = s_0.B_i + s_1.B_i'$$
$$Z_i = C_i$$

Which are the functions for the arithmetic circuit of above figure. The logic operations are generated when $s_2 = 1$. For $s_2s_1s_0 = 101$ or 111, the functions reduce to:

$$X_i = A_i$$
$$Y_i = s_0.B_i + s_1.B_i'$$
$$C_i = 0$$

Output F_i is then equal to X_i (+) Y_i and produces the exclusive-OR and complement operations as specified in above table. When $s_2s_1s_0 = 110$, each A_i is ORed with B_i' to provide the AND operation as explained previously.

The 12 operations generated in the ALU are summarized in below truth table. The particular function is selected through s_2, s_1, s_0, and Cin. The arithmetic operations are identical to the ones listed for the arithmetic circuit. The value of Cin for the four logic functions has no effect on the operation of the unit and those entries are marked with don't-care X's.

Table:1 [FUNCTION TABLE OF ALU]

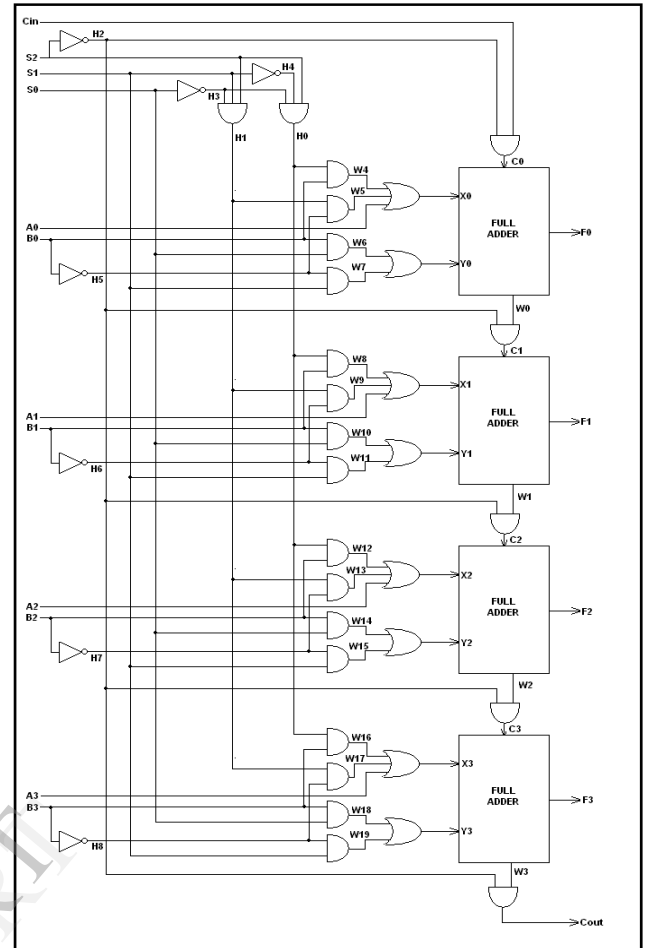| Selection				Output	Function
s2	s1	s0	cin		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A+1	Increment A
0	0	1	0	F = A+B	Addition
0	0	1	1	F= A+B+1	Add with carry
0	1	0	0	F = A-B-1	Subtract with borrow
0	1	0	1	F= A-B	Subtraction
0	1	1	0	F = A-1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	x	F = A V B	OR
1	0	1	x	F = A Θ B	XOR
1	1	0	x	F = A Λ B	AND
1	1	1	x	F = A'	Complement A

Figure:2 [LOGIC DIAGRAM OF ARITHMETIC LOGIC UNIT (ALU)]

Design of Arithmetic Unit

The basic component of the arithmetic section of an ALU is a parallel adder. A parallel adder is constructed with a number of a full-adder circuits connected in cascade. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations. Figure 3 demonstrates the arithmetic operations obtained when one set of inputs to a parallel adder is controlled externally. The number of bits in the parallel adder may be of any value. The input carry Cin goes to the full-adder circuit in the least significant bit position. The output carry Cout comes from the full-adder circuit in the most significant bit position.

