ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2386-7876
Vol.01, Issue.03
March-2026
Pages: 305-318

DESIGN AND PERFORMANCE ANALYSIS OF 32-BIT VLSI HYBRID ADDER

[1] **P Renuka**, [2] **Badugu Sai Kumar**, [3] **Abhiram Vasipalli**, [4] **Appasani Bhargav**, [5] **Chidipothu Giri**, [6] **Dunnapothula Sreekanth**

[1]Guide, Dept of ECE, ABR College of Engineering and Technology, Kanigiri, A.P.
[2,3,4,5,6]B. Tech, Dept of ECE, ABR College of Engineering and Technology, Kanigiri, A.P.

ABSTRACT: Addition is one of the most basic operations performed in all computing units, including microprocessors and digital signal processors. It is also a basic unit utilized in various complicated algorithms of multiplication and division. Efficient implementation of an adder circuit usually revolves around reducing the cost to propagate the carry between successive bit positions. Hence, a new high-speed and area efficient adder architecture is proposed using pre-compute bitwise addition followed by carry prefix computation logic to perform the three-operand binary addition that consumes substantially less area, low power and drastically reduces the adder delay.

Keywords: Carry Propagation Delay, Area Optimization, CMOS Technology, Hybrid Adder Architecture, Carry Select Adder (CSLA), Carry Look-Ahead Adder (CLA), Ripple Carry Adder (RCA), Gate Delay Reduction, Power-Delay Product (PDP), Transistor-Level Design

INTRODUCTION: The challenge of the verifying a large design is growing exponentially. There is a need to define new methods that makes functional verification easy. Several strategies in the recent years have been proposed to achieve good functional verification with less effort. Recent advancement towards this goal is methodologies. The methodology defines a skeleton over which one can add flesh and skin to their requirements to achieve functional verification. The report is organized as two major portions; first part is brief introduction and history of the functional verification of regular Carry Sskip adder which tells about different advantages of Carry skip adder and RCA architecture and in this Regular model, there is a drawback and in order to overcome that complexity, the modified architecture of CSKA has been designed. The electronics industry has achieved a phenomenal growth over the last two decades, mainly due to the rapid advances in integration technologies, large-scale systems design due to the advent of VLSI. The number of applications of integrated circuits in high-performance computing, telecommunications and consumer electronics has been rising steadily and at a very fast pace. Typically, the required computational power of these applications is the driving force for the fast development of this field.

The figure 2.1 gives an overview of the prominent trends in information technologies over the next few decades. The current leadingedge technologies already provide the end-users, a certain amount of processing power and portability.

