ISSN 2349-6193
Impact Factor: 3.866

IJESMR

International Journal OF Engineering Sciences &Management Research

PERFORMANCE ESTIMATION OF VARIOUS ADDERS USED IN ARITHMETIC PROCESSORS

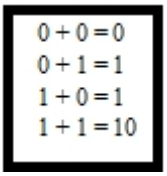
Dr. R. Prakash Rao[*1] & Dr. K. Naveen[2]
[*1&2]Professor, St.Peter's Engineering College, Near Forest Academy, Dulapally, Hyderabad

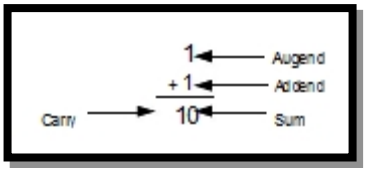
---

Keywords: *Arithmetic and Logic Unit, CPU, subtraction, addition, multiplication, increment / decrement, processor.*

ABSTRACT
Processors are two types arithmetic processors and signal processing processors. Arithmetic and Logic Unit(ALU) is used in the arithmetic processors where as Multiplier Accumulator Content (MAC) is used in signal processing processors like DSP processors. In ALU adders play a major role not only for addition but also in performing many other basic arithmetic operations like subtraction, multiplication, increment / decrement etc. Thus, realizing an efficient adder is required for better performance of a processor and ALU in particular [1,2].

INTRODUCTION
Research into design of efficient adder algorithms for hardware implementation of Very Large Scale Integrated (VLSI) arithmetic circuits started in late 1950s. Many designs based on serial and parallel structures have been proposed to optimize different parameters from time to time [3]. Binary addition consists of four possible elementary operations, which are





The first three operations produce only a 'Sum' whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a 'Carry'. A combinational circuit that performs the addition of two bits is called a half-adder while the one that performs the addition of three bits is known as a full-adder [4].

REVIEW OF EXISTING ADDER DESIGNS
Adders can be broadly classified into following four classes [5]:

- Ripple Carry Adder (RCA)
- Carry Select Adder (CSA)
- Carry Look-Ahead Adder (CLA)
- Parallel Prefix- based Adder (PPA)

Ripple Carry Adder
A full adder (FA) is a combinational circuit that takes two operand bits and a carry bit, say A,B and Ci respectively, as inputs and gives Sum (S) and Carry bit (Co) as outputs. This output Carry bit Co will serve as input Carry bit for the successive full adder. The combinational circuit follows the Boolean equations to implement a full adder and the gate level implementation of the same is shown in Fig 1. A simple implementation of higher operand adder for two operands A and B is carried out by cascading n of these basic full adder units and is known as a ripple carry adder.
A simple 4-bit ripple carry adder is shown in Fig 2. The design of this adder is simple and implementation is

easy, but it suffers from serious delay issues. This is because the next stage full adder needs to wait for Carry bit from the previous stage full adder. By inspecting the FA shown in Fig 1 it can be observed that the gate delay from C_{in} to C_o is 2 gates. Therefore, each full adder contributes to a 2-gate delay in the process of rippling the carry [5].

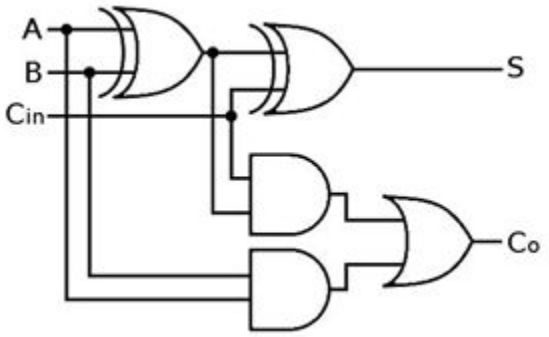


Figure 1 One - Bit Full Adder

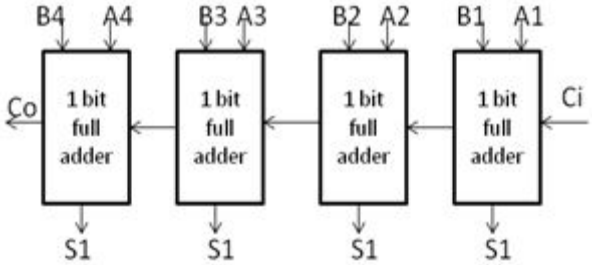


Figure 2 Four-Bit Ripple Carry Adder

Carry Select Adder (CSA)

