FPGA realization of ALU for mobile GPU

Mohammed F. Tolba[1], Ahmed H. Madian[1, 3]and Ahmed G. Radwan[1, 2]
1Nanoelectronics Integrated Systems Center (NISC), Nile University, Giza, Egypt.
2Engineering Mathematics and Physics Department, Cairo University, Egypt.
3Radiation Engineering Dept., NCRRT, Egyptian Atomic Energy, Authority.

***Abstract*— Arithmetic Logic Unit (ALU) is the most important component of processors. All arithmetic and logical computations are performed inside the ALU. This paper presents the design and the implementation of the ALU. The design is based on Approximated Precision Shader and Look-Up Table (LUT) multiplier. The lookup table, Wallace tree, and Carry Look-ahead Adder (CLA) are used in combination to speed up the multiplier operation. The proposed ALU is designed using Verilog and verified using Xilinx Virtex-5 XC5VLX30 FPGA.**

***Index Terms*— ALU; GPU; Multiplier; CLA; LUT Multiplier; Verilog; Xilinx; FPGA**

I. Introduction

ALU is the fundamental arithmetic unit that contains fixed point multiplier, adder, dot product, comparator, absolute values and logical operations [1]. Parallel multiplier and parallel adder in the ALU are the key arithmetic method for the speed in processors. The speed of a processor depends on the hardware architecture, delay, and power. High-speed processors require high-speed multipliers. In mobile GPU, the unified shader processor [2] can perform vector operations in the single instruction multiple data (SIMD) architectures. The execution stage in SIMD is the ALU that performs arithmetic and vector operations.

Table I illustrates the comparison between different multipliers based on various techniques. The proposed multiplier supports signed numbers and produces the partial products by using LUT and decoder. While. The array multiplier, the Braun multiplier [8], and the basic binary multiplier supports only the unsigned numbers.

In this paper, the design and the implementation of ALU are proposed for SIMD processor [2] with approximated precision shader and LUT multiplier technique. The proposed work valid for signed numbers and fixed point multiplication, which is suitable for FPGA architectures.

This paper is organized as follows. Section II presents the ALU Architecture. Section III illustrates the proposed multiplier algorithm. In section IV, a signed multiplier algorithm is discussed. Sections V shows simulation results and finally Section VI is the conclusion.

II. ALU Architecture

An Approximated Precision Shader (APS) [2] reduces the power consumption by truncating the 16 least significant bits. Figure 1 presents 32-bit fixed point format divided into an 8-bit integer part and 24-bit for the mantissa. After truncation, only 8-bit integer and 8-bit mantissa are left. Therefore, the principle of operation emphasizes the accuracy power trade-off that present in the modern digital design. The ALU architecture is shown in Fig.2. The proposed ALU design supports fixed point multiplier, adders, dot product, absolute values, and logical operations. A decoder is used to fetch the operation code, source and destination operands.

Table .I comparison between different multiplier architecture

Multiplier Type	Signed	Un-signed	LUT	Decoder	Tree	Final adder
Proposed multiplier	√	√	√	√	√	√
Radix-4 Booth multiplier	√	√	×	√	√	√
Braun multiplier	×	√	×	×	√	√
Array multiplier	×	√	×	×	√	√
LUT multiplier [8]	×	√	√	×	√	√
Basic Binary multiplier	×	√	×	×	√	√

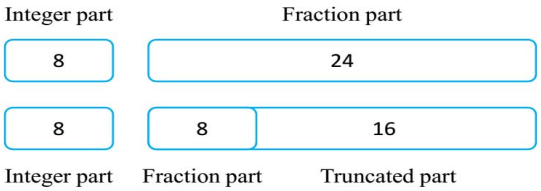

Fig. 1 present truncated fixed point representation

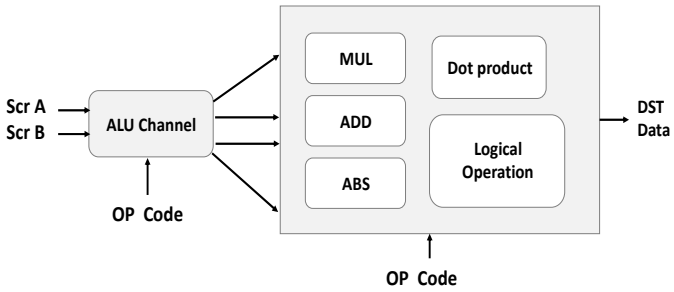

Fig. 2 ALU channel of the SIMD processor.