The arithmetic addition is achieved when one set of inputs receives a binary number A, the other set of inputs receives a binary number B, and the input carry is maintained at 0. This is shown in (a). By making Cin = 1 as in (b), it is possible to add 1 to the sum in F. Now let us consider the effect of complementing all the bits of input B. With Cin = 0, the output produces F = A + (comp B), which is the sum of A plus the 1's complement of B.
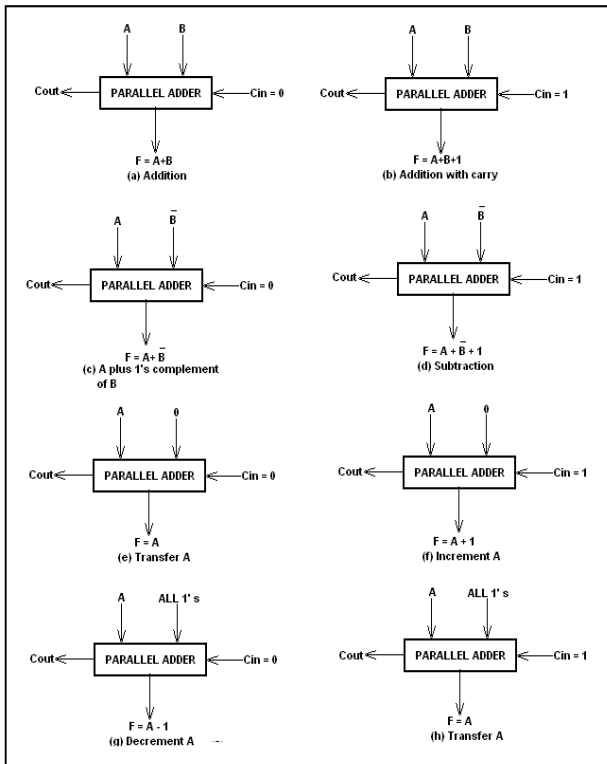
Figure:3 [Parallel Adder Arithmetic Operations]

Adding 1 to this sum by making Cin = 1, we obtain F = A+ (comp B) + 1, which produces the sum of A plus the 2's complement of B. This operation is similar to a subtraction operation if the output carry is discarded. If we force all 0's into the B terminals, we obtain F = A + 0 = A, which transfers input A into output F. Adding 1 through Cin as in (f), we obtain F = A + 1, which is the increment operation.

The condition illustrated in (g) inserts all 1's into the B terminals. This produces the decrement operation F = A – 1. To show this condition is indeed a decrement operation, consider a parallel adder with n full-adder circuits. When the output carry is 1, it represents the number 2^n because 2^n in binary consists of a 1 followed n 0's. Subtracting 1 from 2^n, we obtain $2^n – 1$, which in binary is a number of n 1's. Adding $2^n – 1$ to A, we obtain F = A + 2^n + A – 1. If the output carry 2^n is removed, we obtain F = A – 1.

Fig : True/Complement, One/Zero circuit

The circuit that controls input B to provide the functions i.e. true/complement, one/zero element. The circuit is illustrated in above diagram. The two selection lines s1 and s0 control the input of each B terminal. The diagram shows one typical input designated by Bi and an output designated by Yi. In typical application, there are n such circuits for i= 1, 2, 3……n. As shown in the table below, when both s1 and s0 are equal to 0,

the output Yi = 0, regardless of the value of Bi. When s1s0 = 01, the top AND gate generates the value of Bi while the bottom gate output is 0; so Yi = Bi. With s1s0 = 10, the bottom AND gate generates the complement of Bi to give Yi = Bi`. When s1s0 = 11, both gates are active and Yi = Bi + Bi` = 1.

S1	S0	Y1
0	0	0
0	1	Bi
1	0	Bi'
1	1	1

A 4-bit arithmetic circuit that performs eight arithmetic operations is shown in figure below. The four full-adder (FA) circuits constitute the parallel adder. The carry into the first stage is the input carry .The carry out of the forth stage is the output carry. All other carries Are connected internally from one stage to the next. The selection variables are S1, S0 and Cin. Variables S1 and S0 control all of the B inputs to the full-adder circuits as in figure. The A inputs go directly to the other inputs of the full-adders.
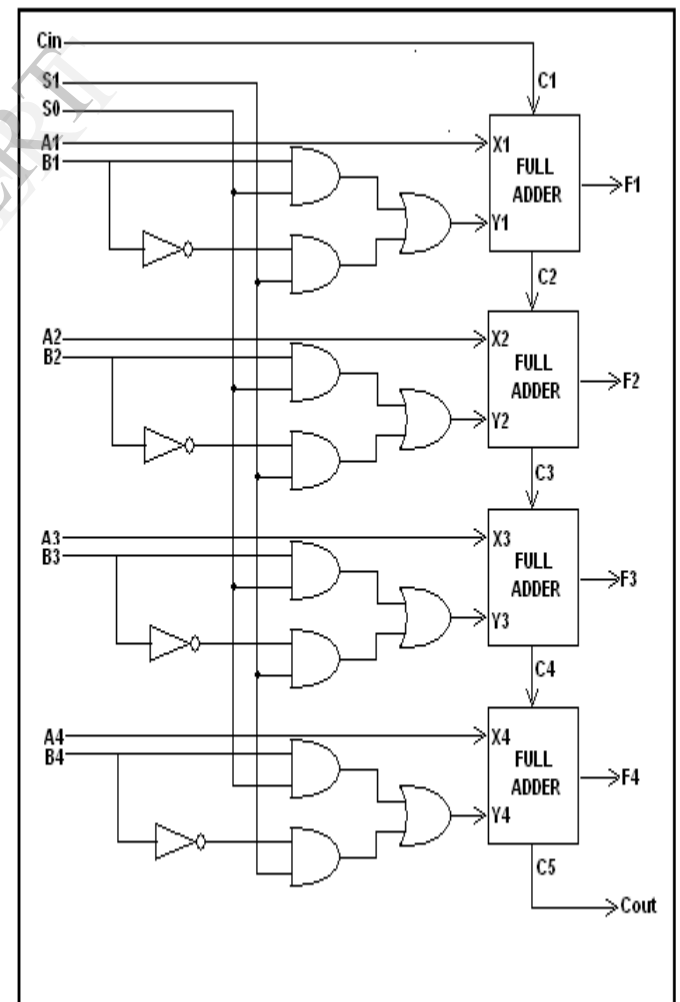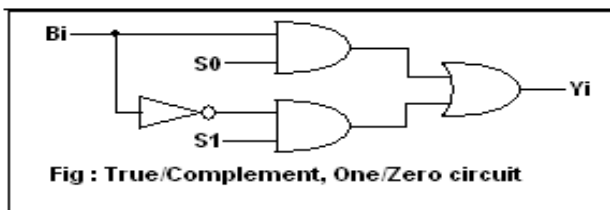

