EXPERIMENTAL STUDIES ON MULTI-OPERAND ADDERS

S. D. Thabah; M. Sonowal and P. Saha
Department of Electronics and Communication Engineering
National Institute of Technology
Meghalaya, India 793003
Emails: sheba.diamond99@nitm.ac.in; mridupawan.sonowal@nitm.ac.in;
sahaprabir1@gmail.com

Submitted: Feb. 15, 2017 ***Accepted: Apr. 15, 2017*** ***Published: June 1, 2017***

Abstract- ***In this paper, different multi-operand adders have been analyzed in terms of propagation delay, power consumption and resource utilization. The functionality of the adders have been verified using Verilog hardware description language and synthesized in Xilinx ISE. The device chosen for implementation is Virtex 6 (XC6VLX240T) with FF1156 package. Simulation results show that Wallace tree adder is the fastest adder and consumes least amount of power. The Wallace tree adder also consumes the least amount of hardware resources as per the synthesis results.***

Index terms: Multi-operand adders; synthesis results; tree adders; Verilog.

I. INTRODUCTION

Arithmetic processing units are basic building blocks of many digital systems like Microprocessors, Digital signal processors, and most of the dedicated signal processing circuits [1-6]. For any arithmetic processing unit, adders are one of the most crucial components because of its resource consumption and the delay involved in processing. Adders also form a part of multipliers which is another resource intensive component of arithmetic circuits [7]. For an efficient implementation of arithmetic circuits, the choice of a particular adder thus becomes an important design consideration [1-9]. Considering the importance associated with the implementation of an adder, in this work, some of the existing adder implementations are compared with respect to the delay incurred, resource utilization and power consumption.

Addition of two bits can be done by using a half adder, while a basic adder that has always been used for performing addition is a full adder. This 1-bit full adder can be used to add multiple operands by using multiple numbers of full adders [8]. Addition of more than two numbers of operands calls for a multi-operand adder [10,11]. In 2005, R. D Kenney and M. J Schulte [1] introduces and analyzes three techniques for performing fast operands addition. Multioperand adder designs are constructed and synthesized for 6 to 12 input operands. S. Singh and R. Waxman [8] described a scheme for multiple operand addition and multiplication, applying the bit-partitioning technique so that each partition contains m-bits of each of these k numbers, where $m=[\log_2(k-1)]$ is an integer $\geq \log_2(k-1)$, the final sum can be obtained in m+1 addition cycles. In 2013, J. Hormigo, J. Villalba et. al. [10] efficiently implemented compressor trees [11] on FPGA, which is more efficient in terms of area and speed, and is made possible by using the specialized carry chains of linear array compressor tree. Linear array compressor trees lead to marked improvements in speed compared to carry propagate adder (CPA) approaches and, in general, with no additional hardware cost. Furthermore the high definition of carry save adder (CSA) arrays based on CPAs facilitates ease-of-use and portability.

Multi-operand adder can simply be represented by an architecture comprising of a compressor tree [13], which reduces the partial sum and propagated carry [10]. There are different types of multi-operand adders but the adders taken up in this work are Array tree adder, Wallace tree adder, Balanced delay tree adder and Overturned-stairs tree adder.

The operands considered for addition can be single bit or of multiple bits, thus the input and output of the adder can be in multiple bits. Nowadays, 8-bits, 16-bits and 32-bits multi operand adder are used in many circuits, for the purpose of comparison, the above mentioned parameters are used. The adders are implemented in Verilog code, and synthesized in Xilinx ISE 13.4 platform. The device chosen for implementation is Virtex 6 (XC6VLX240T) with FF1156 package.

II. CLASSIFICATION OF MULTI-OPERAND ADDERS

Some of the most popular multi-operand adders [13] which have been chosen for implementation purpose are discussed hereunder:

a. Array Tree Adders

Array tree adder is a straight forward multi-operand adder to add and accumulate partial sums [14]. Figure 1, shows architecture of an Array adder for six operands, each of 8 bits. R_i are the inputs (operands), S_i and C_i are the partial sums and partial carries respectively. Sum and C_{out} are the final sum and carry outputs. The shifted version of C_i by one bit position in the left direction are represented by C_i'.