III. Proposed Multiplier

The implementation of a lookup table (LUT) multiplier for both signed and unsigned numbers is proposed. The LUT

multiplier is based on LUT technique, Level reconfigurable architecture [3] and Karatsuba-Ofman's Algorithm [11]. Figure 3 illustrates the dot diagram of the 8-bit multiplier. The algorithm of the multiplier is presented as follows. 1) The multiplicand and the multiplier are divided into two parts (A0, A1) and (B0, B1). 2) The LUT and the decoder generate the partial products (A0*B0, A0*B1, A1*B0, and A1*B1). 3) The Wallace tree [6] is used to reduce the partial product into two rows. 4) The final multiplication result is computed by using CLA (carry look-ahead adder). The partial products are generated based on table II, where the 4-bit multiplier are used to generate the partial products by the decoder.

The multiplier architecture is shown in Fig. 4, where the LUT and the decoder are used to generate the partial products. The Wallace tree and CLA are used to get the final multiplication result. The Wallace tree is used 8 bits full adder.

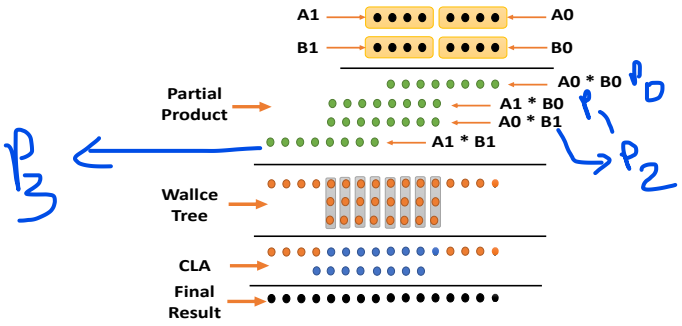

Fig. 3 Multiplier dots diagram

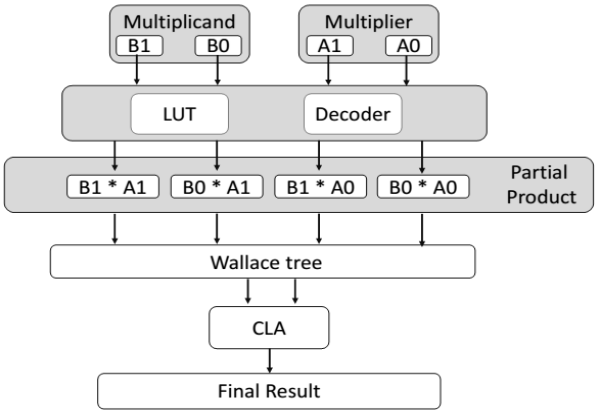

Fig. 4 multiplier architecture

A. *Lookup Table (LUT)*

The LUT is used to hold the multiplication result [4], [5], where the address of LUT is the multiplicand. Figure 5 shows the LUT architecture, where the row and the column address bits determined the desired partial product. For example, the partial product for multiplicand address 0100 and multiplier 0111 refers to 7 * 4.

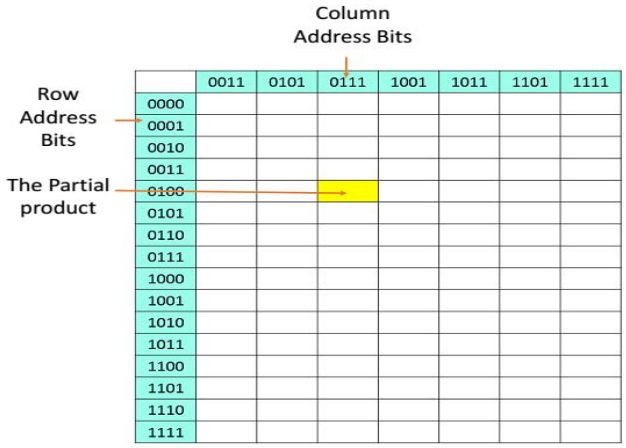

Fig. 5 present the LUT architecture

Figure 6 illustrates the unsigned 16-bit Multiplication by using the same concept. The multiplier and the multiplicand are divided into two parts each part 8-bit. So, 4-partial products are generated. Every partial product is generated by using an 8-bit multiplier. Four 8-bit multiplier work in parallel to produce the partial products. The Wallace tree and CLA compute the final multiplication result.