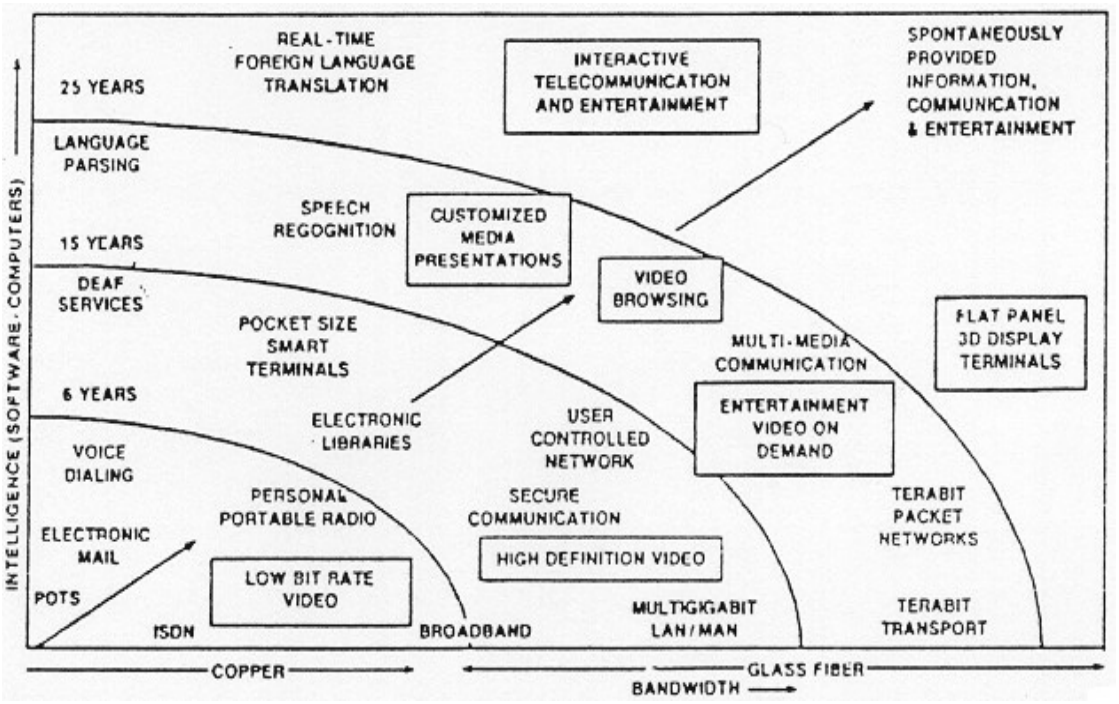

Fig 1: Overview of the prominent trends in information technologies

This trend is expected to continue with very important implications on VLSI and systems design. One of the most important characteristics of information services is their increasing need for very high processing power and bandwidth (in order to handle real-time video, for example). The other important characteristic is that the information services tend to become more and more personalized (as opposed to collective services such as broadcasting), which means that the devices must be more intelligent to answer individual demands and at the same time they must be portable to allow more flexibility/mobility. As more and more complex functions are required in various data processing and telecommunications devices, the need to integrate these functions in a small system/package is also increasing. The level of integration, as measured by the number of logic gates in a monolithic chip, has been steadily rising for almost three decades mainly due to the rapid progress in processing technology and interconnect technology. Table 1 shows the evolution of logic complexity in integrated circuits over the last three decades and marks the milestones of each era. Here, the numbers for circuit complexity should be interpreted only as representative examples to show the order-of-magnitude. A logic block can contain anywhere from 10 to 100 transistors, depending on the function. State-of-the-art examples of ULSI chips, such as the DEC Alpha or the INTEL Pentium contain 3 to 6 million transistors.

ERA (number of logic blocks per chip)	DATE	COMPLEXITY
Single transistor	1959	less than 1
Unit logic (one gate)	1960	1
Multi-function	1962	2-4
Complex function	1964	5-20
Medium scale integration	1967	20-200
Large scale integration	1972	200-2000
Very large scale integration	1978	2000-20000
Ultra large scale integration	1989	>20000

Table2: Evolution of logic complexity in integrated circuits

The most important message here is that the logic complexity per chip has been (and still is) increasing exponentially. The monolithic integration of a large number of functions on a single chip usually provides:

- Less area/volume and therefore compactness.
- Less power consumption.
- Less testing requirements at system level.
- Higher reliability, mainly due to improved on-chip interconnects.
- Higher speed, due to significantly reduced interconnection length.
- Significant cost savings.

The discussionon different types of adders is carried out here and the comparison is carried out with respect to their own functionalities.

LITERATURE SURVEY: The Multi-Operand Adders are generally implemented in two methods i.e Array Adders and Adder Tree structure. In Array Adder structure, two operands are added and output is added with third operand and continues the chain of addition until to get final sum output. It requires 'K' number of adder levels for addition of 'K' operands. But in case of Adder Tree structure the number of levels to add 'K' operands is less than that of Array Adders. It groups 'K' number of operands into sets of two operands. All the sets are added parallel in one level. The sum outputs from first level again grouped into sets of two operands and perform addition. This process continues until to get two operands and added in last level to obtain final sum. In each level it reduces number of operands to half. Therefore it requires log2 K levels. The Adder Tree structure is faster than Adder Array structure with same resources consumed by both configurations. But the Array Adder is having regular routing than Adder Tree structure. The Ripple Carry Adder (RCA) or Carry Look Ahead Adder (CLA) are two general Carry Propagate Adders used in the above methods i.e. Array Adder, Adder Tree is Carry Propagate Adder. The delay of

their CPA depends on bit length of operand. For N-bit operand the of RCA proportional to N and for CLA it is proportional tolog2 N. To reduce the delay these adders where implemented on FPGA by using dedicated carry chains [8]. The RCA on FPGA using fast carry chain is simpler than any other CPA topologies at an expense of high hardware cost [9]. The pipelining technique can be applied more effectively RCA [1]. The delay of Adder Tree using CPA is high due to carry propagation along the bit length. Carry Save Adder tree is used as another approach for implementing Multi-Operand Adders. Here the carry is directly propagated to next level instead of propagating in the same as in case of CPA. The advantage of Carry Save Adder (CSA) tree is utilized in ASIC implementation due to flexible routing. The critical path delay can be minimized by optimizing the interconnection between Full Adders. But to implement on FPGA the Ripple Carry Adder tree is preferred than CSA adder tree. When CSA tree is implemented on FPGA it become slower than RCA tree due to routing delay of CSA. However, a straightforward implementation on FPGAs [6] roughly requires double hardware than a carryripple adder, and does not exploit the fast carry chain to improve speed. This was partially solved by the compressors which compress the operands more than CSA tree [5], [7]. In 2005, R. D Kenney et al. [10] introduced and analyzed three techniques for performing fast operands addition [10]. Multi-operand adder designs are constructed and synthesized for 6 to 12 input operands. In 2005, J. Villalba et al. have studied the on–line addition of multiple operands for conventional, carry–save (CS) and/or Signed–Digit (SD) numbers [11]. They have proposed a new and key element called on-line Full Adder which is used in CSA tree to perform Multi-Operand Addition serially. The on-line Full Adder contains a 1-bit delay register at the sum bit output of FA. An external delay element also inserted at every skipped level in tree structure to maintain proper timing. The on-line Full Adder is used to reduce the hardware resources and cycle time. In 2009, Manuel Ortiz et al. [12] have implemented 3:2 and 4:2 compressors using Dedicated carry chain in FPGA. When more than three operands are to be added the 4:2 compressors is preferred because it fully utilizes the logic of the slice and achieves high speed than 3:2 compressor with same resources. In 2009, William Kamp et al. [13] has implemented the basic 3:2 and 4:2 compressors using dedicated carry chain for redundant numbers to eliminate carry propagation. So that the delay is maintained nearly constant even for large operand width targeting low cost FPGAs. In 2013 J. Harmigo et al. [14] was proposed a complete compressor tree using the dedicated fast carry chain unlike basic compressors developed in [12], [13]. They developed a linear array of carry save adder by mapping to preceding stages of FAs to conventional Ripple Carry Adders (RCA) or ternary adders.