Ripple carry adder waits for the input carry (Ci) and then computes the 'Sum' and the Carry out (Co) thus increasing its delay. In order to reduce the delay, carry select adder is introduced, which pre-computes the 'Sum' and 'Co' for the two possible cases i.e. Ci = 0 and Ci = 1. The calculated Sum is given to a multiplexer, which chooses the correct output depending upon the Ci coming from the previous stage. This pre-computation of Sum reduces the delay of rippling of Carry which is limited to only one multiplexer for each stage. If the gate level diagram of a 16- bit carry select adder is considered, in this, each 4-bit adder is a bit ripple carry adder. Carry select adder uses more hardware even though it gives less delay compared to ripple carry adder. Thus, there is a tradeoff between area, power and delay between different adders [4].

Carry Look-Ahead Adder

Various techniques have been proposed from time to time to decrease the overall delay in parallel addition [5]. One such technique is to derive the 'Sum' and 'Carry' outputs by using intermediate terms defined as 'Generate (G)' and 'Propagate (P)' terms [5-6]. Generate term produces a carry-out independent of the carry-in, i.e. no matter what the carry-in is, the carry-out is always '1', when both of its inputs A and B are '1' thus G = A.B. The Propagate term transfers the input Carry as output Carry when only one of the propagation term is defined

as $P=A \oplus B$.

Table.1 and the example shown below illustrate the concept of Propagate and Generate more clearly. In the Propagate case the 'Carry-out' depends on the 'Carry-in', i.e. when 'Carry-in' is 0 'Carry-out' is 0 and when 'Carry-in' is 1 'Carry-out' is 1 and in the case of Generate, no matter what the 'Carry-in' is 'Carry-out' is always 1.

ISSN 2349-6193
Impact Factor: 3.866

IJESMR
International Journal OF Engineering Sciences &Management Research

A	B	Cin	Sum	Cout	
0	0	0	0	0	G=P=0
0	0	1	1	0	Cout = 0 (no dependence on Cin)
0	1	0	1	0	
0	1	1	0	1	P=1
1	0	0	1	0	Cout = Cin
1	0	1	0	1	
1	1	0	0	1	
1	1	1	1	1	G=1 Cout = 1 (no dependence on Cin)

Table .1 Truth Table of a Full Adder

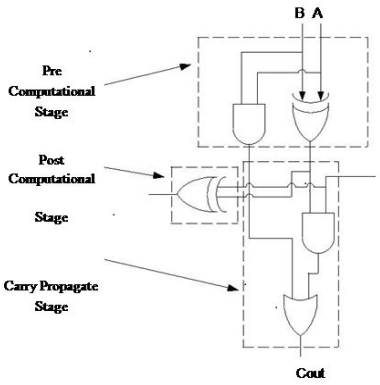
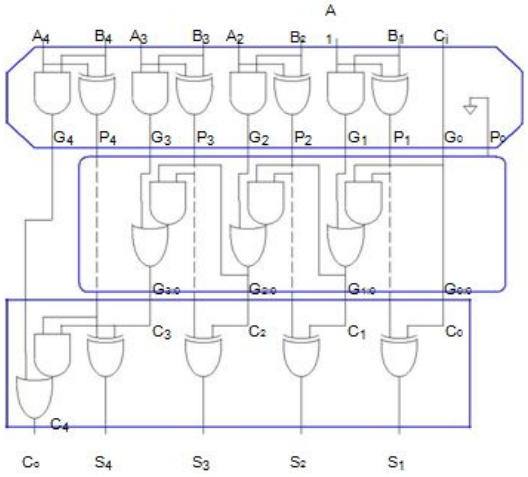


Figure 3 One-bit Full Adder with Carry Propagate and Generate

Figure 3 above illustrates the implementation of One-bit Full Adder with Carry Propagate and Generate which is essentially same as Fig 1 but derived from Table 1. This logic is also called carry look-ahead logic. For each bit in a binary sequence to be added, the carry look-ahead logic will determine whether that bit pair will generate or propagate a Carry. This allows the circuit to "pre-process" two numbers being added to determine the carry ahead of time. Thus, when the actual addition is performed, there is no delay from waiting for the ripple carry effect (time it takes for the carry from the first full adder to be passed on to the last Full Adder) [6].



