International Journal of Emerging Engineering Research and Technology
Volume 2, Issue 4, July 2014, PP 113-122
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Simulation of Advance RISC Machine Unit Using VHDL

Dipali P. Shende, Subha Subramaniam, Vidya Gogate

Shah and Anchor kutchhi Engg.college. Chembur
dipalipshende@gmail.com
subhasubramaniam@yahoo.co.in
vidyagog68@gmail.com

Abstract: *In this work we have virtually simulated 32 bit integrated arithmetic functional units using VHDL.32bit ALU is simulated which can perform arithmetic operations and logical operations. The purpose of using ALU for short but yet frequent computational operations is to speed up the computer processor and increase efficiency.*

We have simulated 32 bit MAC unit (multiply and accumulate unit) using VHDL. Booth's multiplication Algorithm is used to perform multiplication operations. As multiplication of two' n bit number' yields '2n bit product' the result is 64 bits. We have designed and simulated 64 bit ripple carry adder and 64 bit parallel in parallel out shift register .

Restore division algorithm is implemented in VHDL to perform division operations. Barrel shifter which is the key preprocessor of ARM unit is also simulated and results are shown using VHDL. It is observed that the results obtained using Xilinx modelsim 6.3f version is matched with the theoretical and logical values.

Keywords: *32 bit ALU, mac unit VHDL, Simulation,*

INTRODUCTION

Audio and video embedded systems applications are increasingly in demand. They require digital signal processing (DSP) capability that until recently would have been provided by a separate DSP processor. Now, however, the ARM architecture offers higher memory bandwidths and faster multiply accumulates operations, permitting a single ARM core design to support these applications.

ARM cores are widely used in mobile phones, handheld organizers, and a multitude of other everyday portable consumer Devices. Over one billion ARM processors had been shipped worldwide by the end of 2001.

The ARM processor has a specialized block called the MAC unit (Multiply and Accumulate Unit) that will multiply the 2 input operands and add the result to a register that has stored all previous results. The General Purpose Processors would not have multiply instruction ready for use. And even if it were present, would take lot of cycles to complete a single multiplication. Thus arises the need to multiply and add the product to the previous result as fast possible, ideally single clock. Figure 1.1 shows the Advance RISC processor unit. . Basically, computers handle lots of numbers based on the three basic arithmetic operations of addition, multiplication and division.In this work to enhance the performance of 32 bit advanced processor Arithmetic and Logic unit, multiply and Accumulate unit, Ripple carry adder, parallel in parallel out shift register and barrel shifter units are configured using Xilinx ISE VHDL.

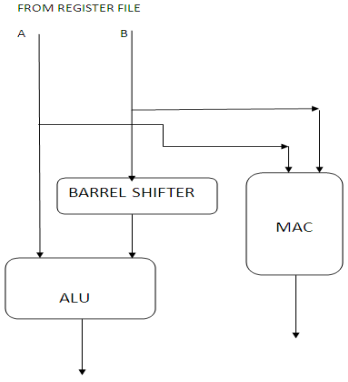


Fig.1.1. *Advanced RISC processor unit.*

Arithmetic logic unit is fundamental building block of a central processing unit.. The purpose of using ALU for handling short but yet frequent computational operations is to speed up computer processor and increase efficiency. The ALU has two 32-bits inputs. The first comes from the register file while the other comes from the barrel shifter. A barrel shifter is a combinational logic circuit with n data input n data outputs. The barrel shifter has a 32-bit input to be shifted.ALU outputs modify the status register flags. The Cout goes to the C flag. While the most significant bit actually represents the S flag

Block Diagram of 32 Bit ALU

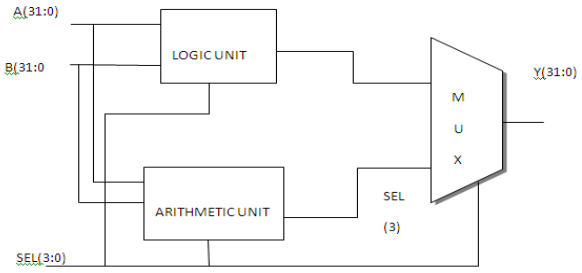


Fig.1.1. *Block diagram of ALU*

Since the ALU operates on 32 bit operands, it is called 32 bit ALU. Zero output indicates if all result lines have '0' value. Carrey out indicates Carrey out and unsigned integer overflow, Our ALU is used to perform functions like logical AND function logical OR function, logical NOR function, logical XOR function, Arithmetic add function, Arithmetic subtracts function, increment, and decrement function.