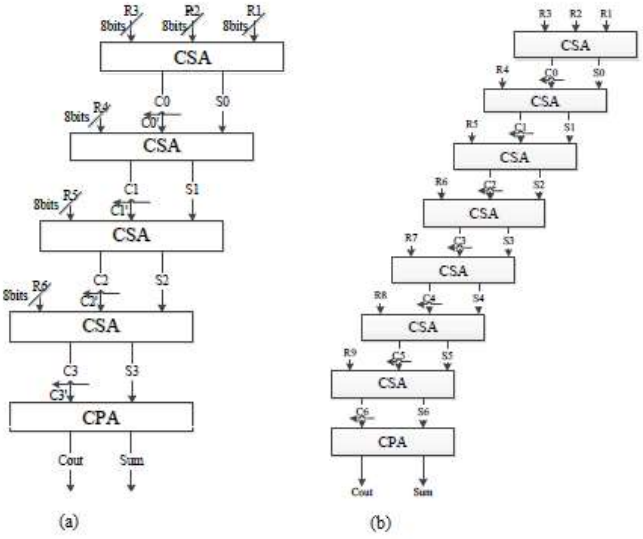


Figure 1: Array tree adders; (a) 6-operand array tree adders (b) 9-operand array tree adders.

b. Wallace Tree Adder

Figure 2 shows the architectures of Wallace tree adder for 6 and 9 operands respectively. In Figure 2, all operands are utilized in a parallel manner, in the first level itself using multiple carry save adders (CSAs). The partial sums and carries (S_i and C_i) generated from the first level are then operated upon in the subsequent levels of the CSA tree, to generate the input for the carry propagete adder (CPA). The final outputs are then obtained from the CPA adder.

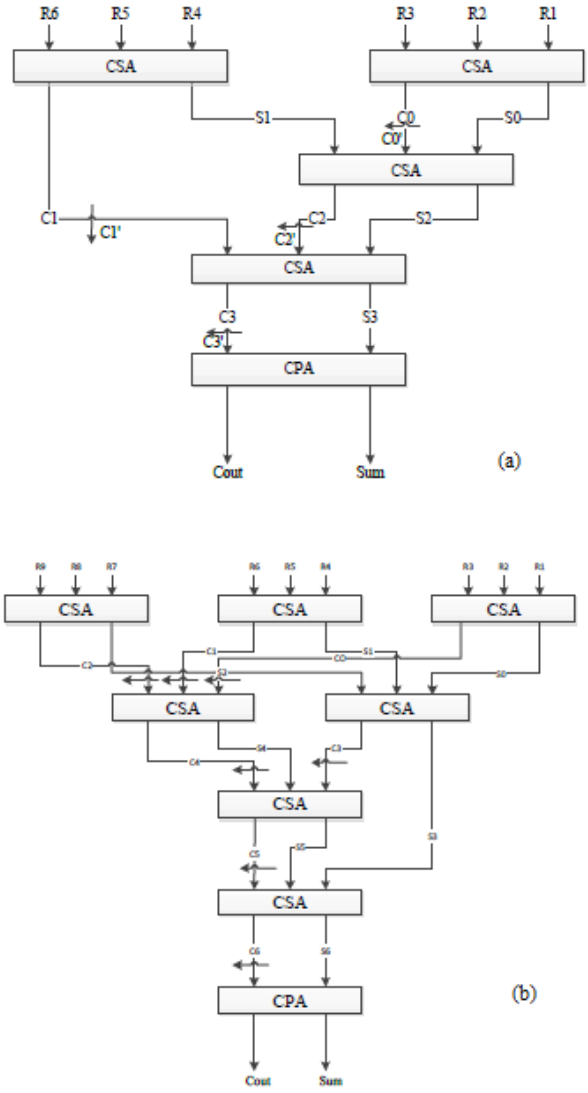


Figure 2: Wallace tree adder; (a) 6-operand (b) 9-operand

c. Balanced Delay Tree Adders

Architectures of balanced delay tree adder for 6 and 9 operands are shown in Figure 3. In this adder, some of the input operands are taken in the first level and the other operands at later stages. In balanced delay tree adders, the amount of delay incurred for each input is approximately same as amount of delay for the other inputs [15]. When the operation of all the partial outputs of the first level is completed, then the other left out inputs are taken in for operation.