EXISTING TECHNIQUE:

CARRY SAVE ADDER:

We are using Carry Save Adder for the design of our 32-bit multiplier, so let us first understand the working of Carry Save Adder. Carry Save Adder is mainly used in the addition of three or more n-bit numbers. CSA is identical to the Full Adder. Instead of using any other adder here we can use CSA for the addition of the partial product terms of each group. Other adders when compared with CSA are slow and CSA working is more easy to understand. We need to add more than two numbers together in many cases. The easiest way of adding together m numbers is by

adding the first two, wherein both the numbers are n bits wide. Then their sum is added to the next one and so on. Totally m − 1 additions will be required, for a total gate delay of O (i.e. m log n), forming a tree of adders, considering only O (i.e. log m * log n) gate delays. With the help of carry save addition, we can reduce the delay to a large extent. The logic is to take three numbers which we want to be added together i.e. x + y + z, and convert them into two numbers c + s, which is given by x + y + z = c + s, this operation is done in O (1) time. The reason why we cannot perform addition in O (1) time is because we must send forward the carry information. In carry save addition, we restrict ourselves from directly passing on the carry information until the very last step. We will first illustrate this concept with an base 10 example. When we add three numbers by hand, we usually align all the three operands, and then move ahead column by column in the same manner as we perform addition with two numbers [3]. The three numbers in the row are added, and if any overflow is generated it goes into the next column. When there is some non-zero carry, we are actually adding four digits.

Carry:		1	1	2	1	
x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
sum:		7	1	1	0	4

This process is broken down into two steps using the carry save approach. Initially we need to compute the sum by ignoring any carries:

x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
s:		6	0	9	9	4

Herein each s_i is the same as the sum of $x_i + y_i + z_i$ modulo 10. Now we can calculate the carry for the same on a column by column basis:

x:		1	2	3	4	5
y:		3	8	1	7	2
z:	+	2	0	5	8	7
c:		1	0	1	1	

Here each ci is the sum of the bits from previous column is divided by 10 discarding any remainder. Any carry over from one column goes into the next column. Now, c and s, can be added together and we can verify it to be x + y + z:

s:		6	0	9	9	4
c:	+	1	0	1	1	
sum:		7	1	1	0	4

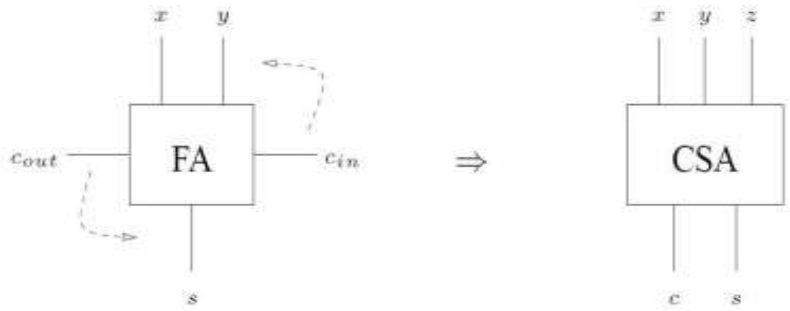

Fig2. The Carry Save Adder (CSA) block and the Full Adder have same circuitry

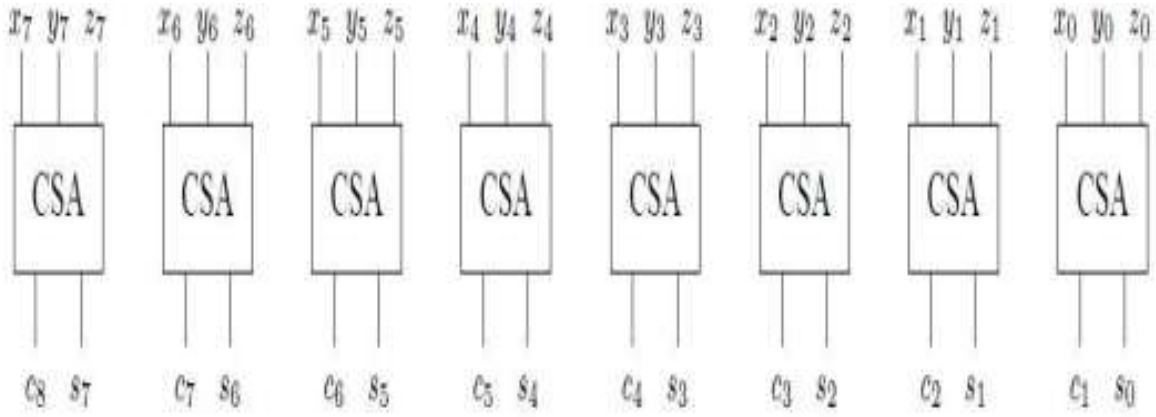

Fig3. One CSA block is used for each bit.