Functioning of 32 Bit ALU
Table1

sel	function
0000	transfer a
0001	increment a
0010	decrement a
0011	transfer b
0100	increment b
0101	decrement b
0110	a+b
0111	a-b
1000	not of a
1001	not of b
1010	a and b
1011	a or b
1100	a nand b
1101	a nor b
1110	a xor b
1111	not(a xor b)

An ALU must process numbers using the same format as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement, two's Complement sign-magnitude format, and even true decimal systems, with ten tubes per digit. ALUs for each one of these numeric systems had different designs, and that influenced the current preference for two's complement, as this is the representation that makes it easier for the ALUs to calculate additions and subtractions. Figure 1.1 represents the general schematic of an ALU.

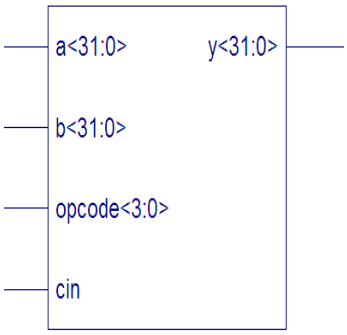
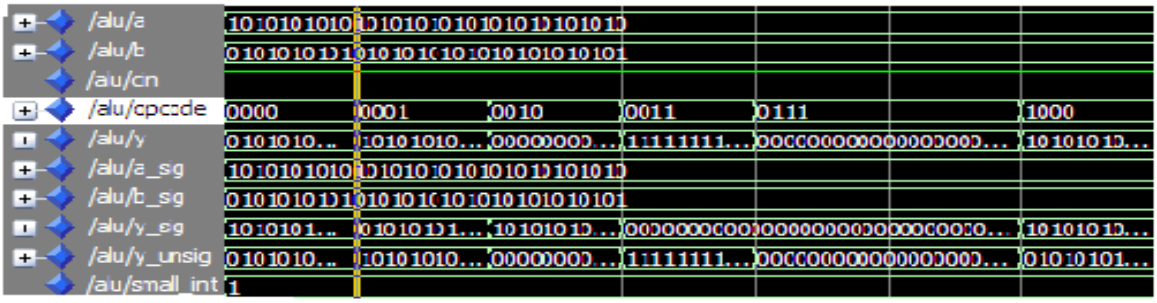


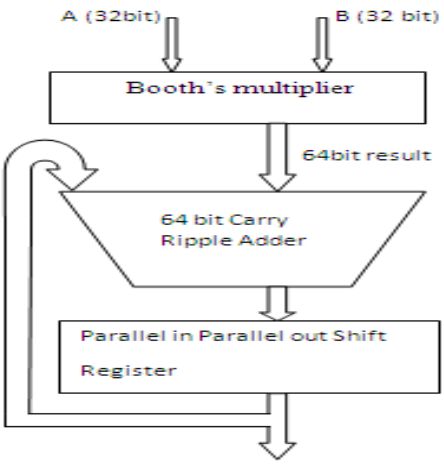
Fig.1.2. *Entity of ALU*

Waveform for ALU operations:



Multiply and Accumulate Unit

The arm processor has a specialized block called the MAC unit (Multiply and Accumulate Unit).That will multiply the 2 input operands and add the result to a register that has stored all previous results.



The Architecture of Mac unit is shown in figure 4.1 which have MAC Unit consisting of 32 bit Booth's Multiplier it handles both signed and unsigned integers uniformly Booth encoding is a method used for the reduction of the number of partial products proposed by A.D.Booth in 1950.The

64 bit result of multiplier is provided to the input of Accumulator through Carry Ripple adder .The product of Ai X Bi is always fed back into the 64 bit RCA in accumulator and added again with the next product Ai X Bi.

Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed –2's complement representation

I will illustrate the booth algorithm with the following example: - $(2)_{10}$ x $(-4)_{10=}$ $(0010)_2$ *$(1100)_2$ Making the Booth table from the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.

