International Conference on Smart Electronic Systems - 2016
KLE Society's Dr M S Sheshgiri College of Engineering & Technology, Belagavi

International Journal of Technology and Science, ISSN (Online) 2350-1111, (Print) 2350-1103 Volume 3, Issue 4, 2016 pp. 45-49

Design And Simulation Of Improved Speed 8 Bit Arithmetic And Logical Unit(ALU) Using Efficient Algorithms

[1] Amruta Wadone , [2] Aditya Desai , [3] Akshata Emmi , [4] Gangotri Gudimani ,
[5] Prof. Nagaraj Vannal , [6] Prof. Gireesha H M
[1,2,3,4,5,6] Department of Instrumentation Technology, B V B College Of Engineering & Technology Hubli, Karnataka,India
Email: [1]amrutaw96@gmail.com , [2]adityadesai453@gmail.com , [3]abemmi45@gmail.com , [4]gangotrigudimani95@gmail.com
[5]nagaraj.vannal@gmail.com , [6]gireesh_hm@gmail.com

Abstract— **The cumulative demand in refining the performance of processors for implementing complex and challenging processes has resulted in the incorporation of a number of processor cores into single chip. But the load on a processor is not less in a system. This load is condensed by adding the main processor with co-processors, which are designed to work upon precise type of functions like signal processing, graphics, numeric computation etc. in which the Arithmetic Unit is most dominant Co-Processor which perform arithmetic operation like addition ,multiplication ,subtraction and division. Quicker Operations are of extreme significance in Arithmetic Unit. The quickness of ALU depends on the logical execution of operational circuit. It is conceivable to increase the performance of ALU by using well-organized algorithms and techniques, thus total enactment of an arithmetic and logic unit can be enriched. In this paper we have described about the 8 bit ALU which performs basic logical operations and arithmetic operations such as addition, subtraction and multiplication. We are providing a choice to the user to select the operation to be performed. Arithmetic operations are carried out by making use of efficient algorithms (such as carry look ahead adder). Verilog code is written for all the operations. Selection of operation is based on select lines. Select line is of 3 bits (so that $2^{\wedge 3}$ =8 operation can be performed).**

I. INTRODUCTION

The design of a well-organized ALU includes many features. Depending on the performance factors that need to be improved, several methods can be developed.

The inputs to an ALU are the data to be operated on, called operands, and a code representing the operation to be achieved; the ALU's output is the outcome of the accomplished operation.3 select lines are provided to select operations. Say for ex: 000 for addition, 001 for subtraction, 010 for multiplication.

In algorithmic and organizational stages, many methods [4] can be used to boost the proficiency of the multiplier, adder and subtractor. The practices for improving performance of multiplier comprise decreasing the fractional products and the approaches of their addition but the principle behind multiplication remains the alike in all cases.

The usage of two's-complement method has benefits over the current one's-complement systems. The two's-complement method [4] creates the system both simpler to implement and skillful of easily managing higher accuracy arithmetic. This method is the most common way of expressing signed integers on computers. An *N*-bit two's-complement numeral system can denote every integer the range $-(2^{N-1})$ to$+(2^{N-1}-1)$ while one's-complement can only represent integers in the range $-(2^{N-1}-1)$ to $+(2^{N-1}-1)$.

Booth's multiplication algorithm [5] can be used to implement a distinct multiplier. Booth's multiplication is a multiplication algorithm that multiplies two signed binary numbers in two's-complement representation.

There are two 8 bit input namely A and B and the result is 8 bit as shown in the Fig 1. Fig 2 represents architecture of ALU. There are 7 operations to be performed namely addition, subtraction, multiplication, and logical operations like AND, OR, XOR and NOT. There are three select lines based on which we can select the required operations. We have written separate code to execute the operations.