Figure:4.a   [Logic Diagram Of Arithmetic Circuit]

The arithmetic operations implemented in the arithmetic circuit are listed in below table 2.

The values of the Y inputs to the full-adder circuits are a function of selection variables S1 and S0.

Table:2 [Function Table Of Arithmetic Unit]

Function select			Y	Output	Function
s2	s1	s0			
0	0	0	0	F = A	Transfer A
0	0	1	0	F = A+1	Increment A
0	1	0	B	F = A+B	Addition
0	1	1	B	F = A+B+1	Add with carry
1	0	0	B'	F = A-B-1	Subtract with borrow
1	0	1	B'	F= A-B	Subtraction
1	1	0	All 1's	F = A-1	Decrement A
1	1	1	All 1's	F = A	Transfer A

Adding the value of Y in each case to the value of A plus the Cin value gives the arithmetic operation in each entry. The eight operations listed in the table follow directly from the function diagrams illustrated.

This example demonstrates the feasibility of constructing an arithmetic circuit by means of a parallel adder. The combinational circuit that must be inserted in each stage between the external inputs Ai and Bi and the inputs of the parallel adder Xi and Yi is a function of the arithmetic operations that are to be implemented. The arithmetic circuit of fig needs a combinational circuit in each stage specified by the Boolean functions:

$$Xi = Ai$$

$$Yi = Bi \, S0 + Bi'S1 \qquad i = 1,2,\ldots, n$$

Where n is the number of bits in the arithmetic circuit. In each stage i, we use the same common selection variables S1 and S0. The combinational circuit will be different if the circuit generates different arithmetic operations.

A. Design of Logic Unit

The logic micro operations manipulate the bits of the operands separately and treat each bit as a binary variable. There are 16 operations that can be performed with two binary variables. The 16 logic operations can be generated in one circuit and selected by means of four selection lines. Since all logic operations can be obtained by means of AND, OR, and NOT (complement) operations, it may be more convenient to employ a logic circuit with just these operations. For three operations, we need two selection variables. But two selection lines can select among four logic operations, so we choose also the exclusive-OR (XOR) function for the logic circuit to be designed in this part. The diagram shown below shows a typical stage designated by subscript i. The four gates generate the four logic operations OR, XOR, AND, and NOT. The two

selection variables in the multiplexer select one of the gates for the output. The table above shows output logic generated as a function of the two selection variables.
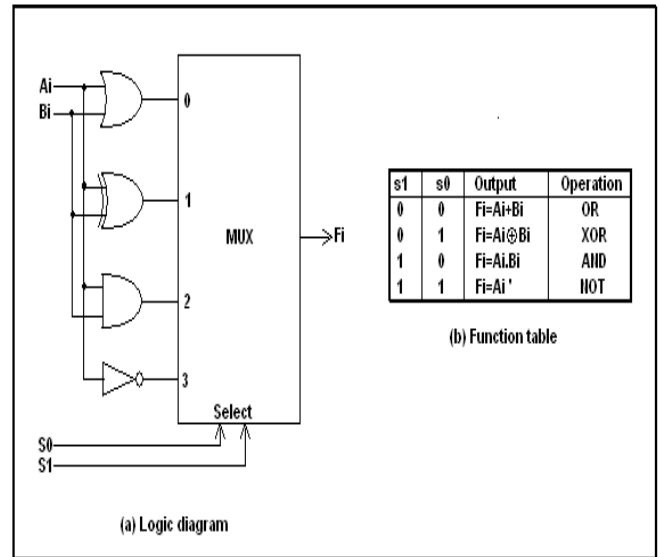

Figure:4.b [Logic Diagram Of Logic Circuit]

The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit. Selection variables s1 and s0 can be made common to both sections provided we use a third selection variable, s2, to differentiate between the two.