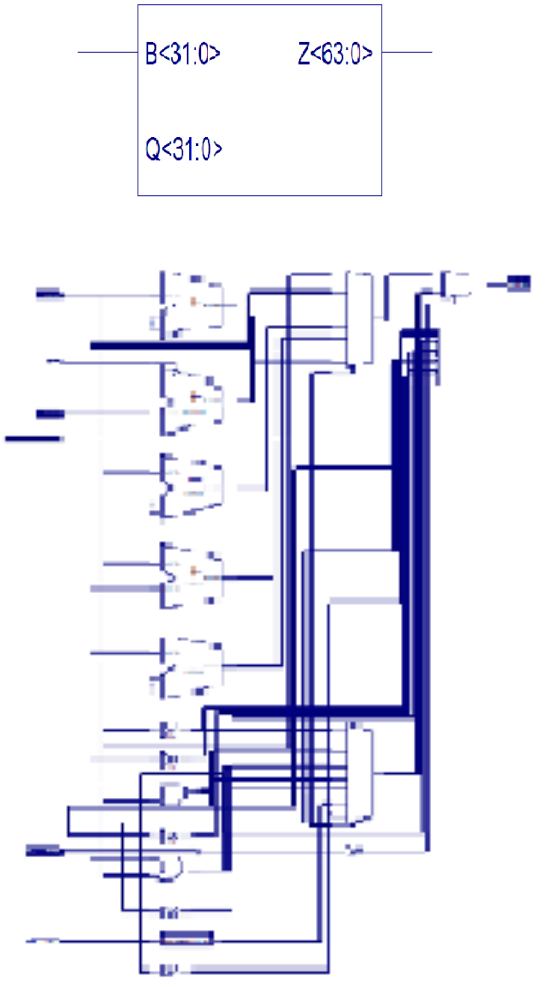


Fig.1.3. *RTL Schematics of ALU.*

Fig .2.3 Entity of Booth's Algorithm

1) i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change ,so there are two changes on this one 1100 -- From 1 to 1 no change, 1 to 0 one change,0 to 0 no change, so there is only one change on this one. Therefore, multiplication of 2 x (– 4), where 2_{ten} $(0010)_{two}$ is the multiplicand and $(-4)_{10=}$ $(1100)_2$ is the multiplier. Let X = 1100 (multiplier) Let Y = 0010 (multiplicand) Take the 2's complement of Y and call it –Y = 1110
2) Load the X value in the table.
3) Load 0 for X-1 value it should be the Previous first least significant bit of x
4) Load 0 in U and V rows which will have the Product of X and Y at the end of operation
5) Make four rows for each cycle; this is because we are multiplying four bits numbers.

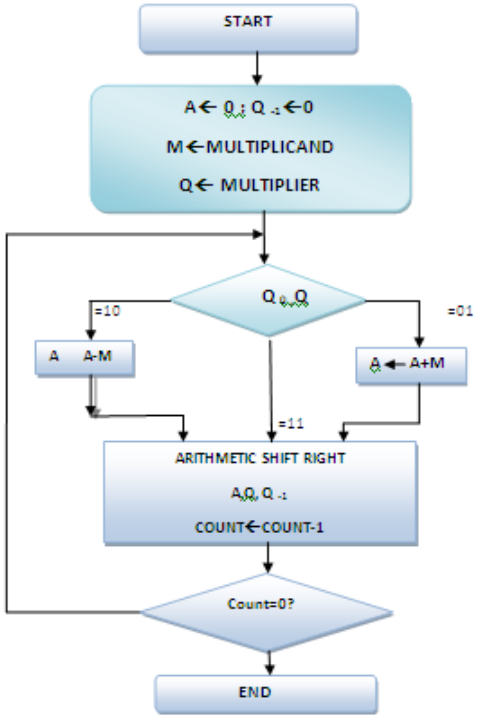


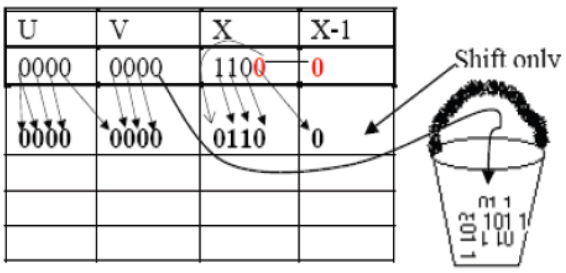
Fig.2.2. *flowchart of Booth's Algorithm*

U	V	X	X-1	
0000	0000	1100	0	Load the value
				1st cycle
				2nd cycle
				3rd Cycle
				4th Cycle

Booth algorithm requires examination of the multiplier bits, & shifting the partial product.