ISSN 2349-6193
Impact Factor: 3.866

IJESMR

International Journal OF Engineering Sciences &Management Research

Figure 4 Ripple Carry Adder with Carry Propagate and Generate

The carry look-ahead type implementation of a ripple carry adder is shown in Fig 4. It can be seen from this figure that the carry propagation stage determines the critical path that determines the delay. To increase the speed of an adder, this stage has to be redesigned for fast carry propagation.

Keeping this in mind, Weinberger and Smith proposed a method for fast carry generation which states that the carry need not depend explicitly on the previous carry, but can be expressed as a function of only the relevant addend and augend digits and some lower order carry [7].

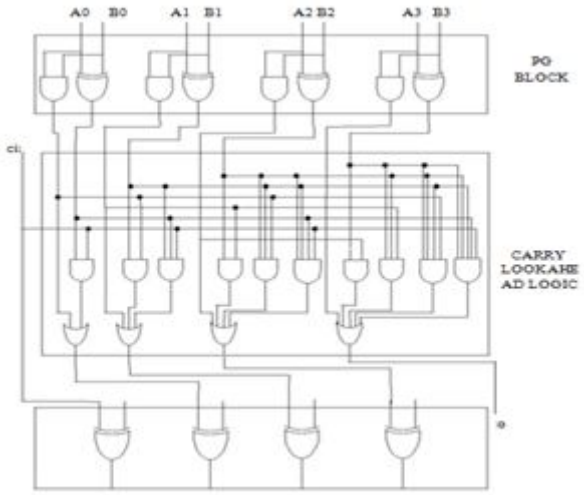


Figure 5 4-bit Weinberger-Smith CLA

A typical 4-bit CLA implementing the above equations is shown in Fig 5. For wide adders where N > 16 (N is the input operand size), the delay of the carry look-ahead adders becomes dominated by the delay of passing the carry through the look-ahead stages and the implementation needs high fan-in gates [8]. To overcome these problems, a new breed of networks has been designed that pass the carry through the look ahead stage in around log(N) stages. These networks are called Tree Networks and the adders that utilize these networks are called tree- adders or prefix- adders [9]. There are many ways to build the tree adders which offer tradeoffs among parameters like, the number of stages of logic, number of logic gates, the maximum fan-out of each gate and the amount of wiring between the stages.

Prefix Based Adders

A prefix adder consists of 3 stages i.e, pre-computation stage, prefix network stage and post-computation stage as shown in Fig 6 [10].

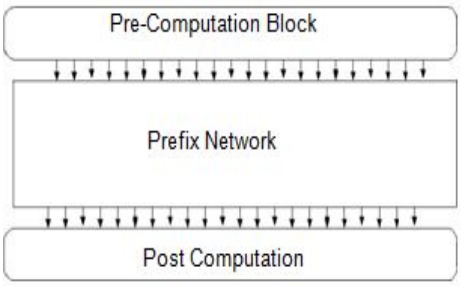


Figure 6 Block level diagram of a prefix adder

The pre-computation stage computes the carry ‘Propagate’ and carry ‘Generate’ bits for each input pair.

One of the prefix networks, Kogge-Stone [9] represented as a graph is shown in Fig. 7. The white color node in the graph represents a feed through node with no logic (generally realized with a buffer in hardware).

Figure 7 Example of a Kogge –Stone prefix adder

The final post computation stage computes the final Sum from carry generated in prefix network stage. These designs are very efficient in terms of delay and area when compared to carry-select and carry look-ahead adders. As operand size increases (32-bits and above) these prefix adders suffer from complexity in prefix network due to an increase in number of logic cells and wiring [9]. This problem can be addressed with a hybrid adder (also called as a sparse adder) which is a combination of prefix and carry-select adders [12]. These adders consist of two segments, one being carry generation block (CGB) that has prefix network and the other the conditional sum computation block (SCB) that has carry-select adders shown in Fig 2.9. As seen from the figure in CGB, where a Kogge-Stone network structure is used, all 'carry's are not computed as in prefix adders (shown in Fig. 2.8) but only a few (in this case C3 and C7) are computed depending on the degree of sparseness where the degree of sparseness is the number of sums selected conditionally.