Table II Partial product selection and generation process

	b3	b2	b1	b0	Operation	Comment
0	0	0	0	0	0	0 *multiplicand
1	0	0	0	1	1	1* multiplicand
2	0	0	1	0	1	Shift 1* multiplicand
3	0	0	1	1	3	LUT
4	0	1	0	0	1	2 shift 1* multiplicand
5	0	1	0	1	5	LUT
6	0	1	1	0	3	Shift result of 3
7	0	1	1	1	7	LUT
8	1	0	0	0	1	3 shift 1* multiplicand
9	1	0	0	1	9	LUT
10	1	0	1	0	5	Shift result of 5
11	1	0	1	1	11	LUT
12	1	1	0	0	6	Shift result of 6
13	1	1	0	1	13	LUT
14	1	1	1	0	7	Shift result of 7
15	1	1	1	1	15	LUT

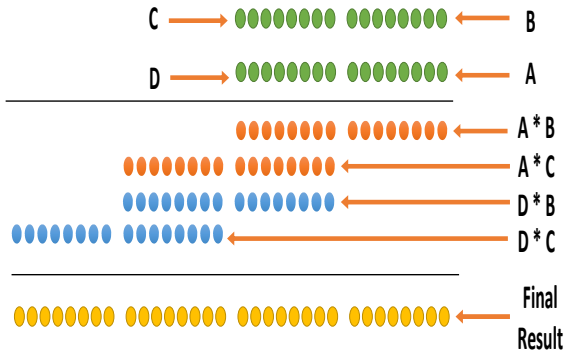

Fig. 6 unsigned 16-bit multiplier

IV. SIGNED MULTIPLIER

The unsigned algorithm is used to produce the unsigned partial products. The one's complement is applied to the multiplicand and the multiplier. The one's complement of the multiplier is AND-ed with the multiplicand sign bit, and the one's complement of the multiplicand is AND-ed with the multiplier sign bit. The sign partial product is generated by adding the output of the two AND gate. The multiplier sign bit XOR-ed and AND-ed with the multiplicand sign bit to get S0 and S1 respectively.

Figure 7 introduces the dot diagram of 8-bit signed Multiplier, where The Wallace Tree [6] is implemented using half adders, full Adders and 4:2 compressors [7]. The Wallace Tree is used to speed up the computation by reducing the number of the addition sequential stages. Using 4:2 compressor increases the speed of the multiplier leading to a smaller number of stages in executing the final result.

A 16-bit multiplier is implemented by using the same algorithm where four unsigned partial product, one sign partial product, and two signal S1 and S0 are generated.

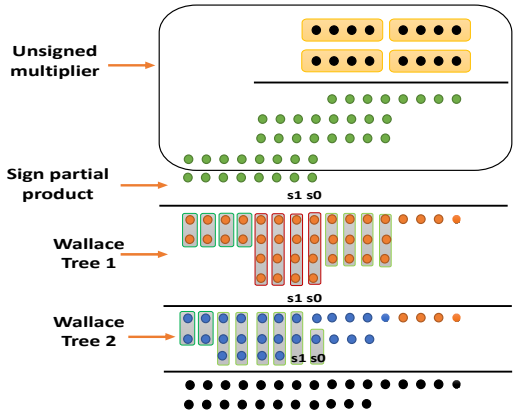

Fig. 7 present the dot diagram of 8 bit signed Multiplier

V. SIMULATION RESULTS

The proposed ALU was implemented on Xilinx Virtex-5 XC5VLX30 FPGA to verify the output. The simulation result is shown in Fig. 8. A 4-bit control signal is used to retrieve the operation code as presented in Table III. The MSB of the 4-bit control signal is used to choose between the logic and arithmetic operations. Table IV illustrates the FPGA implementation results of the ALU, where the total delay is 17.598ns.

Table V shows a performance comparison of different multiplier architectures for 16-bit multiplier in terms of delay with proposed multiplier. The Maximum combinational path delay of proposed multiplier is 6.161ns compared to 9.890ns for [10]. This achieves a reduction up to 38.3% of the total delay. The implementation at table V is verified on Xilinx virtex-6 XC6VCX75T. Another comparison between different 8-bit multiplier architectures regarding delay, no. of occupied slices and percentages of delay is shown in Table VI. The RTL schematic for proposed signed multiplier is presented in Fig. 9.