This configuration is shown below. The outputs of the logic and arithmetic circuits in each stage goes through a multiplexer with selection variable s2. When s2 = 0, the arithmetic output is selected, but when s2 = 1, the logic output is selected.
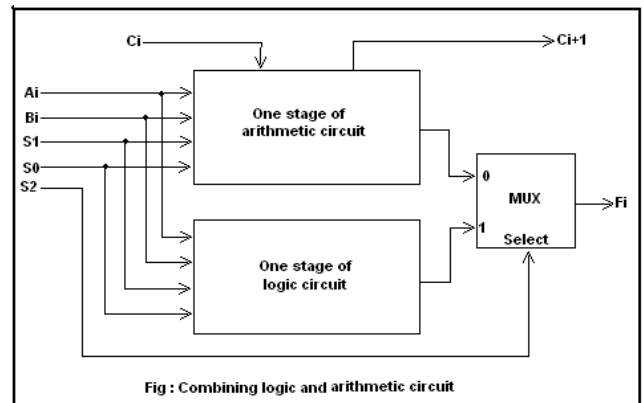

Figure:4.c   [Combining logic and arithmetic circuit]

A more efficient ALU can be obtained if we investigate the possibility of generating logic operations in an already available arithmetic circuit. This can be done by inhibiting all input carries into the full-adder circuits of the parallel adder. Let we consider the Boolean function that generates the output sum in a full-adder circuit:

$$Fi = Xi \oplus Yi \oplus Ci$$

The input carry Ci in each stage can be made to be equal to 0 when a selection variable s2 is equal to 1. The result would be:

$$Fi = Xi \oplus Yi$$

This expression is valid because of the property of the exclusive-OR operation X (+) 0 = X. Thus, with the input carry to each stage equal to 0, the full-adder circuits generate the exclusive-OR operation. Now consider the arithmetic circuit, the value of Y_i can be selected by means of the two selection variables to be equal to either 0, B_i, B_i', or 1. The value of X_i is always equal to input A_i. The below table shows the four logic operations obtained when a third selection variable $s2 = 1$. This selection variable forces C_i to be equal to 0 while $s1$ and $s0$ choose a particular value for Y_i. The four logic operations obtained by this configuration are transfer, exclusive-OR, equivalence, and complement. The third entry is the equivalence operation because:

$$Ai (+) Bi' = AiBi + Ai'Bi' = Ai (.) Bi$$

The last entry in the table is the NOT or complement operation because:

$$Ai (+) 1 = Ai'$$

III. RESULT

It is needed to test whether the design works to meet the given specification to ensure that designed entity is correct. This is verified by the process of simulation. The process of simulation uses a test bench to test the design whether it behaves correctly by stimulating it with artificial input and monitoring the output. The simulation is carried out by using the Xilinx ISE 9.2i tool l and having the test bench and the behavioral design code for 4-bit ALU in the same project folder. We observed from the simulation results that the 4-bit ALU implemented by the above described method and code, worked successfully for all the input combinations and the select codes according to the given specification in Table 1.

The simulation results and RTL Schematics are shown below in Fig.7 and Fig .8 respectively
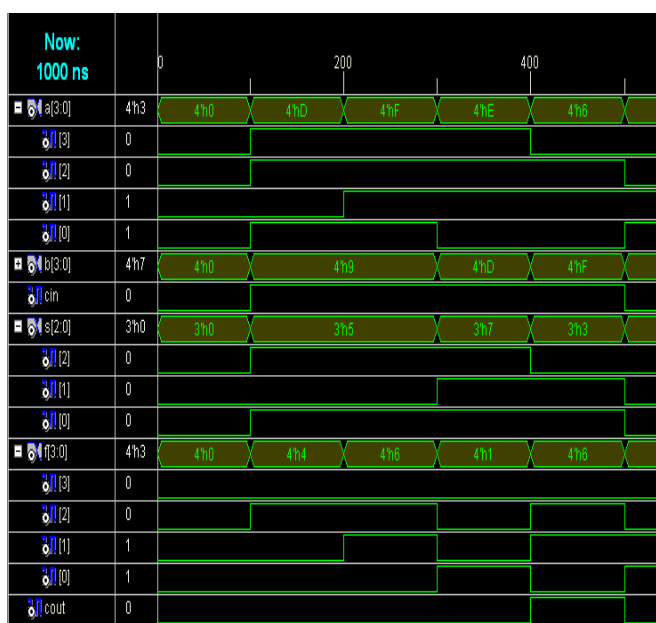
A. Simulation Results