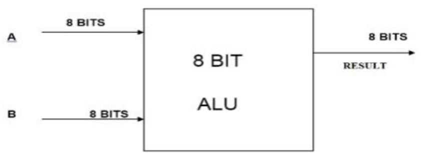


Fig 1.Block diagram of ALU

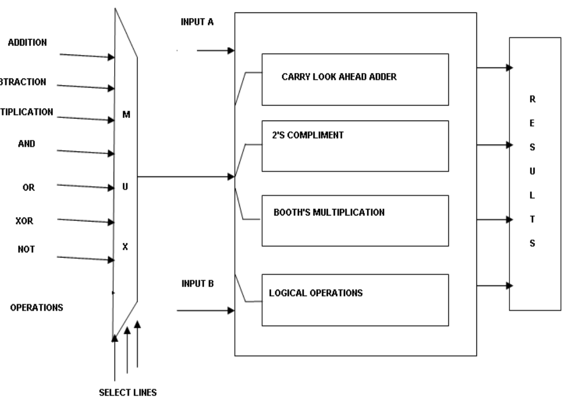


Fig 2 Architecture of ALU

III DESIGN OF ALU (METHODOLOGY)

Many techniques of Ancient Mathematics can be used to develop an algorithm for performing the functions faster and implement the same.The paper includes making use of appropriate algorithms after comparing them. Selecting efficient logical implementation, neat organization, and designing control logic, the ALUs performance can be improved.This is a situation of transparent box wherein we have the components which are placed in the black box. The components used are as follows:

1. Carry look ahead adder: It is used for addition of two numbers.
2. 2's complement: It is used for subtraction of two numbers.
3. Booths multiplication: It is used for multiplication of two numbers.
4. Logical operations: following operations like AND, OR, XOR, XNOR& NOT can be performed.

MUX is used to select the operations i.e addition, multiplication, subtraction, AND, OR, XOR, XNOR& NOT.

Explanation of each block in ALU in given below

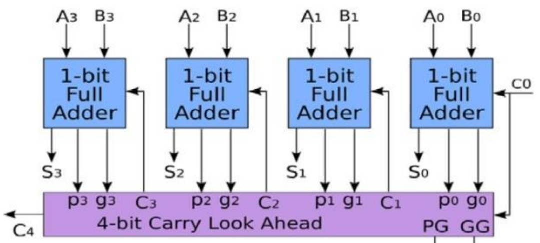
1) Carry look ahead adder:

A carry-look ahead adder (CLA) is a kind of adder used in digital logic. Fig 3.a indicates the RTL schematic of carry look ahead adder. A carry-look ahead adder enhances the speed by reducing the amount of time required to decide carry bits. It can be compared with the easiest, but usually slower, ripple carry adder in which the carry bit is computed beside the sum bit, and each bit must stay until the preceding carry has been computed to begin calculating its own result and carry bits [6]. The carry-look ahead adder calculates one or more carry bits before the sum, [7] which decreases the delay time to calculate the result of the bigger value bits. Carry look ahead depends on two things:

Calculating, for every digit position, whether that position is going to transmit a carry if one comes in from the right.

Combining these calculated values to be able to deduce rapidly whether, for each group of digits, that group is going to circulate a carry that comes in from the right.

Figure 3: Carry look ahead adder



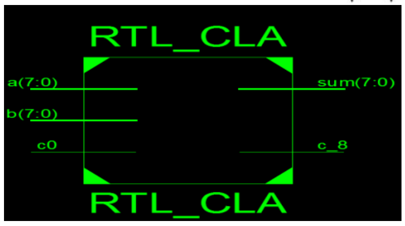


Figure 3.a: RTL schematic of Carry look ahead adder

Fig 3.a represents the RTL schematic diagram of carry look ahead adder. It includes the 2 inputs namely 'a' and 'b' which are of 8 bits. Co- input carry bit, and output named as sum (which is of 8 bit). C8- carry generated will stored in this bit. Fast addition is achieved by carry look ahead adder algorithm.

2. 2's complement:

Two's complement is one of the methods used for subtraction operation on bi-nary numbers, as well as a binary signed number representation based on this operation. . Its extensive use in computing makes it the most important example of a radix complement [8]. The two's complement of an M-bit number is defined as the complement with respect to 2M; in other words, it is the sum of subtracting the number from 2M, which in b-nary is one suffixed by M zeroes. In this method we first need to take the ones' complement and then add one, since the sum of a number and its ones' complement is all 1 bits. The two's complement of a number acts as the negated original number in most arithmetic, and positive and negative numbers can coexist in a natural way.[9]

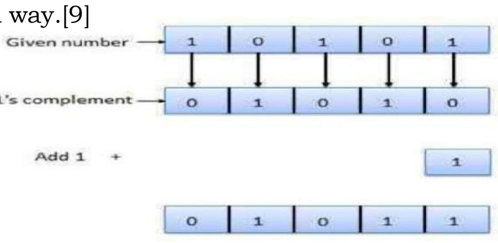


Fig 4. Subtraction using 2's complement.