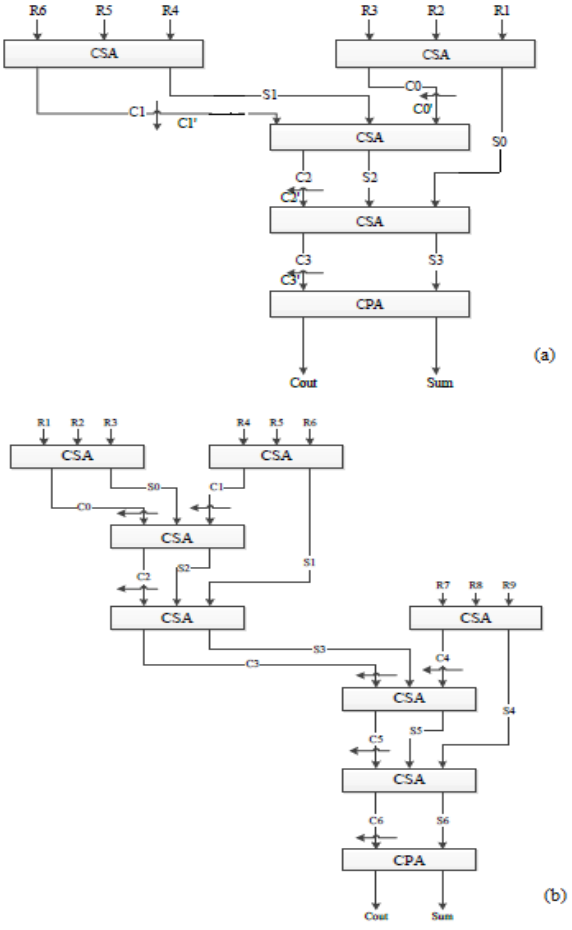


Figure 3: Balanced delay tree adder; (a) 6-operand (b) 9-operand

d. Overturned-stairs Tree Adder

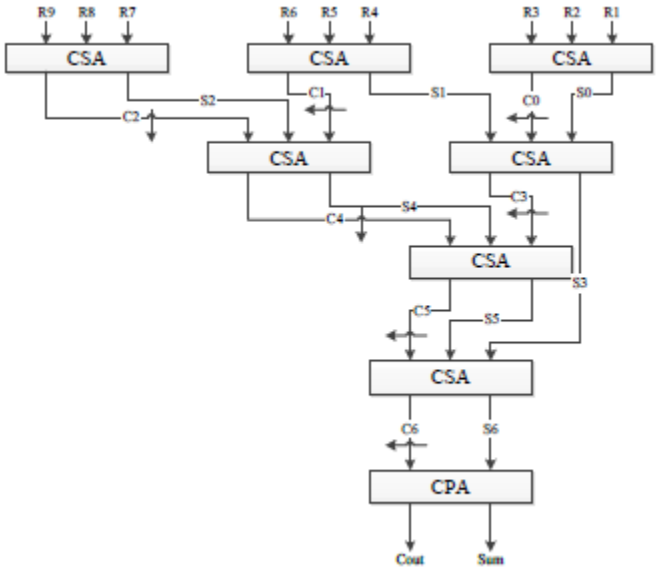


Figure 4: Overturned-stair tree adder

Figure 4, shows architecture of Overturned-stairs tree adder [17] for 9 operands. For 6 operands, the resulting architecture is same as that of a Balanced delay tree adder. Here, it doesn't wait to balance the sets of inputs given to the CSA tree. All operands are accommodated in the first level itself. For 9 operands, in Overturned-stairs tree adder, all operands are accommodated in the first level itself but not in Balanced delay tree. In Balanced delay tree, some sets of operands are taken in the first level and others in the next level so as to balanced the partial sums of the adder.

III. BUILDING BLOCKS

Given below is the brief description of the components used to create the multi-operand adders described in the preceding sections.

a. Carry Propagate Adder (CPA)

Carry propagate adder [18] is designed from a 1-bit full adder (FA). A cascade of n FAs gives a n-bits CPA. Figure 5 shows a block diagram of 4bit-CPA, which add two operands of 4-bits.

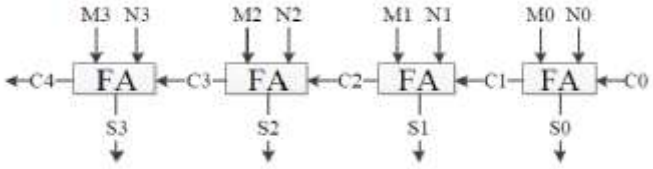


Figure 5: Block diagram of carry propagate adder

Each FA is taking M_i and N_i as inputs and where, M_i and N_i are the corresponding bits of a 4-bit word. S_i and C_i are the sum and carry output of the addition of the two operands. The carry output of the first FA is fed to the second FA and so on, thus the carry is propagated from first FA to the last FA. The first operation of this design starts from the first FA, there is no carry bit in the first FA, thus a half adder can be used or a FA with carry-bit = '0'(C0='0'). The output of first FA is S0 and C1. The carry output C1 is fed to the carry input of second FA and so on. The final carry output is C4 in the Figure 5.

b. Carry Save Adder (CSA)

Carry save adder [17,19] is simply a ripple carry adder where the carries are stored rather than propagated. Figure 6 shows a block diagram of CSA.