Table III control word for ALU operations

Input	Operation
0000	Fixed point multiplication
0001	Addition
0010	Subtraction
0011	ABS
0100	Buffer
0101	Dot product
1000	OR gate
1001	NOR gate
1010	AND gate
1011	NAND gate
1100	XOR gate
1101	XNOR gate
1110	NOT In1
1111	NOT In 2
Default	0

Table IV FPGA Implementation Results for the ALU

Logic Utilization	Used	Available	Utilization
No of slices registers	64	19200	1%
No of occupied Slices	451	4800	9%
Max output time after clock	2.783ns		
Delay	17.598ns		

Table V the max combinational path delay comparison of 16-bits multiplier.

Multiplier width	Multiplier type	Max combinational path delay
16 x 16 bit	proposed signed multiplier	6.161 ns
	Radix-4 Modified Booth Multiplier [10]	9.890 ns
	Vedic Multiplier [9]	13.234 ns
	Array multiplier [9]	21.358 ns

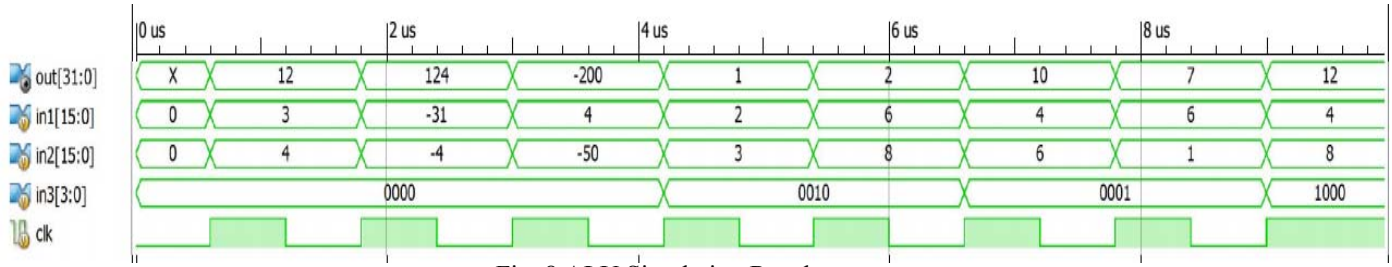

Fig. 8 ALU Simulation Results.