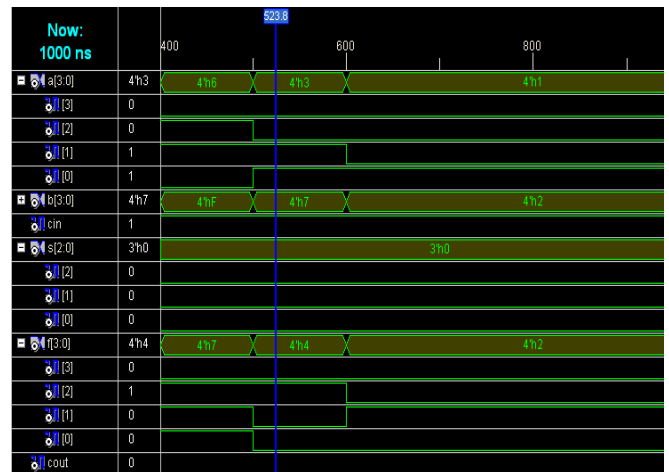Figure:7. b (S2 S1 S0=000, Cin=1 :Increment A i.e. A + 1)

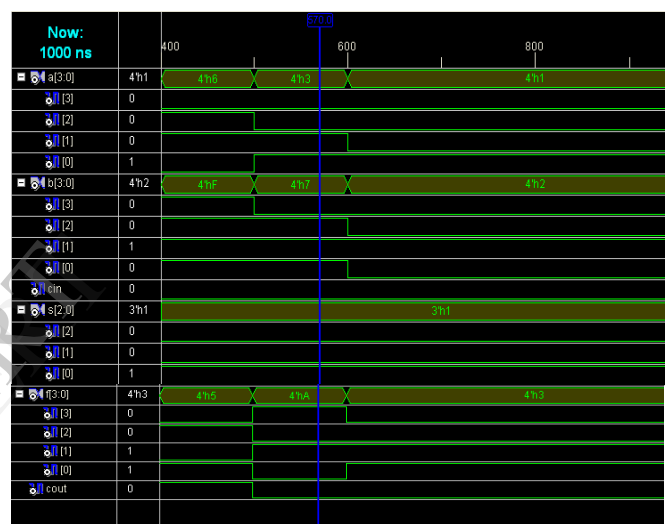
Figure:7. c   (S2 S1 S0=001, Cin=0 : A + B)

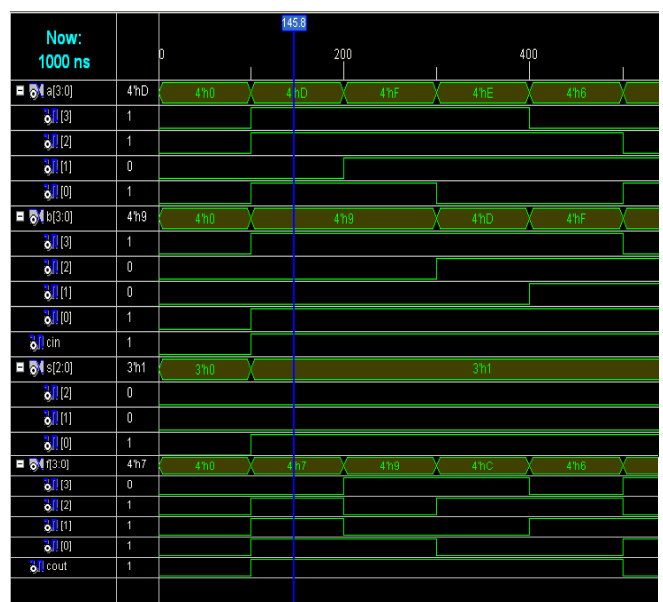Figure:7. a   (S2 S1 S0=000, Cin=0 : Transfer A)