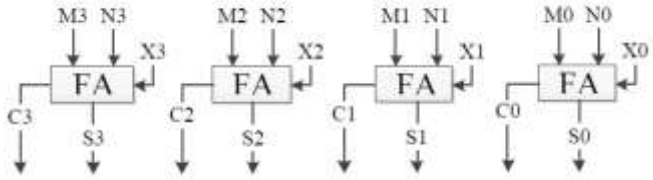


Figure 6: Block diagram of CSA

Similar to Figure 5, the inputs to each of the FA are M_i and N_i and the carry input is Xi, outputs are S_i and C_i. The carry output from each of the FA are not fed to the input of the next FA, implying that the carries are not propagated.

Example: Consider a 4-bit CSA adder

Mi= (1111)2 = (15)10

Ni= (1111)2 = (15)10

Xi= (1111)2 = (15)10

Mi →	1 1 1 1	Mi →	1 1 1 1
Ni →	1 1 1 1	Ni →	1 1 1 1
Xi →	1 1 1 1	Xi →	1 1 1 1
Si→	1 1 1 1	Ci→	1 1 1 1
Si→	1 1 1 1		
Ci'→	1 1 1 1		

Final Sum→ (1 0 1 1 0 1)2 = (45)10

In the above example, a 4-bit addition is performed where each FA inputs are of 1bit, implying that the CSA used above is a 4-bit CSA each FA of single bit inputs. The calculation is done separately for sum Si and carry-out Ci. The carry-out is then shifted by one bit position to the left and added with the sum to get the final sum.

IV. RESULTS AND DISCUSSIONS

The adders are implemented in Verilog code, and synthesized in Xilinx ISE 13.4 platform. The device chosen for implementation is Virtex 6 (XC6VLX240T) with FF1156 package. Performance parameters like as delay, power and resource utilization in terms of look-up tables (LUT) have been considered for the comparison purpose.

Performance parameters as a function of logic delay and routing delay of various multi-operand adders is shown in Table I, II and III. In Table I, 6 operand adders delay have been tabulated where length of each operand length 8-bits, 16-bits and 32-bits have been taken for the reference purpose of the calculations. In Table II, 9 operand adders delay have been tabulated where length of each operand length 8-bits, 16-bits and 32-bits have been considered. Finally in Table III, 12 operand adders have been considered with the same bit length respectively.

Table I: Performance parameters comparison as a function of propagation delay of 6 operand adders

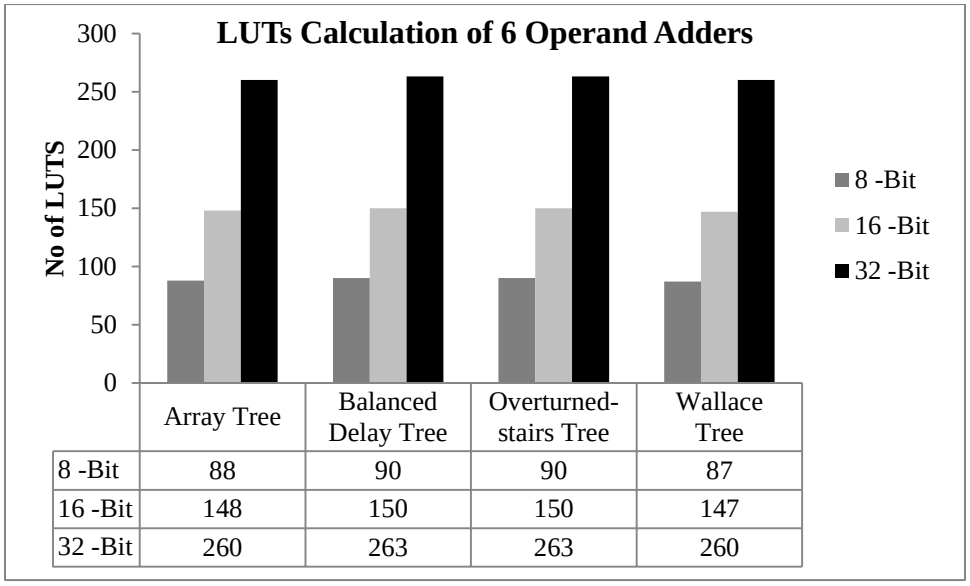
Bit Size →	8bits			16bits			32bits		
	Propagation delay(ns)			Propagation delay(ns)			Propagation delay(ns)		
Architecture ↓	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay
Array Tree	0.686	6.369	7.055	0.958	8.389	9.347	1.502	12.568	14.070
Balanced Delay Tree	0.618	5.523	6.141	0.890	7.543	8.433	1.434	11.575	13.009
Overturned-stairs Tree	0.618	5.523	6.141	0.890	7.543	8.433	1.434	11.575	13.009
Wallace Tree	0.550	5.360	5.910	0.822	7.380	8.202	1.366	11.412	12.778