Fig 4.a represents the RTL schematic diagram of subtraction. Subtractor does the subtraction using 2's complement algorithm. It includes the 2 inputs namely 'a' and 'b' which are of 8 bits. Co- input carry bit, and output named as sum (which is of 8 bit). C8- carry generated will stored in this bit.

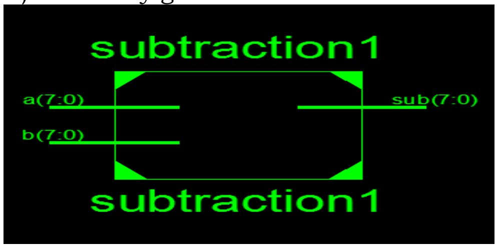
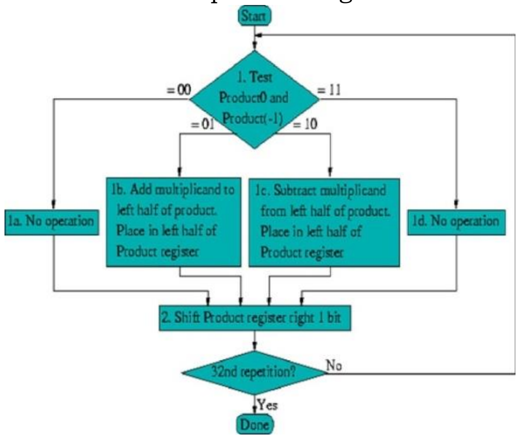


Figure 4.a: RTL schematic of subtraction

3) Booth's multiplication

Booth's multiplication algorithm is an efficient algorithm used for multiplication that multiplies two signed binary numbers in two's complement notation. [11], fig 5.a represents the RTL schematic diagram of multiplication. Booth's algorithm tests adjacent pairs of bits of the M-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, x-1 = 0. For each and every bit xj, for j running from 0 to M-1, the bits xj and xj-1 are considered. The product accumulator P is left unchanged when these two bits become equal, Where xj = 0 and xj-1 = 1, the multiplicand times 2j is added to P; and where xj = 1 and xj-1 = 0, the multiplicand times 2j is subtracted from P. The value of P is the signed product. The product and multiplicand are not specified; typically, these two are also in two's complement representation, like the multiplier, but any number system that holds addition and subtraction will work as well. The order of the steps is not established. Typically, it proceeds from LSB to MSB, starting at j = 0; the multiplication by 2j is then replaced by incremental shifting of the P accumulator to the right between steps; lower bits can be moved out, and subsequent additions and subtractions can then be done just on the highest M bits of P[1]. There are many optimizations and variations on these details. The algorithm is often described as converting strings of 1's in the multiplier to a high-order +1 and a low-order 1 at the ends of the string. When a string runs through the MSB, there is no high-order +1[12] and the net effect is interpretation as a negative of the appropriate value.

Figure 5: Booth's multiplication algorithm


Start
1. Test Product0 and Product(-1)
= 00
= 11
= 01
= 10
1a. No operation
1b. Add multiplicand to left half of product. Place in left half of Product register
1c. Subtract multiplicand from left half of product. Place in left half of Product register
1d. No operation
2. Shift Product register right 1 bit
32nd repetition?
No
Yes
Done

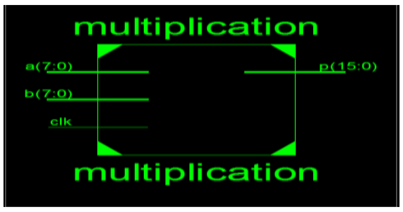


Figure 5.a: RTL schematic diagram of multiplier

Fig 5.a represents the RTL schematic diagram of multiplication. It includes the 2 inputs namely 'a' and 'b' which are of 8 bits. Co- input carry bit, and output named as sum (which is of 8 bit). C8- carry generated will stored in this bit. Fast multiplication is achieved by booths multiplier algorithm if the product of two 8 bit input if exceeding 8 bit then MSB will be hold in result,

The logical operations performed are and, or, xor. The description of and gate is as follows:

The AND gate is one among the basic logic gate that implements logical combination. The output is HIGH(1) only when all the inputs to the AND gate are HIGH (1). If any one of the input is low then the output is LOW. Therefore, the output is always 0 except when all the inputs are 1.