This circuit adds three n = 8 bit numbers together into two new numbers. The important point which can be seen here is that c and s can be calculated individually and also each ci (and si) can be calculated individually from all of the other c's (and s's). In this way we can achieve our goal of converting three numbers that we wish to add into two numbers that add up to the same sum, and in O (1) time. The same logic can be applied and implemented on binary numbers. Which we can see using the below example:

x:		1	0	0	1	1	
y:		1	1	0	0	1	
z:	+	0	1	0	1	1	
s:		0	0	0	0	1	
c:	+ 1	1	0	1	1		
sum:		1	1	0	1	1	1

The circuit which is used to compute s and c, how does it actually look like? It is actually very much similar to the conventional full adder, but some of the signals are renamed. Fig shows a full adder and a carry save adder. A carry save adder is actually a full adder only but with the cin input renamed to z, the output z is renamed as s, and the output cout is renamed as c. Fig 2 shows how 'n' carry save adders can be arranged to add three n bit numbers x, y and z into two numbers c and s. It is also very important to note that the carry save adder block in the zero bit position generates c1 and not c0. Similarly the least significant column when we are adding numbers by hand, c0 will be equal to zero. All of the CSA blocks are actually independent, thus the entire circuitry takes only O (1) time. We still need a LCA to obtain the final sum, for which we require O (log n) amount of delay. The asymptotic gate delay for the addition of three n-bit numbers is in this way is the same as to add just two n-bit numbers. So to what extent does it take us to include m diverse n-bit numbers together? The straightforward methodology is just to repeat this process roughly m times over. This is shown in Fig 3. We have m − 2 CSA blocks wherein every block in the figure really shows numerous one bit CSA blocks in parallel, that we need to experience and after that the last LCA. Whenever we pass through a CSA block, the size of our number increases by one bit.

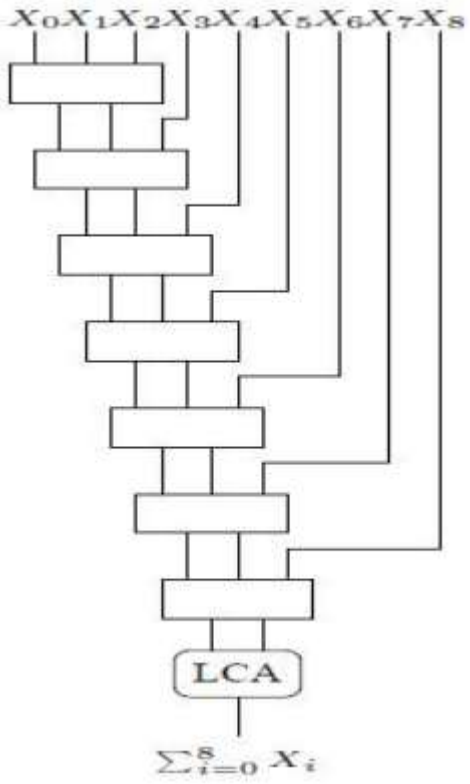

Fig.4 Chain of CSA's used to add m n-bit numbers

Hence the numbers that go to the LCA will be at mostly n + m − 2 bits long. Therefore the final LCA will have a gate delay of O (log (n + m)). Therefore O (m + log (n + m)) is the total gate delay. But instead of rearranging the CSA blocks to form a chain, we can actually use a tree formation. This circuit is called a Wallace tree the depth of the tree is log3/2 m. But as we move deeper in the tree there is an increment in the width of the numbers. At the end of the tree, the numbers will be O (n + log m) bits wide, and therefore the LCA will have an O (log (n + log m)) gate delay. The total gate delay of the Wallace tree is thus O (log m + log (n + log m)). The carry signals that propagate along the rows of the ripple-carry adders can instead be connected into the next row in a diagonal fashion. This strategy implements the carry-save technique. See below Figure. The delay of the array of carry-save adders is O(log n) since the worstcase delay path corresponds to a carry signal that begins in the first row and ends in the last. The array does not, however, deliver the required sum, since the diagonal propagation of carry signals leaves two intermediate results at the bottom of the array, the sum outputs and the unresolved carries. An O(log n) fast adder can add these and keep the total time to O(log n).