Table II: Performance parameters comparison as a function of propagation delay of 9 operand adders

Bit Size →	8bits			16bits			32bits		
	Propagation delay(ns)			Propagation delay(ns)			Propagation delay(ns)		
Architecture ↓	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay
Array Tree	0.890	8.673	9.563	1.162	10.693	11.855	1.774	15.230	17.004
Balanced Delay Tree	0.754	6.948	7.702	1.026	8.968	9.994	1.638	13.505	15.143
Overturned-stairs Tree	0.686	6.441	7.127	0.958	8.461	9.419	1.570	12.998	14.568
Wallace Tree	0.618	5.956	6.574	0.890	7.976	8.866	1.502	12.660	14.162

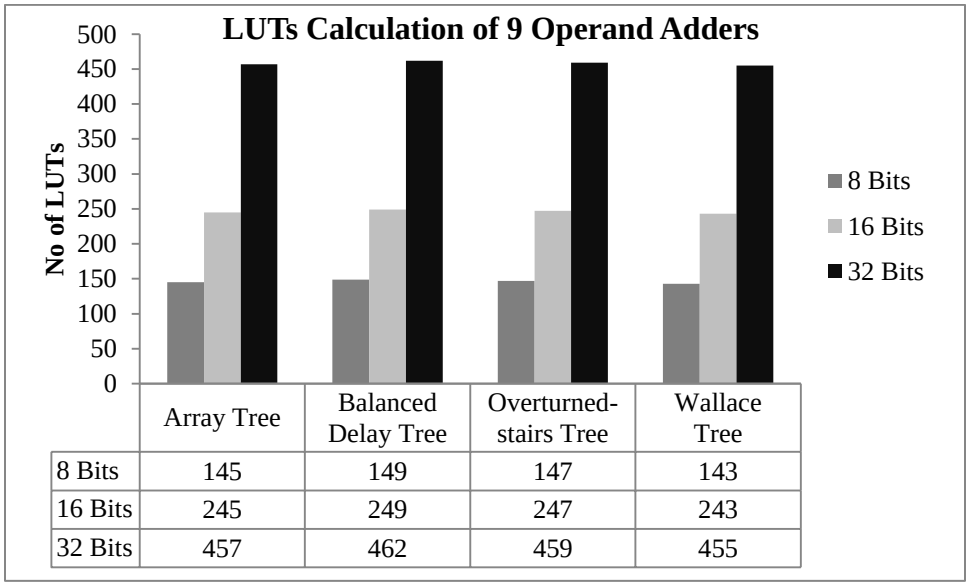
Table III: Performance parameters comparison as a function of propagation delay of 12 operand adders

Bit Size →	8bits			16bits			32bits		
	Propagation delay(ns)			Propagation delay(ns)			Propagation delay(ns)		
Architecture ↓	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay	Logic Delay	Routing Delay	Total Delay
Array Tree	1.230	11.460	12.69	1.502	13.480	14.98	2.046	17.520	19.566
Balanced Delay Tree	0.958	8.465	9.42	1.230	10.485	11.71	1.774	14.525	16.3
Overturned-stairs Tree	0.822	8.197	9.02	1.094	10.217	11.31	1.638	14.257	15.89
Wallace Tree	0.822	7.776	8.59	1.094	9.796	10.89	1.638	13.836	15.47