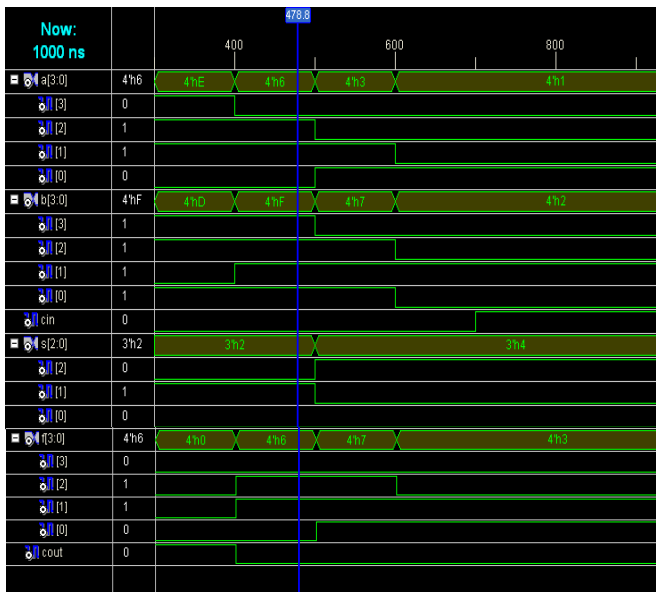Figure:7. d   (S2 S1 S0=001, Cin=1 : A + B+1)

Figure:7. e (S2 S1 S0=010, Cin=0 : A – B–1)
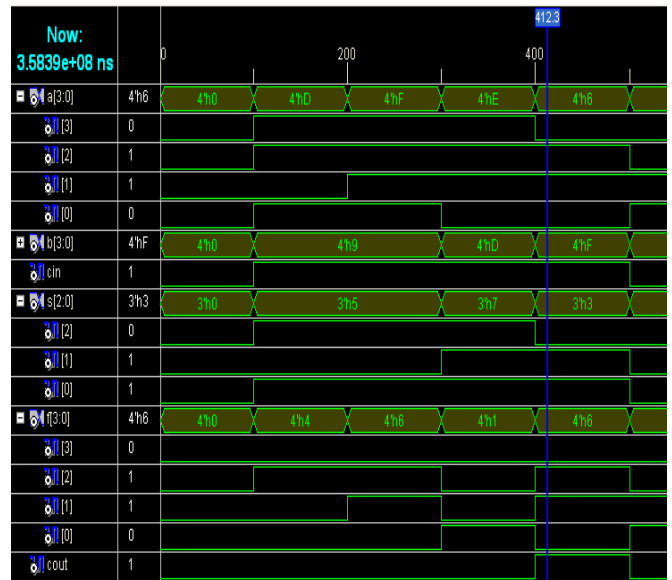

Figure:7.h (S2 S1 S0=011, Cin=1 : Transfer A)

Figure:7. f (S2 S1 S0=010, Cin=1 : A - B)
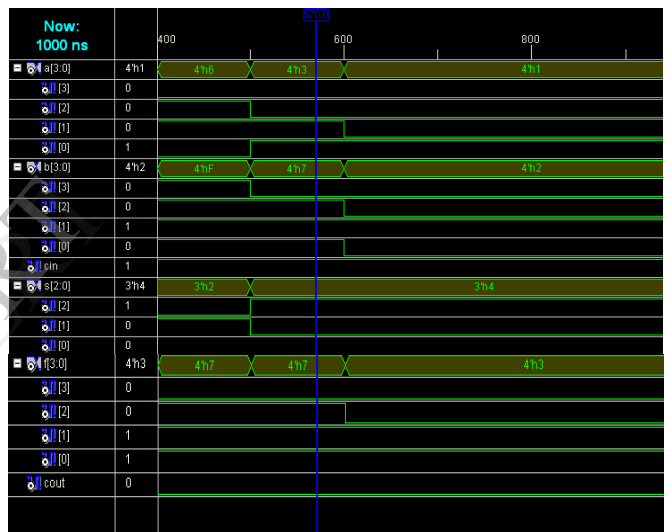

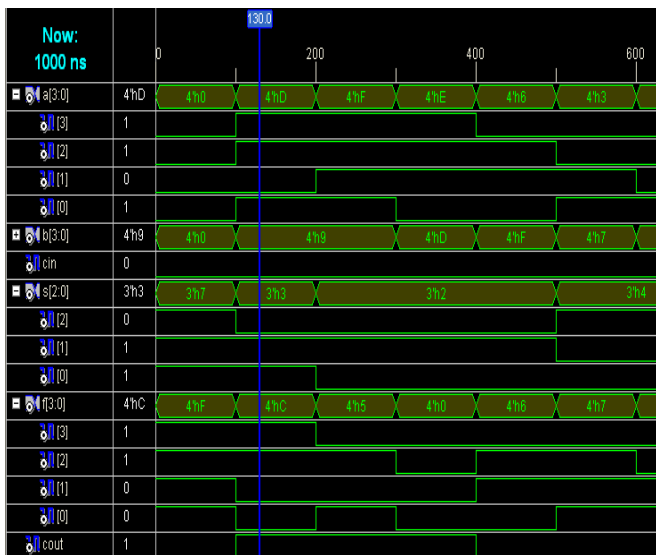Figure:7. i   (S2 S1 S0=100, Cin=x : A OR B)