Or gate: The OR gate implements logical combination is a digital logic gate. The output is HIGH (1) if one or all the inputs are 1 (HIGH). If neither input is high, a output is LOW (0) . and OR logical operation,

Xor gate: The XOR gate implements an exclusive or which is a digital logic gate; that is, output is high only when one of its input is high. If both inputs are low or both are high then the output of xor is low. XOR represents the dissimilarity function, i.e., the output is 1 (true) if the inputs are not similar otherwise the output is false. Below figure 6.a , figure 6.b shows RTL schematic diagram of some of the logical operation i.e AND and OR logical operation,

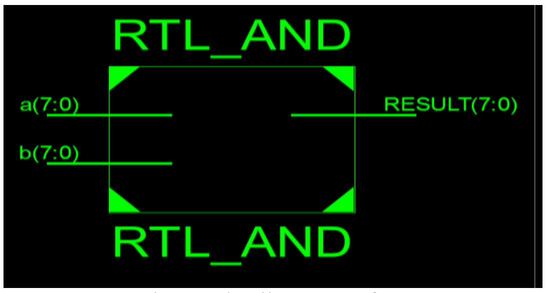


Figure 6.a: RTL schematic diagram of AND gate

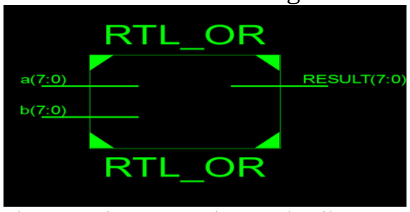


Figure 6.b: RTL schematic diagram of OR gate

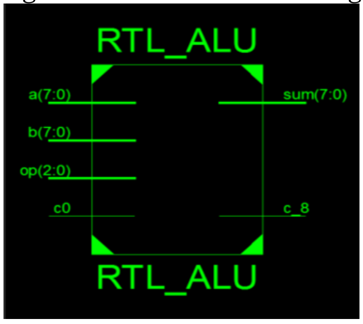


Figure 7: RTL schematic diagram of ALU

IV. RESULTS AND DISCUSSION

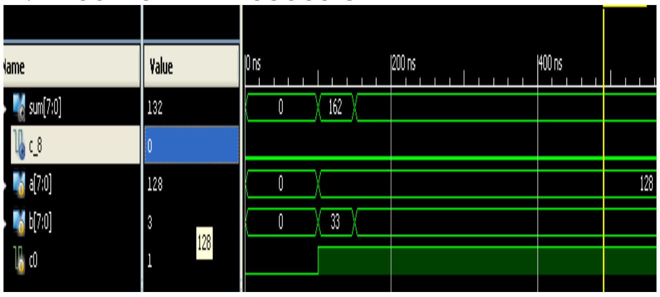


Fig 8: simulation result of carry look ahead adder

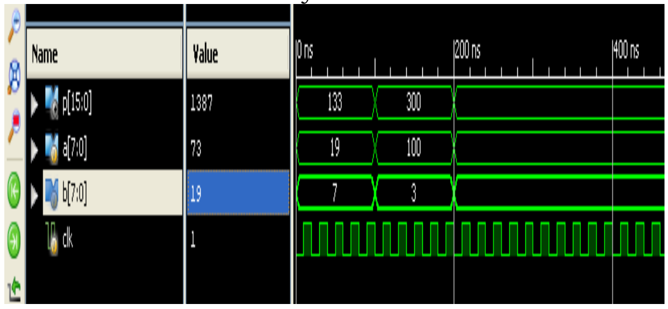


Fig 9: simulation result of multiplier

Above figures represents the simulation result of particular block of ALU using Xilinx software, ISE 14.2 version.

In the simulation figures 'a' and 'b' are operands, results are named based on operation such as s=sum, p=product,

Here operands 'a' and 'b' are of 8 bit.c0 is one of the input which is of 1bit. And result is of 16 bit. Maximum range is limited to 216. Both signed and unsigned integer operation is computed.

Fig 8 represents the simulation result of addition using carry look ahead adder algorithm. The result found is accurate and efficient in terms of speed. Ex: if a=128, b=33, c0=1, then the result sum=162.

Fig 9 represents the simulation result of multiplication built using booths multiplication algorithm. The result found is accurate. Ex: a=100, b=3 and result p=300 .As discussed above 'a' and 'b' are of 8 bits and results is 16 bit. If result of multiplication is exceeding 16 bit, then most significant bit (MSB) will be stored in result block.