- Prior to the shifting, the multiplicand may be added to or subtracted from the partial product, or left unchanged according to the following rules:
- Look at the first least significant bits of the multiplier "X", and the previous least
- Significant bits of the multiplier "X - 1".
- 0 0 Shift only
- 1 1 Shift only.
- 0 1 Add Y to U, and shift
- 1 0 Subtract Y from U, and shift or add (-Y) \ to U and shift

7) Shift X circular right shift because this will prevent us from using two registers for the X Value



U	V	X	X-1	
0000	0000	1100	0	
0000	0000	0110	0	
0000	**0000**	**0011**	**0**	← Shift only

U	V	X	X-1	
0000	0000	1100	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	**0000**	**0011**	**0**	← Add –Y
1111	**0000**	**1001**	**1**	← Shift

U	V	X	X-1	
0000	0000	*1100*	0	
0000	0000	0110	0	
0000	0000	0011	0	
1110	0000	0011	0	
1111	0000	1001	1	
1111	**1000**	***1100***	**1**	← Shift only

Simulation result of 32 bit Booths Algorithm:

With reference to the table 4.1 shown above we can find outComputation of s and cout from truth table.Where s refers to sum and cout refers to carry out.

Computation of s and cout from truth table:S =a xor b xorcin,

Cout=(a and b)or (a and cin)or(b and cin) .

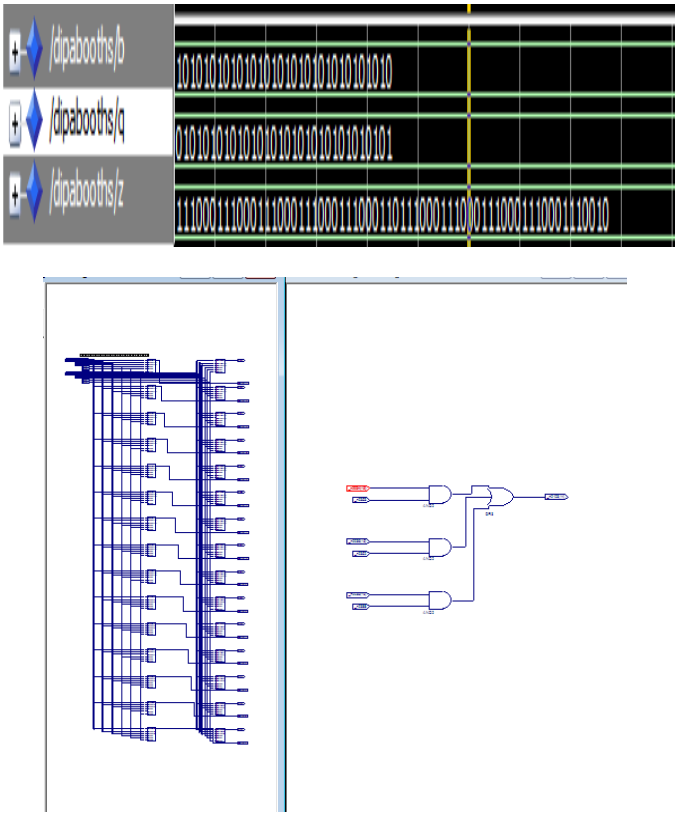


Fig.:2.3. *RTL of Boooth's Algorithm*

Ripple Carry Adder

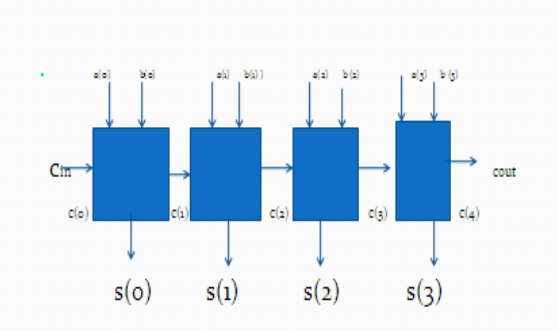
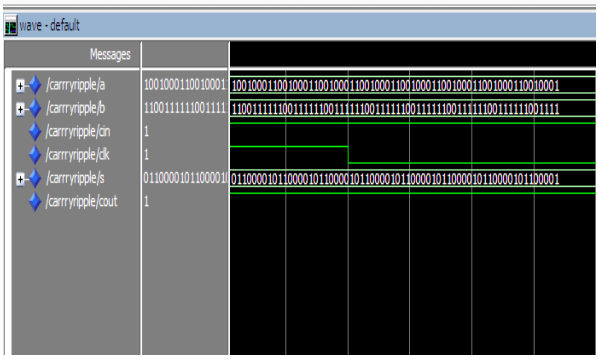