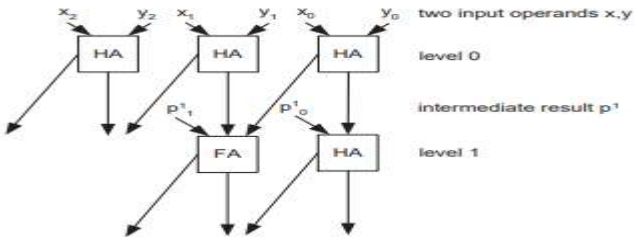

Figure5: Upper-Right Corner of the Carry-Save Arrays of Adders

Each row of the CSA array reduces three inputs to two output operands. This observation leads to efficient tree structures, such as illustrated in Figure.

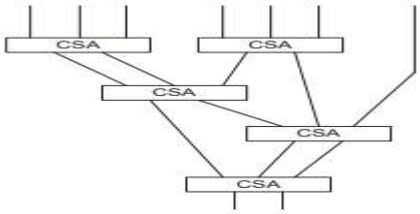

Figure 6: CSA tree for 7 operands

1. Idea: Add three **k**-bit numbers in a single bit step, producing two **k+1**-bit numbers.
2. Implementation: Use a binary full adder for each bit position, producing a low order bit and a carry bit for each bit position.
3. The final sum is the sum of the low order bits and the shifted carry bits.
4. Example:

$$\begin{aligned} a &= 10110001 \\ a^{I} &= 01101101 \\ a^{II} &= 10100100 \end{aligned}$$

5. Time required to reduce the sum of **N k**-bit numbers to the sum of two $k + \log N$-bit numbers: $\log_{\frac{3}{2}} N + 1$ bit steps.

   Proof:

   1. After the first step of CSA, how many numbers are we left with? (At most $\frac{2}{3}N + \frac{2}{3}$. Why?)
   2. After the second step?
   3. After the **j**th step:

$$\left(\frac{2}{3}\right)^{j} + 2$$

   4. After $\log N$ steps, we're left with $\frac{2}{3}^{\log N} N + 2 = 3$ numbers, so we need $\log N + 1$ steps to reduce to two $k + \log N$-bit numbers.

6. Use a carry lookahead adder to reduce the two-number redundant representation to a single, non-redundant number.

PROPOSED TECHNIQUE:

PRE-COMPUTE BITWISE ADDITION FOLLOWED BY CARRY PREFIX COMPUTATION This method presents a new adder technique and its VLSI architecture to perform the three-operand addition in modular arithmetic. The proposed adder technique is a parallel prefix adder. However, it has four-stage structures instead three-stage structures in prefix adder to compute the addition of three binary input operands such as bit-addition logic, base logic, PG (propagate and generate) logic and sum logic. The logical expression of all these four stages are defined as follows,

Stage-1: Bit Addition Logic:

$$S_i' = a_i \oplus b_i \oplus c_i,$$
$$cy_i = a_i \cdot b_i + b_i \cdot c_i + c_i \cdot a_i$$

Stage-2: Base Logic:

$$G_{i:i} = G_i = S_i' \cdot cy_{i-1}, \quad G_{0:0} = G_0 = S_0' \cdot C_{in}$$
$$P_{i:i} = P_i = S_i' \oplus cy_{i-1}, \quad P_{0:0} = P_0 = S_0' \oplus C_{in}$$

Stage-3: PG (Generate and Propagate) Logic:

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j},$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

Stage-4: Sum Logic:

$$S_i = (P_i \oplus G_{i-1:0}), \quad S_0 = P_0, \; C_{out} = G_{n:0}$$

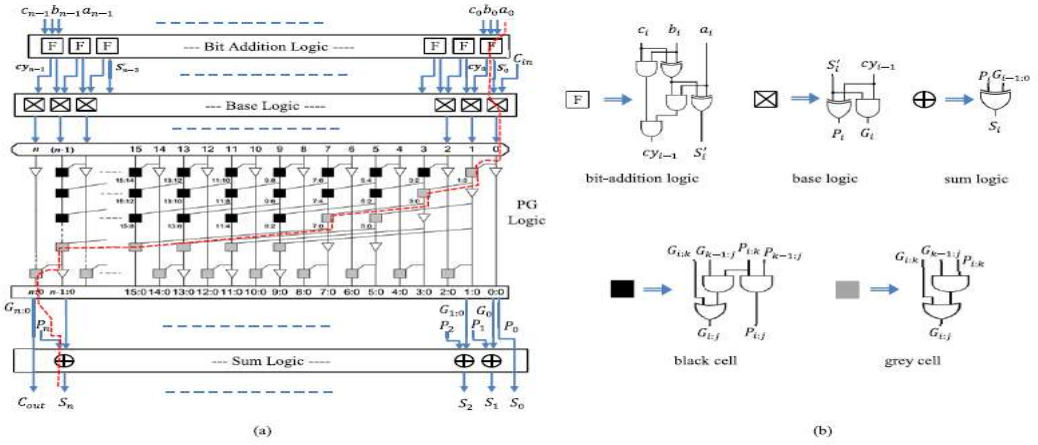

Fig7. Proposed three-operand adder; (a) First order VLSI architecture, (b) Logical diagram of bit addition, base logic, sum logic, black-cell and grey-cell.