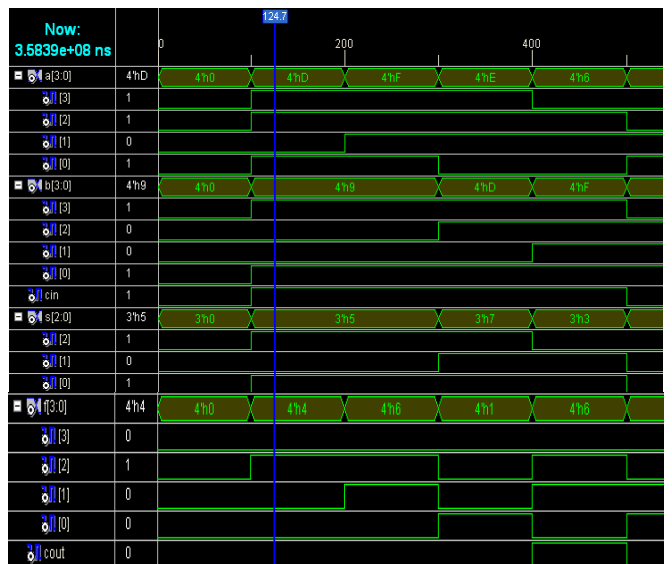Figure:7.g   (S2 S1 S0=011, Cin=0 : Decrement A i.e. A – 1)
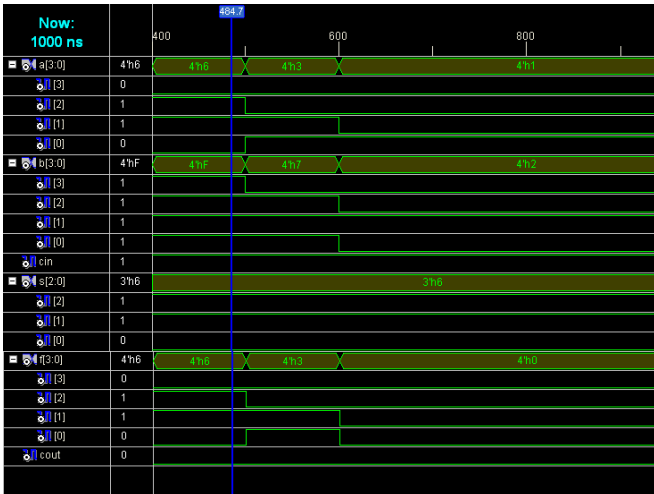

Figure:7. j   (S2 S1 S0=101, Cin=x : A XOR B)
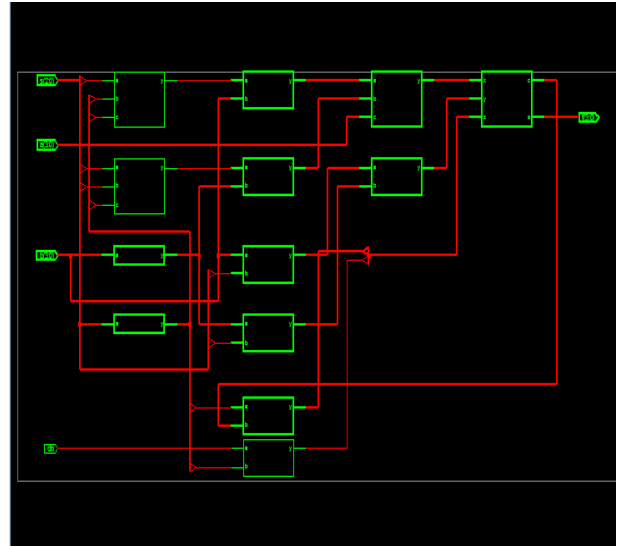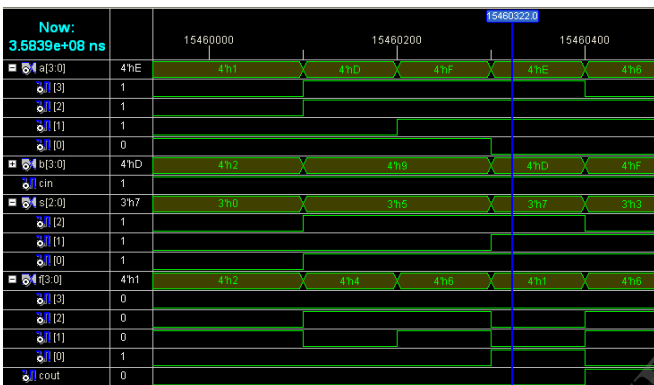
Figure:7. k (S2 S1 S0=110, Cin=x : A AND B)