Figure shows 4.5 shows a 4 bit carry ripple adder. For each bit a full adder unit is employed. The truth table of the full adder unit is also shown. where in a and b represents the input bits, cin is carry in bit, and cout is the carry out bit s must be high where ever the number of inputs that are high is odd parity function. While cout must be high when two or more inputs are high.

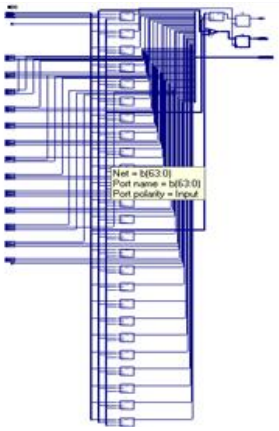
a	b	cin	s	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



Result of Ripple Carry Adder

Simulation Result of Ripple carry adder is shown in figure 4.6 for cin equal to one. The expected result is obtained in's'.

Figure no 4.7 shows the RTL schematic of ripple carry adder



Parallel in Parallel out shift register

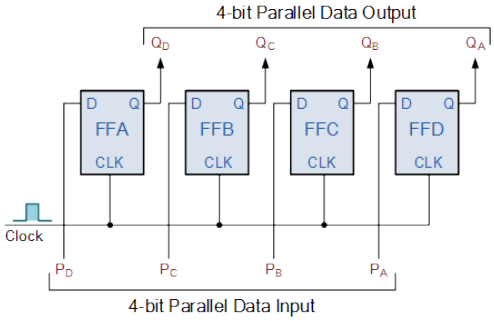
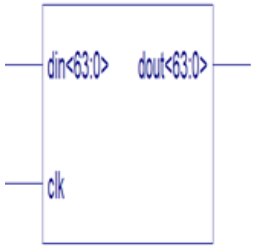


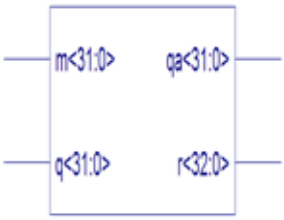
Figure no. 4.8: *4 bit parallel in parallel out shift register*

[http://www.electronics-tutorials.ws/sequential/seq_5.html**]**

The Figure 4.9 shows the entity of Shift Register here din is the input dout is the output of shift register this is a type of sequential logic circuit that is used for the storage or transfer of data in the form of binary numbers. The data is presented in a parallel format to the parallel input pins P_A to P_D and then transferred together directly to their respective output pins Q_A to Q_A by the same clock pulse. Then one clock pulse loads and unloads the register. This arrangement for parallel loading and unloading is shown in the figure 4.8

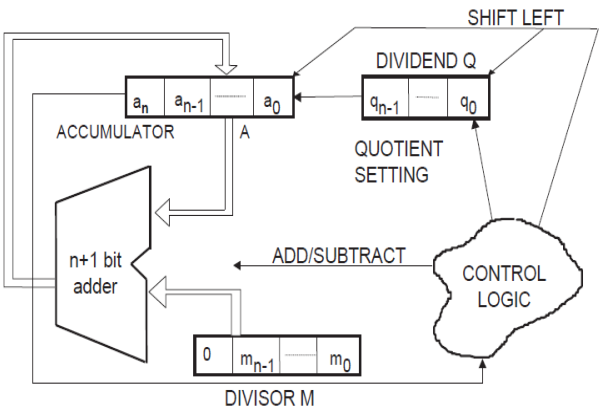


Restoring Division Operation



Entity of restore division Algorithm

Figure shows the hardware for implementation of division process. It consists of N+1 binary adder ,shift, add, and Subtract control logic and registers A,M ,Divisor and dividend are loaded into register A,M,Q respectively. Register A is set to zero