DESIGN METHODOLOGY:

The proposed VLSI architecture of the three-operand binary adder and its internal structure is shown in Fig. The new adder technique performs the addition of three n-bit binary inputs in four different stages. In the first stage (bit-addition logic), the bitwise addition of three n-bit binary input operands is performed with the array of full adders, and each full adder computes "sum (S_ i)" and "carry (cyi)" signals as highlighted in Fig. 3(a). The logical expressions for computing sum (S_ i) and carry (cyi) signals are defined in Stage-1, and the logical diagram of the bit-addition logic is shown in Fig. 3(b). In the first stage, the output signal "sum (S_ i)" bit of current full adder and the output signal "carry" bit of its right-adjacent full adder are used together to compute the generate (Gi) and propagate (Pi) signals in the second stage (base logic). The computation of Gi and Pi signals are represented by the "squared saltire-cell" as shown in Fig. 3(a) and there are n+1 number of saltire-cells in the base logic stage. The logic diagram of the saltire-cell is shown in Fig. 3(b), and it is realized by the following logical expression,

$$G_{i:i} = G_i = S_i' \cdot cy_{i-1};$$
$$P_{i:i} = P_i = S_i' \oplus cy_{i-1}$$

The external carry-input signal (Cin) is also taken into consideration for three-operand addition in the proposed adder technique. This additional carry-input signal (Cin) is taken as input to base logic while computing the G0 (S_ 0 · Cin) in the first saltire-cell of the base logic. The third stage is the carry computation stage called "generate and propagate logic" (PG) to pre-compute the carry bit and is the combination of black and grey cell logics. The logical diagram of black and grey cell is shown in Fig. 3(b) that computes the carry generate Gi: j and propagate Pi: j signals with the following logical expression,

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j},$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

The number of prefix computation stages for the proposed adder is (log2 n+1), and therefore, the critical path delay of the proposed adder is mainly influenced by this carry propagate chain.

The final stage is represented as sum logic in which the "sum (Si)" bits are computed from the carry generate Gi: j and carry propagate Pi bits using the logical expression, Si = (Pi _ Gi−1:0). The carryout signal (Cout) is directly obtained from the carry generate bit Gn:0.

A single full adder performs the addition of two one bit numbers and an input carry. But a **Parallel Adder** is a digital circuit capable of finding the arithmetic **sum** of two binary numbers that is **greater than one bit** in length by operating on corresponding pairs of bits in parallel. It consists of **full adders connected in a chain** where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain. **A n bit parallel adder requires n full adders to perform the operation.** So for the two-bit number, two adders are needed while for four bit number, four adders are needed and so on. Parallel adders normally incorporate carry lookahead logic to ensure that carry propagation between subsequent stages of addition does not limit addition speed.

