ISSN: 2249-7196
IJMRR/April-June 2025/ Volume 15/Issue 2/427-434
K. Sandhya / International Journal of Management Research & Review

A Novel Architecture for a High-Speed Multiply and Accumulate Circuit Using a Three-Operand Binary Adder

Semal Sekhar Mummana[1], T. Pattalu Naidu[2], K. Sandhya[3], K. Sai Srinivasu[4], T. Ramya[5], R. Anusha[6]

[1]Associate Professor, Department of Electronics and Communication Engineering

[2]Assistant Professor, Department of Electronics and Communication Engineering

[3,4,5,6]UG Student, Department of Electronics and Communication Engineering

Avanthi Institute of Engineering & Technology, Tamaram, Makavarapalem, Anakapalli District – 531113, Andhra Pradesh, India

Abstract

The three-operand binary adder is a core element in cryptographic and pseudorandom bit generator algorithms, where it performs the modular arithmetic those algorithms depend on. The carry-save adder (CS3A) is the technique normally used for this, but its ripple-carry second stage gives a propagation delay of O(n), so the delay grows linearly with word length. Using a parallel prefix two-operand adder such as Han-Carlson (HC3A) in two stages reduces the critical path but costs additional hardware. This paper implements an alternative architecture that performs three-operand addition through pre-computed bitwise addition followed by carry-prefix computation logic, reducing the adder delay to O($\log_2$ n) while using less area and dissipating less power than HC3A. The four-stage structure comprises bit-addition logic, base logic, propagate-generate logic and sum logic. A 16-bit version of the adder was described in Verilog, and used to build a multiply-and-accumulate (MAC) unit with 8-bit inputs and a 16-bit output, the arrangement used in digital filters and neural network accelerators, where the same multiply-add is executed repeatedly and its latency sets the throughput of the whole datapath. The design was implemented and simulated in Xilinx tools, and the synthesis results show it achieving the lowest area-delay and power-delay products among the three-operand adder techniques compared.

Index Terms— *Carry-save adder, FPGA, Han-Carlson adder, multiply and accumulate, parallel prefix adder, pseudorandom bit generator, three-operand adder, VLSI, Verilog.*

I. Introduction

Adders are fundamental to computer arithmetic. Binary adders appear in microprocessors for addition, subtraction and floating-point multiplication and division, so improving their performance is a substantial concern in digital design. Theoretical work has established minimum bounds on the area and delay of an n-bit adder: area grows linearly with adder size, while delay follows O($\log_2$ n). High-speed adders use parallel-prefix architectures including Brent-Kung, Kogge-Stone, Sklansky, Han-Carlson, Ladner-Fischer and Knowles.

Area, power and delay are the three quantities that matter in any VLSI circuit, and there is generally a trade-off between them. The speed of an adder is set by how long the carry takes to propagate, so carry generation defines the speed limit of the circuit. Since the datapath consumes close to a third of total power and adders are a large part of the datapath, an efficient adder reduces power consumption meaningfully as well as latency.

A. Three-Operand Addition

Three-operand binary addition is central to congruential modular arithmetic architectures and to LCG-based pseudorandom bit generator methods such as CLCG, MDCLCG and CVLCG. It can be performed either through two stages of two-operand adders or through a single three-operand adder stage. Because the three-operand adder is the principal component in these architectures, its critical path delay determines the latency of the whole cryptographic or PRBG design, which is what makes the problem worth attention.

II. Existing Techniques

A. Carry-Save Adder

The carry-save adder is the prevalent technique for three-operand addition. It computes in two stages. The first is an array of full adders, each calculating a carry bit and a sum bit concurrently from the three input bits a_i, b_i and c_i. The second is a ripple-carry adder producing the final n-bit sum and the one-bit carry-out. The architecture is shown in Fig. 1.

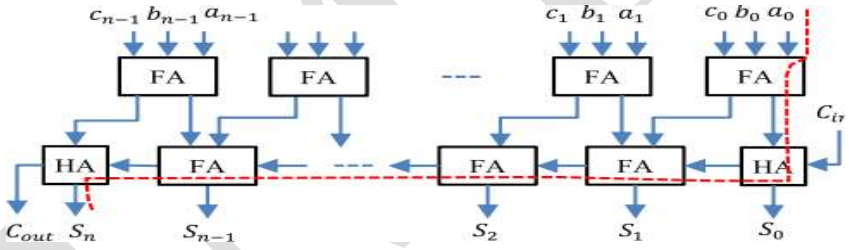

Fig. 1. Three-operand carry-save adder (CS3A).