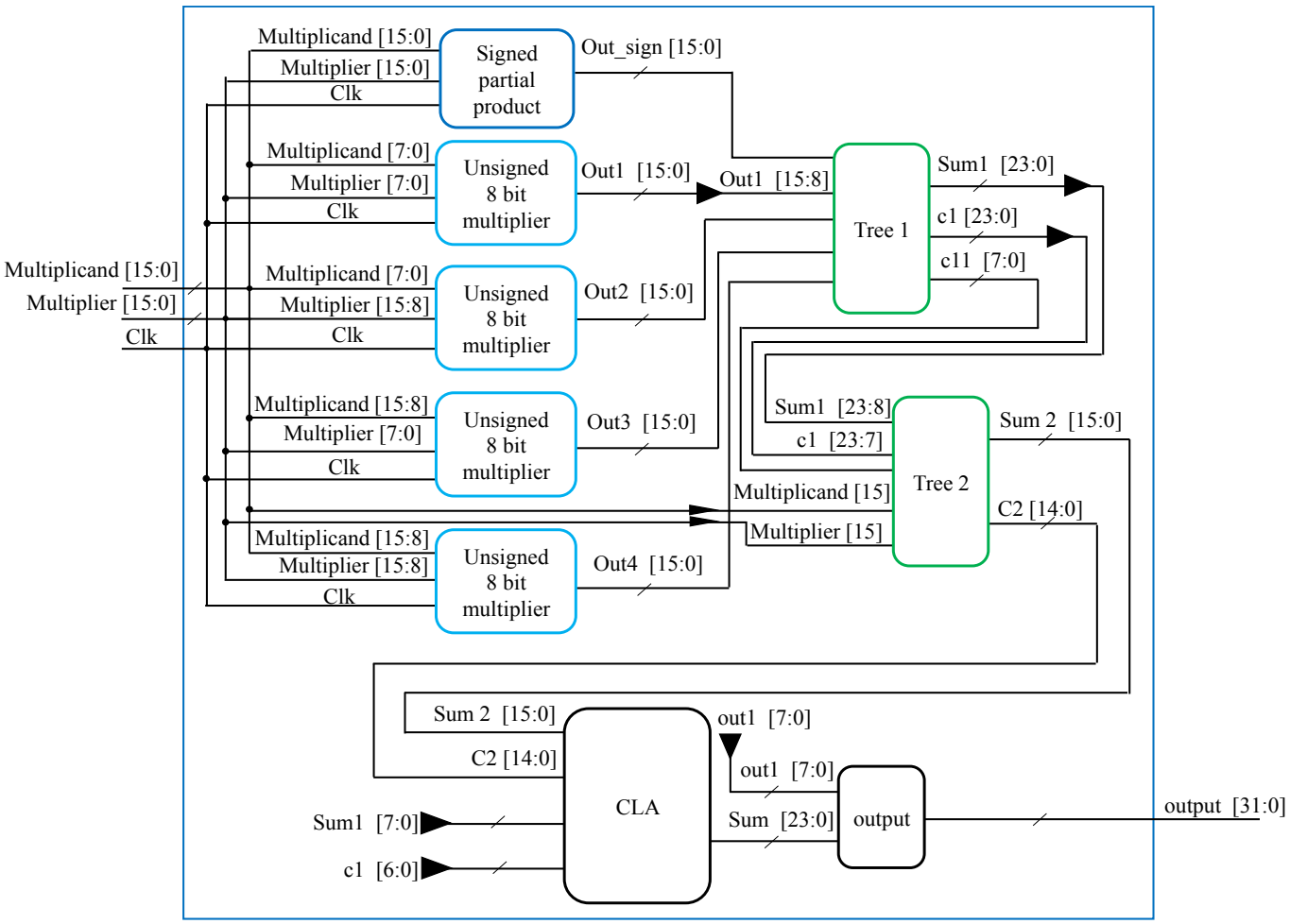

Fig. 9 Proposed signed RTL multiplier schematic

Table VI a performance comparison between different multiplier architectures

Multiplier type	Proposed multiplier	Basic Multiplier [8]	Carry Ripple Multiplier [8]	Radix-2 Modified Booth Multiplier [8]	Radix-4 Modified Booth Multiplier [10]	Karatsuba [12]	Vedic Karatsuba Algorithm[12]	Optimized Vedic Multiplier [12]	Gupta [13]
FPGA Type	Xilinx virtex-5					Xilinx virtex-2			
Delay	7.491 ns	15.67 ns	14.08	11.56 ns	10.124 ns	31.029 ns	18.695 ns	15.418 ns	11.886 ns
No. of occupied slices	80	30	23	23	60	-	-	-	-
Percentage of delay improvement	-	52.2%	46.8%	35.2%	26%	75.9%	59.9%	51.4%	36.98%

The proposed multiplier architecture has better performance than the traditional one by increasing the number of bits. An example for 8-bit multiplier based on proposed signed multiplier illustrates in Fig. 10. Figure 11 shows the comparison between proposed multiplier and radix-4 modified Booth multiplier in terms of delay for different widths. Fig.12 presents the occupied slices for proposed multiplier and radix-4 Booth multiplier.

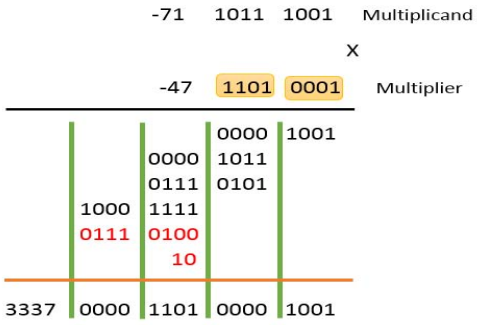

Fig. 10 an example for 8-bit based on proposed signed multiplier

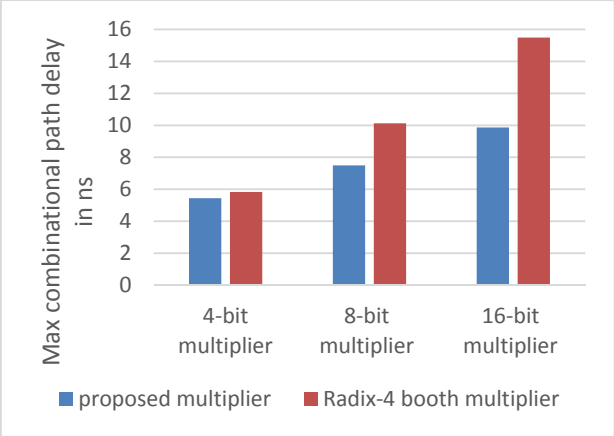

Fig. 11 the delay comparison between proposed multiplier and radix-4 modified booth multiplier.