Results

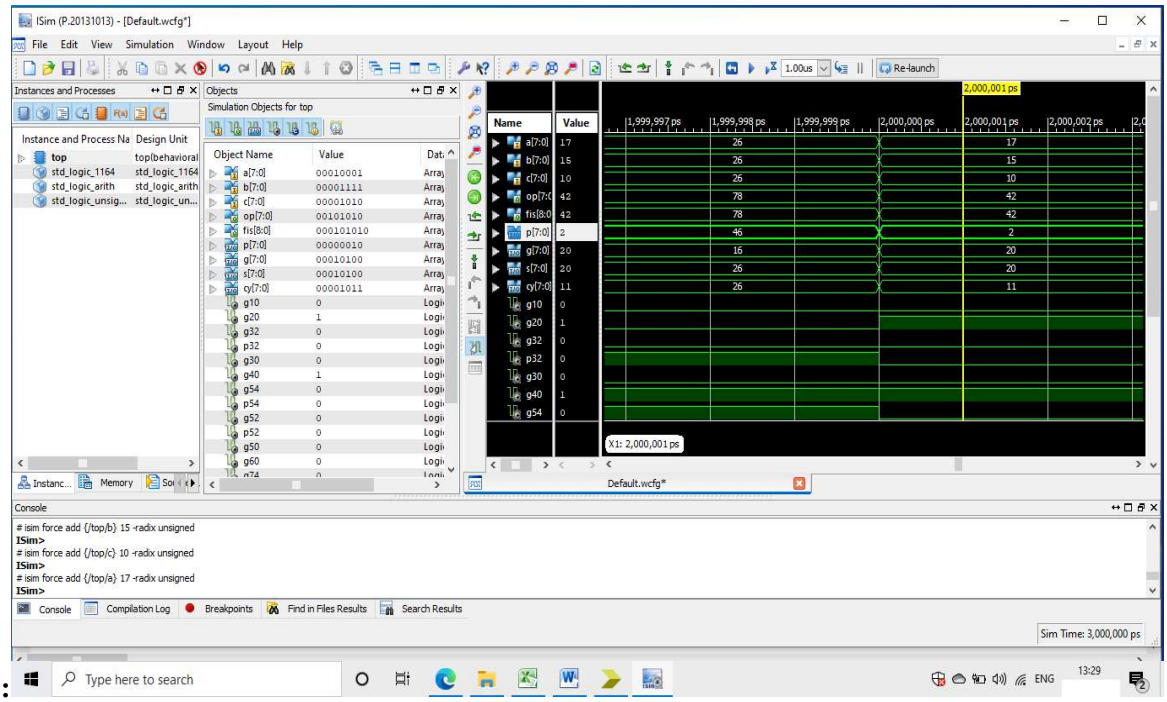

Fig a: simulation output1

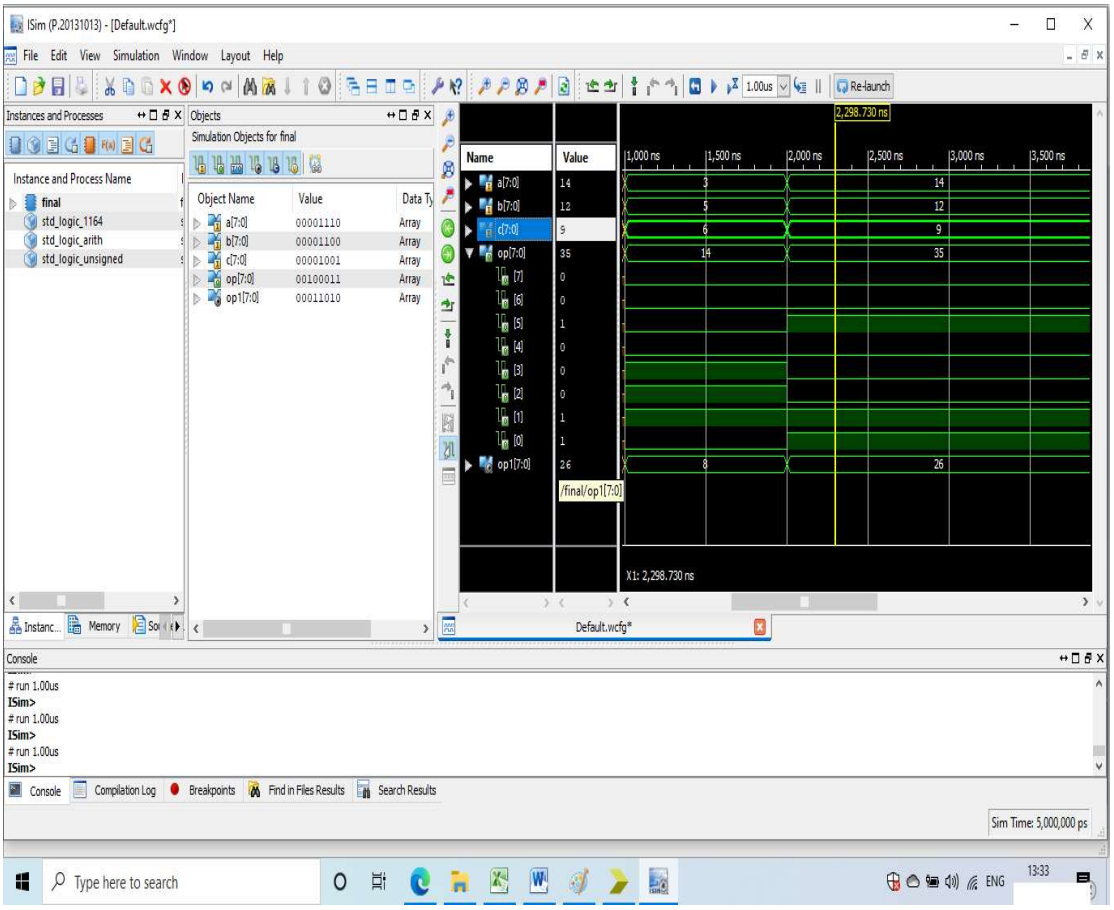

Fig:b simulation output2

ADVANTAGES

1. The parallel adder/subtractor performs the addition operation faster as compared to serial adder/subtractor.
2. Time required for addition does not depend on the number of bits.
3. The output is in parallel form i.e all the bits are added/subtracted at the same time.
4. It is less costly.
5. Low gate density

APPLICATIONS:

- Adders & Subtractors are wildly used in in computer's ALU (Arithmetic logic unit) to compute addition as well as CPU (Central Processing unit) and GPU (Graphics Processing unit) for graphics applications to reduce the circuit complexity.
- Adder and subtractor are basically used for performing arithmetical functions like addition, subtraction, multiplication and division in electronic calculators and digital instruments.
- Adders are used in digital calculators for arithmetic addition and devises that uses some kind of increment or arithmetic process
- They are also used in microcontrollers for arithmetic additions, PC (program counter) and timers.
- It is also used in processors to calculate address, tables and slimier operations
- It is also used in networking and DSP (Digital signal processor) oriented system

CONCLUSION:

In this paper, a high-speed area-efficient adder technique and its VLSI architecture is proposed to perform the three operand binary addition for efficient computation. The proposed three-operand adder technique is a parallel prefix adder that uses four-stage structures to compute the addition of three input operands. The novelty of this proposed architecture is the reduction of delay and area in the prefix computation stages in PG logic and bit-addition logic.