The critical path delay depends on the carry propagation through the ripple-carry stage:

$$T\_CS3A = (n + 1)\ T\_FA = 3T\_X + 2n\ T\_G \quad (1)$$

and the total area is:

$$A\_CS3A = 2n\ A\_FA = 4n\ A\_X + 6n\ A\_G \quad (2)$$

where A_G and T_G are the area and propagation delay of a basic 2-input gate, and A_X and T_X those of a 2-input XOR gate. The limitation of CS3A is visible in (1): the carry-out signal must propagate through all n full adders of the ripple stage, so delay rises linearly with bit length and the design becomes progressively worse as word width increases.

B. Han-Carlson Based Three-Operand Adder

To reduce the critical path, two stages of a parallel prefix two-operand adder can be used instead. Parallel prefix adders are the fastest two-operand methods available, spanning six topologies: Brent-Kung, Sklansky, Knowles, Ladner-Fischer, Kogge-Stone and Han-Carlson. Of these, Han-Carlson is the fastest once bit size exceeds 16.

Other parallel prefix designs have been proposed, including Ling, Jackson-Talwar, the ultra-fast adder, hybrid PPFCSL and hybrid Han-Carlson. The ultra-fast adder is reported as the fastest, beating Han-Carlson by three gate delays, but it requires roughly twice the gate area. The hybrid Han-Carlson design places Brent-Kung stages at the start and end with Kogge-Stone stages in the middle, giving two gate delays more than Han-Carlson but reducing gate complexity by 10 to 18 per cent. Han-Carlson therefore delivers high speed at low gate complexity relative to the alternatives, with the smallest area-delay and power-delay products among two-operand techniques. Three-operand addition using two Han-Carlson stages is denoted HC3A, and its delay depends on the number of black-grey cell stages in the propagate-generate logic.

III. Parallel Prefix Adders

A parallel prefix adder performs addition in three steps. Pre-processing computes the carry generate and carry propagate signals from the input bits. Prefix computation calculates all carry signals in parallel through a tree network. Post-processing evaluates the final sum. The tree network is what reduces latency to $O(\log_2 n)$, where n is the number of bits.

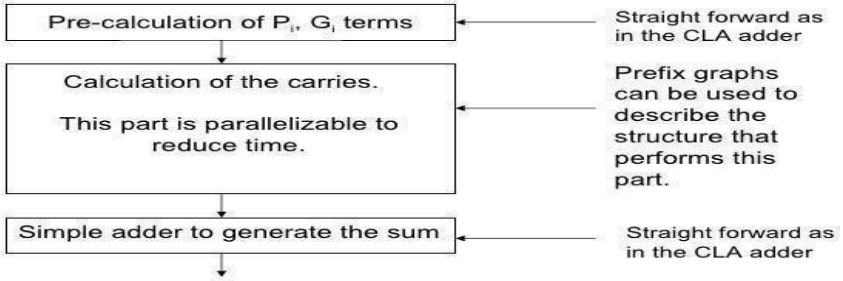

Fig. 2. The mechanism of parallel prefix adders.

The generate and propagate signals for bit i are:

$$G_i = A_i \cdot B_i\,, \quad P_i = A_i \oplus B_i \qquad (3)$$

and the group signals combine as:

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \qquad (4)$$

$$P_{i:j} = P_{i:k} \cdot P_{k-1:j} \qquad (5)$$

with the final carry and sum given by $C_i = G_i$ and $S_i = P_i \oplus G_{i-1}$. The carry generation network is built from two cell types: the black cell, which computes both group generate and group propagate, and the grey cell, which computes only group generate and is used where the propagate output is not needed further along the tree. Using grey cells wherever the propagate signal is redundant is one of the standard ways of trimming area from a prefix network.

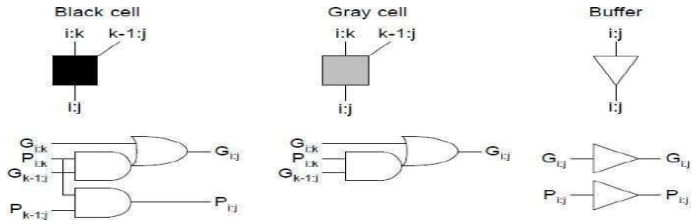

Fig. 3. Black cell and grey cell structures.

The topologies differ in how they arrange this tree. Sklansky offers minimum depth at the cost of high fan-out on certain nodes. Kogge-Stone has both optimal depth and low fan-out but produces complex circuits with many interconnects. Brent-Kung uses fewer computation nodes but has maximal depth and so higher latency. Ladner-Fischer sits slightly above Sklansky in depth while reducing maximum fan-out on the critical path. Han-Carlson combines Brent-Kung and Kogge-Stone to trade off logic depth, interconnect count and node count. No single topology is best at every word width, which is why the choice depends on the bit size the design targets.

IV. Proposed Three-Operand Adder

The proposed architecture uses pre-computed bitwise addition followed by carry-prefix computation logic. Where a conventional prefix adder uses three stages, this design uses four to handle three input operands. The basic architecture is shown in Fig. 4.