DESIGN AND IMPLEMENTATION OF EFFICIENT SUM COMPUTATION BLOCK FOR HIGHER BIT SPARSE ADDERS

As discussed in section 2.2.4 earlier, the power, area and fan-out overheads limit the usage of carry-select adder in SCB as the degree of sparseness increases. The area overhead can be reduced by using prefix structure with late Carry-in concept proposed by Sklansky [15]. This late Carry-in concept or fastest input-carry processing is achieved by adding an extra row of node ● at the end of the prefix carry network as shown in Fig 8. This addition of extra node ● however increases the overall delay of the adder by one node stage. Any prefix structure can be preferred to implement the prefix carry network depending on the design requirement. However, the fan-out or loading on the Carry signals from the CGB is still a problem. In this work, the structure of prefix network and the late carry-in scheme are analyzed and a new structure is developed to address these problems.

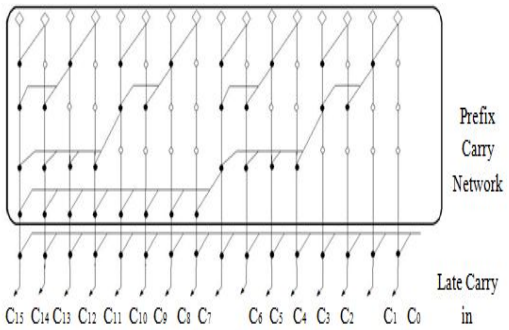


Figure 8 Sklansky parallel prefix adder with late Carry-in

The proposed approach is to reduce the fan-out or loading on the late carry-in signals. It is achieved by feeding the late Carry-in signal only for a few 'carry's. The remaining 'carry's are to be computed with these few 'carry's to generate all the 'carry's required for sum computation. The proposed technique is illustrated through an example by taking Han-Carlson prefix structure. This prefix structure is chosen because of its uniformity in fan-in and fan-out requirements as well as reduced number of nodes when compared to other prefix structures [16].

A 16-bit traditional Han-Carlson adder with late carry-in is shown in Figs 9(a). From the figure, it can be seen that the loading on the C_{in} signal is 16. To reduce this loading, the proposed technique is applied to this structure

ISSN 2349-6193
Impact Factor: 3.866

IJESMR

International Journal OF Engineering Sciences &Management Research

wherein the late carry-in signal (Cin) is fed only to odd 'carry's i.e., G1:0, G3:0, etc… Even 'carry's are than computed from the odd 'carry's. The modified Han-Carlson with late carry-in using the proposed technique is shown in Fig. 9 (b). It can be observed that a 16-bit modified adder has a fan-out requirement of only 9 compared to the traditional late carry-in prefix structure that has a fan-out of 16. Thus, the proposed technique results in the reduction of fan-out on the late carry-in signal.

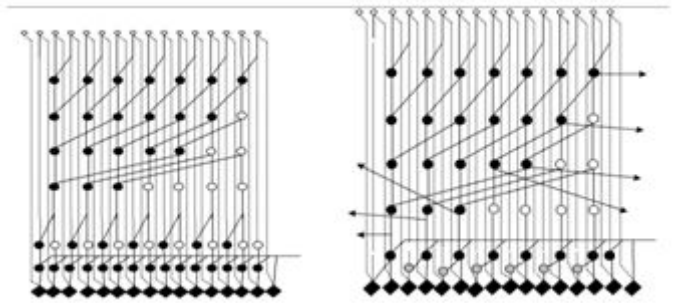


Figure 9 (a) 16-Bit Han-Carlson Adder with late Carry-in (b) Modified 16-Bit Han-Carlson Adder with late Carry-in.

SIMULATION RESULTS

All adders have been described in Verilog HDL and simulated using Cadence Incisive Unified Simulator (IUS) v6.1 and are mapped on to the Synopsys 90*nm* generic Technology library, using Cadence RTL Compiler v7.1. The derived netlist was then passed to Cadence First Encounter XL v7.1 for floor-planning and routing.