FUTURE SCOPE:

In the future work, further research will be done by applying dual-path FP architecture to the three-operand FP adder and using other redundant FP representations. Use of improved techniques in the termination phase of the design (i.e., redundant LZD, normalization, and rounding) would lead to faster architectures, though higher area costs are expected.

REFERENCES:

[1] S. Yu and E. E. Swartzlander, "DCT implementation with distributed arithmetic", IEEE Transactions on Computers, vol. 50, no. 9, pp. 985–991, Sept. 2001.

[2] T.-S. Chang, C. Chen, and C.-W. Jen, "New distributed arithmetic algorithm and its application to IDCT," IEE Proceedings Circuits, Devices and Systems, vol. 146, no. 4, pp. 159–163, Aug. 1999.

[3] T.-S. Chang and C.-W. Jen, "Hardware-efficient implementations for discrete function transforms using LUT-based FPGAs," IEE Proceedings Circuits, Devices and Systems, vol.146, no. 6, pp. 309–315, Nov. 1999.

[4] F. de Dinechin, H. D. Nguyen and B. Pasca, Pipelined FPGA Adders, LIP Research Report no. ensl00475780, Apr. 2010.

[5] J. Hormigo, M. Ortiz, F. Quiles, F. J. Jaime, J. Villalba and E.L. Zapata, Efficient Implementation of CarrySave Adders in FPGAs, 20th IEEE international Conference on Application-Specific Systems, Architectures and Processors, pp. 207–210, Jul. 2009.

[6] P. M. Martinez, V. Javier, and B. Eduardo, On the design of FPGA-based Multioperand Pipeline Adders, XII Design of Circuits and Integrated System Conference, 1997.

[7] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient Synthesis of Compressor Trees on FPGAs," in Asia and South Pacific Design Automation Conference (ASPDAC). IEEE, 2008, pp. 138–143. [8] Xilinx Inc., Virtex-6 User Guide, 2009, http://www.xilinx.com/. [9] S. Xing and W. H. Yu, FPGA Adders: Performance Evaluation and Optimal Design, IEEE Design and Test of Computers, vol. 15, no. 1, pp. 24–29, Jan.- Mar. 1998.

[10] 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.

[11] 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 ApplicationSpecific Systems, Architecture Processors (ASAP'05), pp. 322-327, 2005

[12] M. Ortiz, F. Quiles, J. Hormigo, F. J. Jaime, J. Villalba, and E. L. Zapata, "Efficient Implementation of CarrySave Adders in FPGAs," in IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP), 2009, pp. 207– 210.

[13] W. Kamp, A. Bainbridge-Smith, and M. Hayes, "Efficient Implementation of Fast Redundant Number Adders for Long Word- Lengths in FPGAs," in 2009 International Conference on Field- Programmable Technology (FPT). IEEE, 2009, pp. 239–246. [14] J. Hormigo, J. Villalba, and E. L. Zapata, "Multioperand Redundant Adders on FPGAs," submitted to IEEE Transactions on Computers, vol. 62, no. 10, pp. 2013– 2025, 2013.

[15] S. D. Thabah; M. Sonowal and P. Saha ,"EXPERIMENTAL STUDIES ON MULTI-OPERAND ADDERS", INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS VOL. 10, NO. 2, JUNE 2017

[16] S. Singh and D. Waxman, "Multiple Operand Addition and Multiplication", IEEE Transactions on Computers, vol. C-22, no. 2, pp. 113-120, Feb. 1973. [17] C. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, no. 1, pp. 14– 17, 1964.

[18] L. Dadda, "Some Schemes For Parallel Multipliers," Alta Frequenza, vol. 45, no. 5, pp. 349–356, 1965.

[19] A. Omondi and B. Premkumar, Residue Number Systems: Theory and Implementation. Imperial College Press, 2007.

[20] A. R. Meo, "Arithmetic Networks and Their Minimization Using a New Line of Elementary Units," submitted to IEEE Transactions on Computers and currently under review, vol. C-24, no. 3, pp. 258– 280, 1975.

[21] K.A.C. Bickerstaff, M. Schulte, and E.E. Swartzlander, "Reduced area multipliers," in Application-Specific Array Processors, 1993

[22] Suhas B. Shirol, S. Ramakrishna and Rajashekar B. Shettar, "Design and Implementation of Adders and Multiplier in FPGA Using ChipScope: A Performance Improvement", Information and Communication Technology for Competitive Strategies pp 11-19, 31 August 2018

[23] Duncan J. M. Moss , David Boland, and Philip H. W. Leong, "A Two-Speed, Radix-4, Serial–Parallel Multiplier", IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, (Volume: 27 , Issue: 4 , April 2019) Page no. 769 – 777.

[24] Martin Kumm and Johannes Kappauf." Advanced Compressor Tree Synthesis for FPGAs", IEEE Transactions on Computers (Volume: 67 , Issue: 8 , Aug. 1 2018).