Fig 10 represents the simulation result of subtraction built using 2's complement algorithm. The result found is accurate. Ex: a=134, b=18 and result s is s=116.

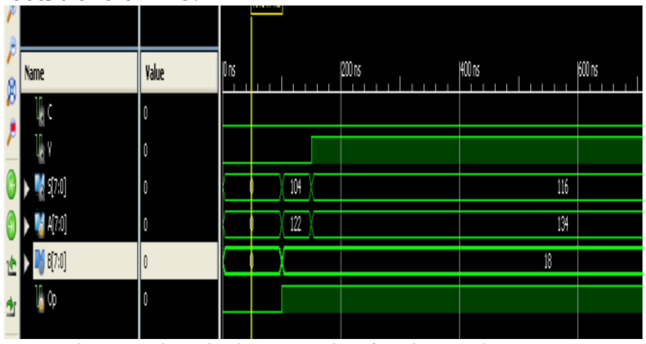


Fig 10: simulation result of subtraction

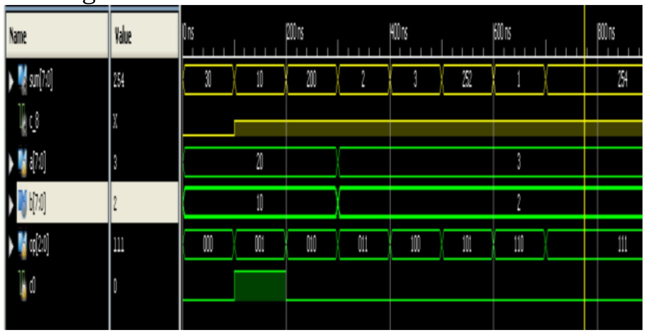


Fig 11.a: simulation result of ALU Operation

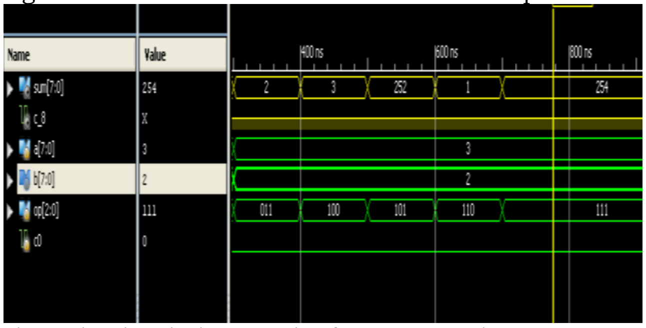


Fig 11.b: simulation result of ALU Operation

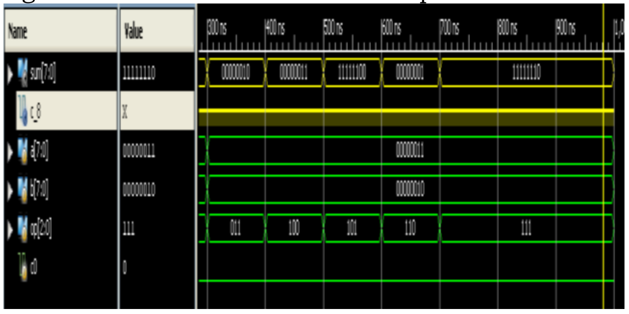


Fig 11.c: simulation result of ALU Operation

Fig 11.a. 11.b, 11.c represents the simulation result of entire ALU which contains arithmetic operations and logical operations. As mentioned in the architecture (fig 2) basically there are 3 select line to select the operation to be performed

Here operands 'a' and 'b' are of 8 bit. And result is of 16 bit. Maximum range is limited to 216. Both signed and unsigned integer operation is computed. Input must be operands and operation to be performed and output is processed inputs

TABLE 1.
LIST OF OPERATION FOR RESPECTIVE SELECT LINES WITH EXAMPLES

Select line	Operation	A	B	Result
000	Addition	20	10	30
001	Subtraction	20	10	10
010	Multiplication	20	10	200
011	AND	00000011	00000010	00000010
100	OR	00000011	00000010	00000011
101	NOT	00000011		11000000
110	XOR	00000011	00000010	00000001
111	XNOR	00000011	00000010	11111110