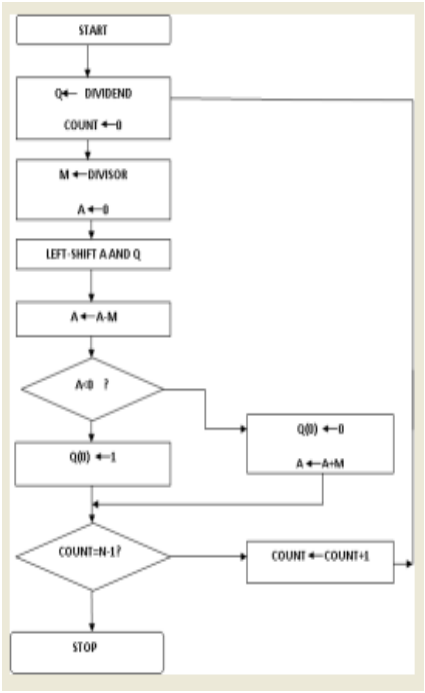


Algorithm for Restoring Division

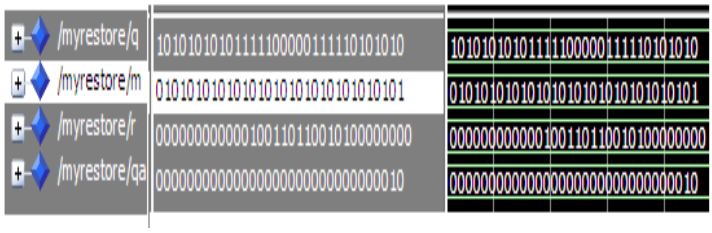
1. Shift A and Q left one binary position.

2. Subtract divisor from A and place answer in A
3. If sign bit of A is 1 set Q_0 to 0 and add divisor back to A, Otherwise ,set Q_0 to 1
4. Repeat step 1,2,3 n times.

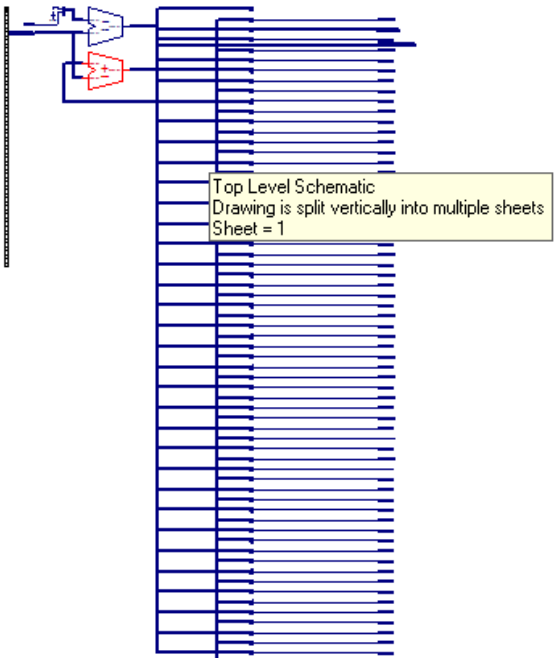
Flow chart of restore Division Algorithm



Simulation result of Restore division Algorithm



RTL schematics of Restore division Algorithm



CONCLUSIONS

32 bit ALU is designed and synthesized using xilinx ise 9.1i ALU is a major component of the CPU(central processing unit.It performs arithmetic computations such as addition subtraction and all basic logical operations(AND,OR,NOT,NAND,XOR,XNOR,).We have verified the results obtained from xillinx ISE Design Suit v9.1i with the therotical results for all operations that we performed and found that they match with the theoretical values.In addition to this Booth's multiplication and restore division algorithm are implementated to increase the efficiency of ALU.

REFERENCES

[1] J. R. Allen, et al, "IBM Power NP network processor: Hardware, software, and applications," in IBM Journal of Research & Development, Vol. 47, No. 2/3March/May 2003.

[2] Xiaoning Nie, et al, "A New Network Processor Architecture for High-speed Communications," in IEEE Workshop on Signal Processing Systems, 1999.

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

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

[5] Hwang, Enoch, "Digital Logic and Microprocessor Design with VHDL", Thomson, 2006.

[6] Stallings, William. "Computer Organization & Architecture: Designing for performance" 7th ed. Pearson Prentice Hall, 2006.

[7] H. Guzman-Miranda, L. Sterpone, M. Violante, M. A. Aguirre Coping with the Obsolescence of Safety- or Mission-Critical Embedded Systems using FPGAs IEEE Transactions on Industrial Electronics, Vol. 58,