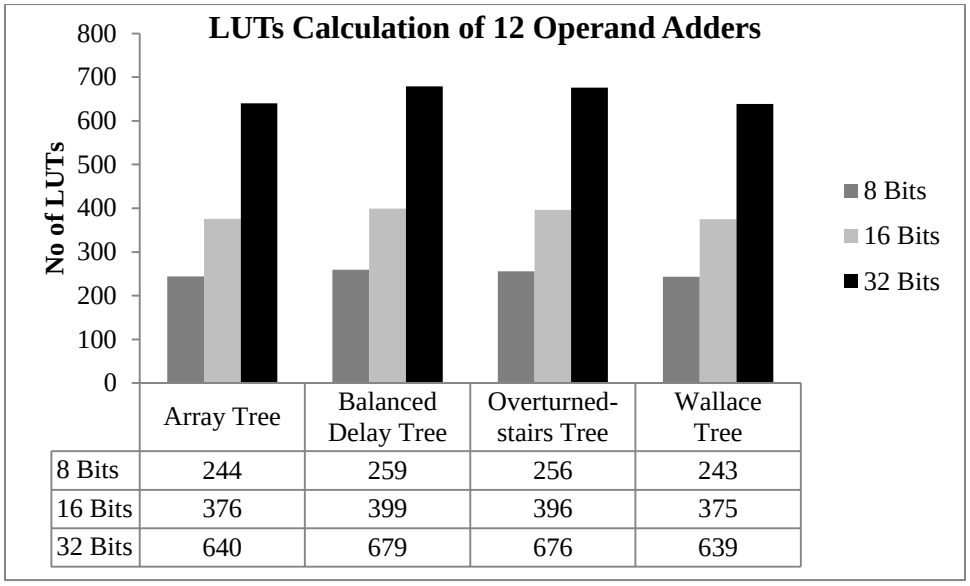
	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 -Bit	88	90	90	87
16 -Bit	148	150	150	147
32 -Bit	260	263	263	260

Figure 7: (a)



	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 Bits	145	149	147	143
16 Bits	245	249	247	243
32 Bits	457	462	459	455

Figure 7: (b)

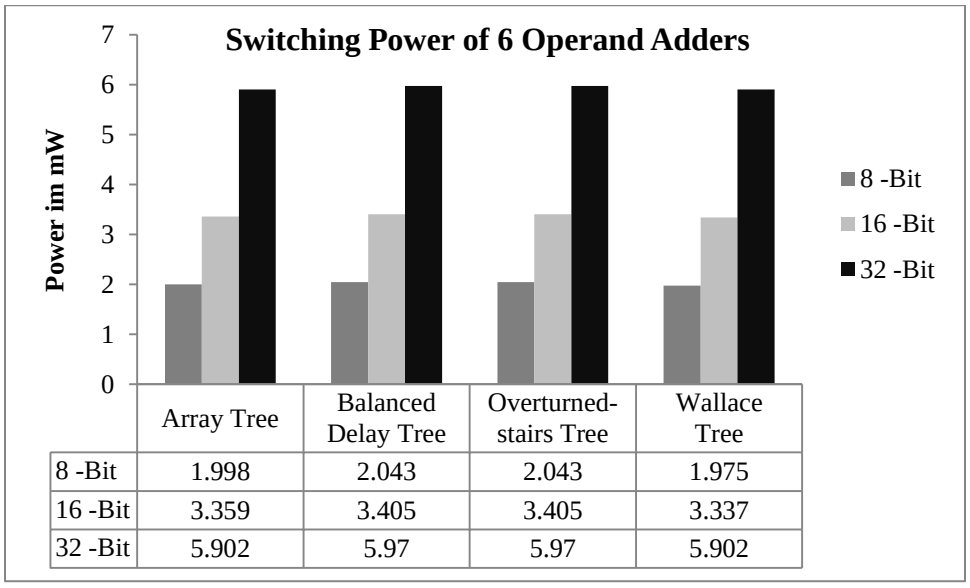


	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 Bits	244	259	256	243
16 Bits	376	399	396	375
32 Bits	640	679	676	639

Figure 7: (c)