TABLE II
XILINX DEVICE UTILIZATION SUMMAR

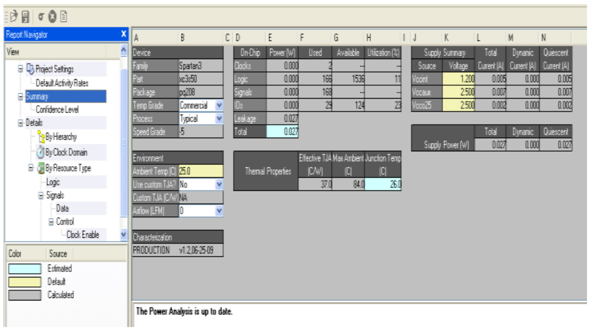
Logic utilization	Available	used	Utilization (%)
Number of Slices	768	91	11%
Number of slice flip-flops	1536	1	0%
Number of 4 input LUT's	1536	169	11%
No. of bounded IOBs	124	29	23%
Number of GCLKs	8	1	12%

TABLE III
XILINX DEVICE TIMING SUMMARY

Table II and III is inferred from synthesis and time analysis of The project.

Minimum input arrival time before clock	14.638ns
Maximum output required time after clock	6.141ns
Maximum combinational path delay	24.535ns

Fig 12. Represents the Result of Power analysis inferred from the Power analysis report.



Color	Source
	Estimated
	Default
	Calculated

Device	
Family	Spartan3
Part	xc3s50
Package	pq208
Temp Grade	Commercial
Process	Typical
Speed Grade	-5

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	2	-	-
Logic	0.000	166	1536	11
Signals	0.000	168	-	-
IOs	0.000	29	124	23
Leakage	0.027			
Total	0.027			

Supply Summary Source	Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.005	0.000	0.005
Vccaux	2.500	0.007	0.000	0.007
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.027	0.000	0.027

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	37.0	84.0	26.0

Characterization	
PRODUCTION	v1.2,06-25-09

The Power Analysis is up to date.

Fig 12. Result of Power analysis

V. CONCLUSION

We have designed and simulated 8 bit ALU by selecting efficient algorithms amongst the plenty of algorithms available. For addition we have used Carry look ahead adder, for subtraction we have used 2's complement and for multiplication we have used booth's multiplication algorithm. The logical operations performed here are matching with their respective truth tables. The results of Table II and Table III are inferred from the synthesis and timing analysis report. Table II briefs about the utilization summary through which we can infer that ALU uses less number of resources. We have used efficient algorithms such as carry look ahead adder (which consumes less time for computation) , 2's complement method for subtraction and booth's multiplication (which is fastest algorithm) to implement our design. Therefore by writing suitable code for required operations we get the simulated result.

VI REFERENCE

[1] Learn about electronics - http://www.learnabout-electronics.org/Digital/dig58.php

[2] CSE 675.02: Introduction to Computer Architecture- Arithmetic / Logic Unit – ALU Design Slides by Gojko Babi http://web.cse.ohio-state.edu/~teodores/download/teaching/cse675.au08/Cse675.02.F.ALUDesign_part1.pdf

[3]//http://www.digilentinc.com/classroom/realdigital/M7/ RealDigital_Module_7.pdf

[4] Booth's multiplier © 2009 Daniel J. Sorin ECE 152 from Roth and Lebeck http://people.ee.duke.edu/~sorin/priorcoues/ece152-spring2009/lectures/3.3-arith.pdf

[5] Dr. Ravi Shankar Mishra,Prof. Puran Gour,Braj Bihari Soni / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com Vol. 1, Issue 3, pp.905-910 905 | P a g e Design and Implements of Booth and Robertson's multipliers algorithm on FPGA http://www.ijera.com/papers/vol%201%20issue%203/YW013905910.pdf

[6] Sakshat lab Carry look ahead adderhttps://en.wikipedia.org/wiki/Carry-lookahead_adder

[7] Virtual lab _iit kharagpur, computer organization and architecture

[8] 2s complement by Thomas Finley, April 2000 https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html

[9]https://en.wikipedia.org/wiki/Two%27s_complement

[10]http://www.cs.utah.edu/~rajeev/cs3810/slides/3810- 08.pdf

[11] 8-bit Verilog code for Booth's multiplier

[12]http://www.enhanceedu.iiit.ac.in/wiki/images/Booth's_multiplier_-_task1_hint.pdf