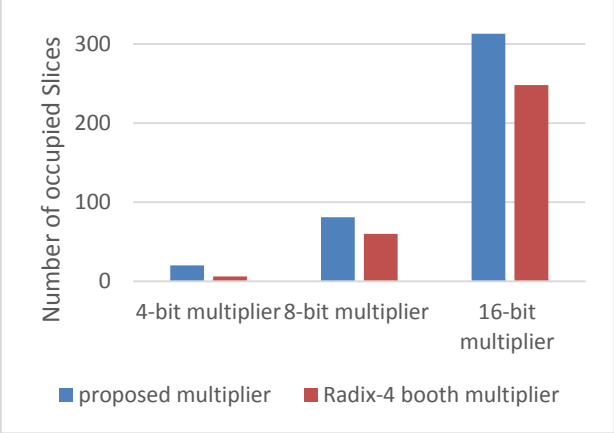

Fig. 12 comparison of the number of occupied Slices between proposed multiplier and radix-4 Booth multiplier.

VI. CONCLUSION

An ALU with LUT multiplier for mobile GPU is proposed. The ALU is based on Approximated Precision Shader (APS). The LUT multiplier was compared with different multipliers regarding delay with various bits width. The proposed multiplier technique improves the speed multiplier by reducing the number of partial products. The ALU design has been simulated using Xilinx ISE and realized on Xilinx virtex-5 XC5VLX30 FPGA. The proposed work gives an overall good performance compared with the conventional one.

REFERENCES

[1] B. Parahmi, Computer Arithmetic: Algorithms and Hardware Designs, 2nd ed. New York, NY, USA: Oxford Univ. Press, 2010.

[2] Y.-J. Chen et al., "A 130.3 mW 16-Core mobile GPU with power-aware Pixel approximation techniques," IEEE Journal of Solid-State Circuits, vol. 50, no. 9, pp. 2212–2223, Sep. 2015

[3] Wei Li, Zi-Bin Dai, Tao Meng Qiao Ren, "Design and Implementation of A High-speed Reconfigurable Multiplier" IEEE International conf. ASIC, pp177-180, Oct.2007.

[4] P. Meher, "LUT Optimization for Memory-Based Computation", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 4, pp. 285-289, 2010.

[5] P. K. Meher, "New approach to LUT implementation and accumulation for memory-based multiplication", Proc. 2009 IEEE Int. Symp. Circuits Syst., ISCAS\'09, pp. 453-456, 2009.

[6] G. W. Bewick, " Fast Multiplication: Algorithms And Implementation," Ph.D. Thesis, Stanford University, February 1994.

[7] V. Oklobdzija, "High-Speed VLSI Arithmetic Units: Adders and Multipliers in Design of High-Performance Microprocessor Circuits", Book Chapter, Book edited by A Chandrakasan, IEEE Press, 2000

[8] Zakaria, Zulhelmi, and Shuja A. Abbasi. "Optimized Multiplier based upon 6-Input Luts and Vedic Mathematics." Proceedings of World Academy of Science, Engineering and Technology. No. 73. World Academy of Science, Engineering and Technology (WASET), 2013.

[9] Jamgade, Roshni, Shrikant Ambatkar, and Sandeep Kakde. "HDL Implementation of PN Sequence Generator Using Vedic Multiplication and Add & Shift Multiplication." Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on. IEEE, 2015.

[10] A.-E. G. Qoutb, A. M. El-Gunidy, M. F. Tolba, and M. A. El-Moursy, "High speed special function unit for graphics processing unit," 2014 9th International Design and Test Symposium (IDT), Dec. 2014.

[11] A. Karatsuba and Y. Ofman, "Multiplication of many-digit numbers by automatic computers," inProceedings of the USSR Academy of Sciences, vol. 145, 1962, pp. 293 –294.

[12] M. Ramalatha, K. Deena Dayalan, P. Dharani, S. Deborah Priya, " High speed energy efficient ALU design using vedic multiplication techniques ", ACTEA, July 15-17, 2009, Lebanon.

[13] A. Gupta, U. Malviya and V. Kapse, "Design of speed, energy and power efficient reversible logic based vedic ALU for digital processors", *2012 Nirma University International Conference on Engineering (NUiCONE)*, 2012.