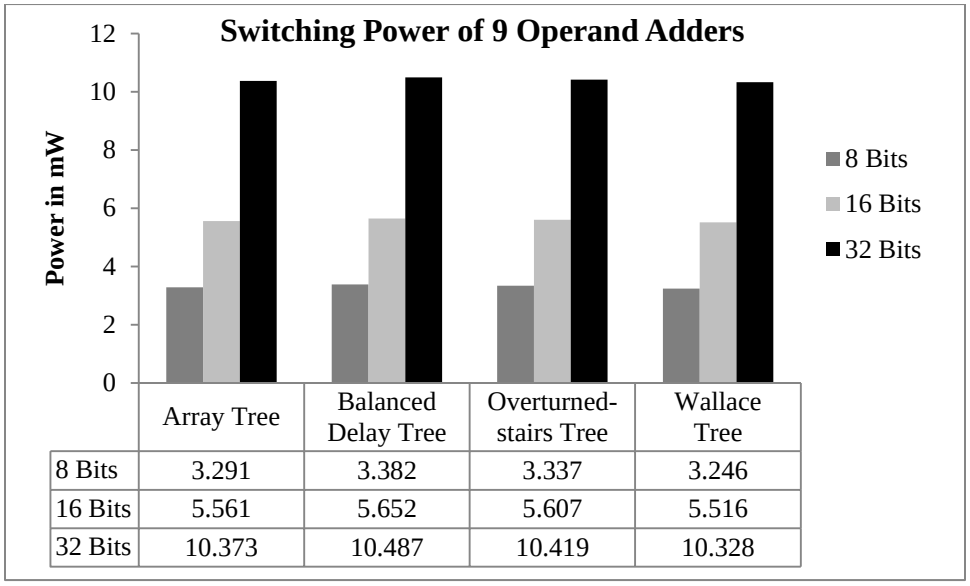
Figure 7: Resource utilization as a function of Look-Up-Tables (LUTs) (a) 6 operand adders (b) 9 operand adders (c) 12 operand adders

The resource utilization as a function of Look-Up-Tables (LUTs) of various length multi operand adders is shown in Figure 7. Each operand of 8-bits, 16-bits and 32-bits respectively has been considered for the calculation purpose. The utilization of slices LUTs is more in Balanced delay tree adder and least in Wallace tree adder.



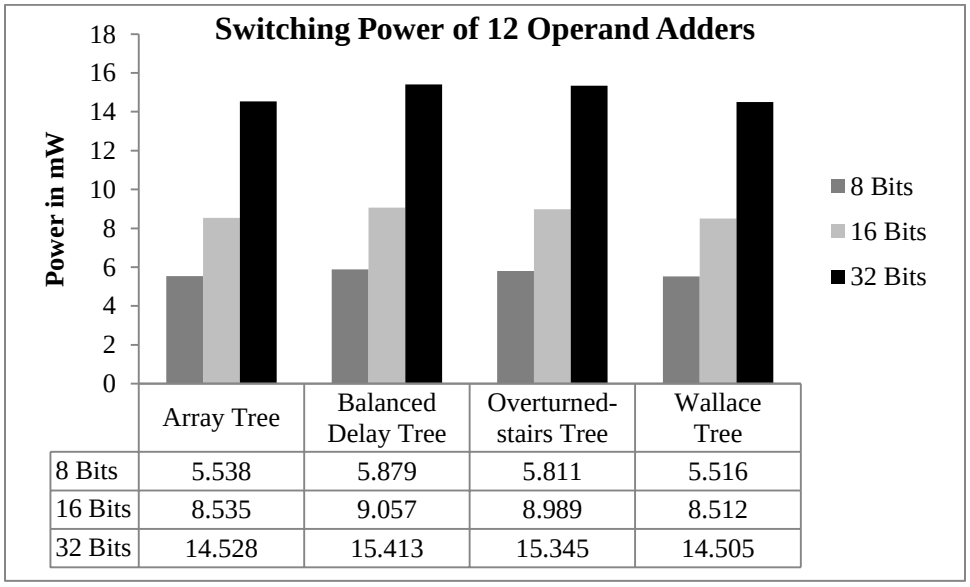
	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 -Bit	1.998	2.043	2.043	1.975
16 -Bit	3.359	3.405	3.405	3.337
32 -Bit	5.902	5.97	5.97	5.902

Figure 8: (a)



	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 Bits	3.291	3.382	3.337	3.246
16 Bits	5.561	5.652	5.607	5.516
32 Bits	10.373	10.487	10.419	10.328

Figure 8: (b)



	Array Tree	Balanced Delay Tree	Overturned-stairs Tree	Wallace Tree
8 Bits	5.538	5.879	5.811	5.516
16 Bits	8.535	9.057	8.989	8.512
32 Bits	14.528	15.413	15.345	14.505

Figure 8: (c)

Figure 8: Switching power analysis (a) 6 operand adders (b) 9 operand adders (c) 12 operand adders

The switching power analysis of various length multi operand adders is shown in Figure 8. It can be seen from the simulated values [Figure (8)] Balanced delay tree adder consume more power and Wallace tree the least.