The modified Han-Carlson Sum computation block for 64-bit Sparse-8, and -16 has been compared with 64-bit Sparse-8, and -16 Han-Carlson late Carry-in adder and 64-bit sparse-4 with conditional Sum adder [17]. Table 2.2 presents performance parameters such as area, power, delay and power-delay product for all the two designs.

64-Bit adder	Sparse -8 with Han Carlson late Carry in	Sparse -8 with Modified Han Carlson late Carry in	Sparse 16 with Han Carlson late Carry in	Spars-16 with Modified Han Carlson late Carry in
Area (um^2)	4739	2998	4257	3383
Power(mW)	0.281	0.3471	0.409	0.3627
Delay(ns)	0.956	0.869	1.05	1.03
Power-Delay	0.2686	0.1394	0.2245	0.1706

CONCLUSIONS

In this work, novel designs for higher bit (64) sparse adders have been proposed. The increased complexity of the sum computation block at larger bit lengths have been compensated with alternate designs of carry generation block that results in reduced complexity. A detailed analysis of the 64 -bit sparse adders with different degree of sparseness indicates that they perform better than the designs reported in existed work..

REFERENCES

1. *B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", 2nd edition, Oxford University Press, New York, 2010*
2. *Milos Ercegovac, Tomas Lang, "Digital Arithmetic", Morgan Kaufman, 2004.*
3. *I . Koren, Computer Arithmetic Algorithms. Englewood Cliffs, NJ, Prentice Hall, 1993.*

ISSN 2349-6193
Impact Factor: 3.866

IJESMR
International Journal OF Engineering Sciences &Management Research

4. *R. Zimmermann, Binary Adder Architectures for Cell-Based VLSI and their Synthesis, PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Hartung-Gorre Verlag, 1998.*
5. *Neil Weste, David Harris, "CMOS VLSI Design: A Circuits and Systems Perspective" Pearson Education, 2004.*
6. *Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, "Digital Integrated. Circuits: A Design Perspective" Pearson Education, 2003.*
7. *Weinberger, J.L. Smith, "A Logic for High-Speed Addition," Nat. Bur. Stand. Circ., 591:3-12, 1958.*
8. *H. Ling, "High-Speed Binary Adder," IBM Journal of Research and Development, vol. 25, no.3, pp. 156-166, May 1981*
9. *Harris D, "A taxonomy of parallel prefix networks", in. Proc. 37th Asilomar Conf. Signals, Systems, and. Computers, Nov. 2003, Vol. 2, pp. 2213-2217.*
10. *R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," IEEE Trans. Computers, vol. 31, no. 3, pp. 260-264, Mar. 1982.*
11. *D. Harris, " Logical Effort of Higher Valency Adders," in Proc. 38th Asilomar Conf. Signals, Systems, and Computers, vol. 2, pp. 1358 - 1362, 2004.*
12. *S. Mathew, M.A. Anders, B. Bloechel, T. Nguyen, R.K. Krishnamurthy, S. Borkar, "A 4GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90nm CMOS," IEEE J. Solid State Circuits, vol. 40, no.1, pp. 44-51, Jan. 2005.*
13. *Kumashikar et al., "Sparse Tree Adder", US Patent 20070299902A1.*
14. *T.L. Lynch, E.E. Swartzlander, "A Spanning Tree Carry Lookahead Adder", IEEE Trans. Comput., Vol.41, N°8, pp.931-939, 1992.*
15. *V. Kantabutra, "A Recursive Carry-Lookahead/Carry-Select Hybrid Adder", IEEE Trans. Comput., Vol.42, N°12, pp.1495-1499, 1993.*
16. *O. Kwon, E. Swartzlander, and K. Nowka, "A fast hybrid Carry-lookahead/Carry-select adder design", Proc. of the 11th Great Lakes symposium on VLSI, pp.149-152, March 2001.*
17. *Oklobdzija, V.G.; Zeydel, B.R.; Dao, H.; Mathew, S.; Ram Krishnamurthy; , "Energy-delay estimation technique for high-performance microprocessor VLSI adders," Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on , vol., no., pp. 272- 279, 15-18 June 2003.*