Figure:7. l (S2 S1 S0=111, Cin=x : Complement of A)

B. *RTL SCEMATIC OF ALU:*

Figure:8.a
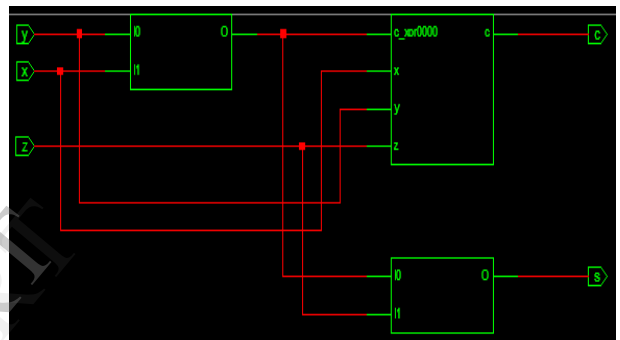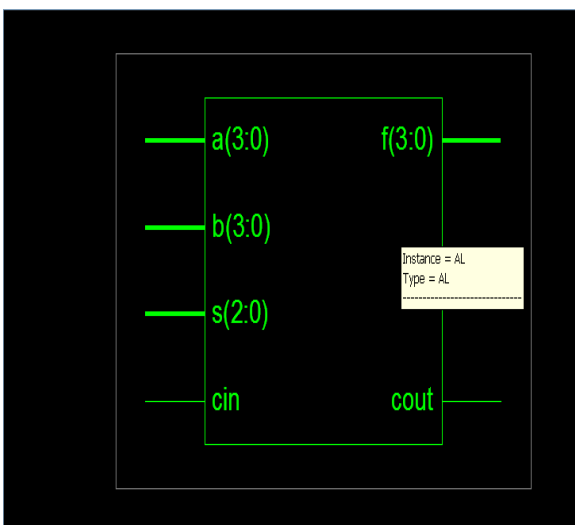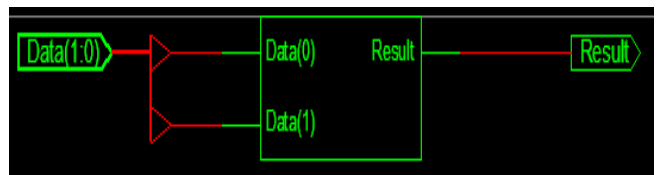

Figure:8.b

Figure:8. c

Figure:8. d

IV.   CONCLUSION & FUTURE WORKS

This study helped to understand the complete flow of RTL design, starting from designing a top level RTL module for 4-bit ALU using hardware description language, VHDL. Verification of the designed RTL code using simulation techniques, synthesis of RTL code to obtain gate level net list using Xilinx ISE tool and Arithmetic Logic Unit was successfully designed and simulated using Xilinx ISE 9.2i package. The ALU model is designed in such a manner that the whole module can be used as component to design 8,16,32 bit ALU using structural model.

REFERENCES

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

[2] http://www.forteds.com/behavioralsynthesis/index.asp
Douglas L. Perry, VHDL, third edition, McGraw-Hill, pp.60-63, 238, July 1999.

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

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

[5] B.Stephen Brown, V.Zvonko, "Fundamentals of digital logic with VHDL Design" 2nd Edition , Mc Graw Hill International Edition, 2005.

[6] Charles H.Roth, Jr., "Digital System Design using VHDL", PWS Publishing Company, 2006.

[7] Mark Zwolinski, "Digital System Design with VHDL", Prentice Hall, 2000.Pedroni, "Digital Logic Design using VHDL".

[8] S.Kaliamurthy, R.Muralidharan, "VHDL Design of FPGA Arithmetic Processor" International Conference on Engineering and ICT, 2007.

[9] Xilinx Technologies, Xilinx Data Sheet for XC3S100E. http:// direct. xilinx.com/bvdocs/ publications/ ds312.pdf.

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

[11] Prof. S. Kaliamurthy & Ms. U. Sowmmiya,"VHDL design of arithmetic processor" ,Global Journals Inc.(USA) , November 2011.

[12] Geetanjali and Nishant Tripathi "VHDL Implementation of 32-Bit Arithmetic Logic Unit (Alu)"

[13] Shikha Khurana, Kanika Kaur "Implementation of ALU using FPGA"

[14] Mr. Abhishek Gupta , Mr. Utsav Malviya , Prof. Vinod Kapse "A Novel Approach to Design High Speed Arithmetic Logic Unit Based On Ancient Vedic Multiplication Technique" International Journal of Modern Engineering Research (IJMER) www.ijmer.com. Vol.2, Issue.4, July-Aug 2012 pp-2695-2698. ISSN: 2249-6645 www.ijmer.com