The simulation results offered, Wallace tree adder gives the lowest overall propagation delay and Array tree adder the highest overall propagation delay. The result of Balanced delay tree and Overturned-stairs tree for 6 operands is same since the architecture is same for 6 operands. So, with increasing number of operands and bit length, Wallace tree adder offered the lowest propagation delay compared to others along-with least consumption of power.

V. CONCLUSIONS

In this paper, different multi-operand adders have been analyzed in terms of propagation delay, power consumption and resource utilization. The adders are implemented in Verilog code, and synthesized in Xilinx ISE 13.4 platform. The device chosen for implementation is Virtex 6 (XC6VLX240T) with FF1156 package. The simulation results shows that Wallcace tree adder gives the best performace among all the adders for all the parameters taken up for consideration.

REFERENCES

[1]. R. D. Kenney and M. J. Schulte, "High-Speed Multioperand Decimal Adders", IEEE Transactions on Computers, vol. 54, no. 8, pp. 953-963, Aug. 2005.

[2]. N. Chabini and S. Belkouch, "Area and delay aware approaches for realizing multi-operand addition on FPGAs using two-operand adders", in Proc. of IEEE/ACS International Conference of Computer Systems and Applications (AICCSA), pp. 1-4, 2015.

[3]. P. Kumar and R. K. Sharma, "Real-time fault tolerant full adder design for critical applications", Engineering Science and Technology, an International Journal, vol. 19 no. 9, pp. 1465–1472, Sept. 2016.

[4]. M. Sushmidha and B. Premalatha "Design of high performance parallel self timed adder", in Proc. of Int. Conf. on Communication and Signal Processing (ICCSP) pp. 1400 – 1404, 2016

[5]. P. I. Balzola, M. J. Schulte, J. R. J. Glossner and E. Hokenek "Design Alternatives for Parallel Saturating Multioperand Adders", in Proc. of *IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 172-177, 2001.

[6]. L. Dadda, and V. Piuri, Pipelined Adders, IEEE Transactions on Computers, vol. 45, no. 3, pp. 348-356, March 1996.

[7]. J. Saini, S. Agarwal and A. Kansal, "Performance, Analysis and Comparison of Digital Adders", in Proc. *Int. Conf. on Advances in Computer Engineering and Applications (ICACEA), pp.* 80-83, 2015.

[8]. S. Singh and D. Waxman, "Multiple Operand Addition and Multiplication", IEEE Transactions on Computers, vol. C-22, no. 2, pp. 113-120, Feb. 1973.

[9]. A. Albeck and S. Wimer, "Energy efficient computing by multi-mode addition", Integration the VLSI journal, vol. 55, no. 9, pp. 176-182, Sept. 2016.

[10]. J. Hormigo, J. Villalba and E. L. Zapata, "Multioperand Redundant Adders on FPGAs", IEEE Transactions on Computers, vol. 62, no. 10, pp. 2013-2025, Oct. 2013.

[11]. U. Cini and O. Kurt "MAC unit for reconfigurable systems using multi-operand adders with double carry-save encoding", Int. Conf. on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1-4, 2016

[12]. J. Villalba, J. Hormigo, J. M. Prades and E. L. Zapata, "On–line Multioperand Addition Based on On–line Full Adders∗", in Proc. *Int. Conf. on Application-Specific Systems, Architecture Processors (ASAP'05)*, pp. 322-327, 2005

[13]. H. Parandeh-Afshar, P. Brisk and P. Ienne "Efficient Synthesis of Compressor Trees on FPGAs", in Proc. of *Asia and South Pacific Design Automation Conference*, Mar. 2008, pp. 138-143

[14]. Hardware Algorithm for Arithmetic Modules: http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html

[15]. H. Al-Twaijry and M. Flynn "Performance/Area Tradeoff in Booth Multipliers" Technical Report, Nov. 1995, Stanford University.

[16]. W. Li and L. Wanhammar, "A complex multiplier using overturned-stairs adder tree," in Proc. of *IEEE Int. Conf. on Electronics, Circuits and Systems*, pp. 21-24, 1999.

[17]. S. J Piestrak, "Design of Residue Generators and Multi-operand Modular Adders using carry save adder", *IEEE Transactions on Computers*, vol.43, no.1, pp. 68-77, January 1994.

[18]. A. Ibrahim and F. Gebali, "Optimized structures of hybrid ripple carry and hierarchical carry lookahead adders", Microelectronics Journal, vol. 46, no. 9, pp. 783–794, Sept. 2015.

[19]. I. Koren, Computer Arithmetic Algorithms. Englewood Cliffs, N.J.:Prentice-Hall, 1993.