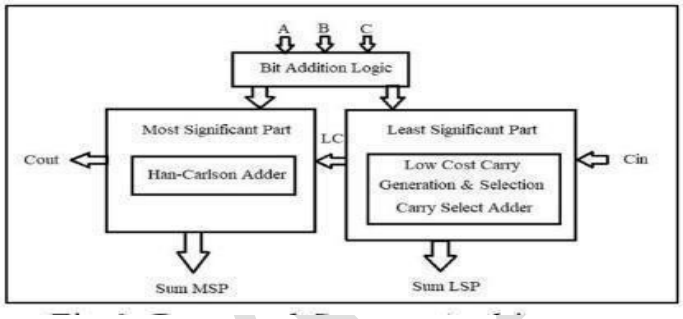

Fig. 4. Architecture of the three-operand adder.

A. Stage 1: Bit-Addition Logic

Each F block in the first stage takes the three input bits a_i, b_i and c_i and produces a sum and a carry:

$$S_i' = a_i \oplus b_i \oplus c_i \quad (6)$$

$$Cy_i = a_i \cdot b_i + b_i \cdot c_i + c_i \cdot a_i \quad (7)$$

This stage reduces the three operands to two vectors before any carry propagation begins, which is what allows the remaining stages to use a two-operand prefix structure.

B. Stage 2: Base Logic

The base logic takes the sum and carry vectors and computes the propagate and generate signals. Each cell takes the sum bit of the present position and the carry output of the previous position:

$$G_{i:i} = G_i = S_i' \cdot Cy_{i-1} \quad (8)$$

$$P_{i:i} = P_i = S_i' \oplus Cy_{i-1} \quad (9)$$

The design uses n + 1 cells. An external carry-input signal C_in is accommodated by using it as the input to the first saltire cell when computing $G_0 = S_0' \cdot$ C_in, so that the adder supports a carry in without an additional stage.

C. Stage 3: Propagate-Generate Logic

The carry computation stage combines grey and black cell logic to pre-compute the carry bits, using the group relations of (4) and (5). This is the stage that carries the logarithmic depth, and it is where the delay advantage over the ripple-carry stage of CS3A comes from.

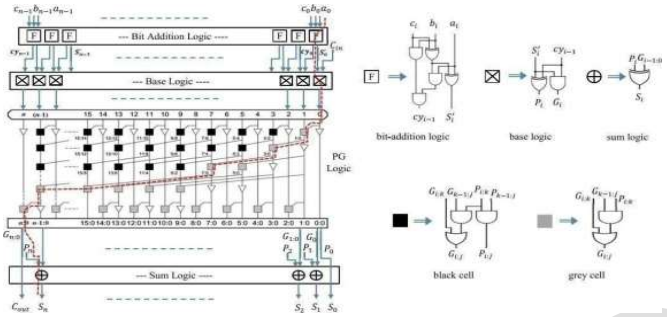

Fig. 5. Grey cells in the carry generation network.

D. Stage 4: Sum Logic

The final stage combines the propagate signals with the computed carries to produce the output sum bits.

V. Mac Unit Design

Multiply-and-accumulate is the operation at the centre of digital filtering, digital signal processing and deep neural network inference. A MAC unit multiplies two operands and adds the product to a running accumulator, and because this sequence repeats for every tap of a filter or every weight of a layer, the latency of the MAC directly sets the throughput of the datapath around it. Using a faster adder inside the MAC therefore has an effect out of proportion to the size of the adder itself.

A MAC unit with 8-bit inputs and a 16-bit output was implemented using the proposed 16-bit three-operand adder. The structure is shown in Fig. 6.

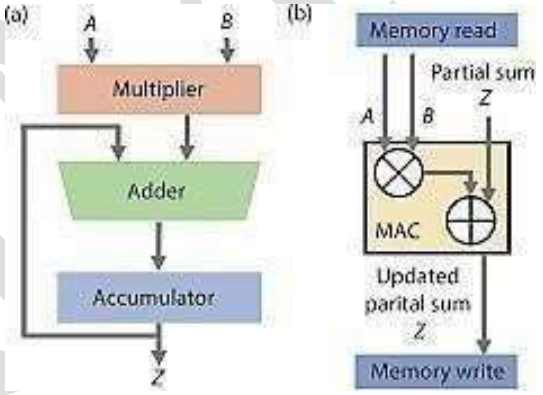

Fig. 6. MAC unit built with the proposed adder.

VI. Implementation

The design was described in Verilog and implemented using Xilinx tools. The workflow followed the standard sequence: creating a project and specifying the target device, entering the HDL source for the adder and MAC modules, declaring the module ports, synthesizing the design, writing a test bench to drive the inputs, and running behavioural simulation to inspect the output waveforms. The RTL schematic was examined after synthesis to confirm that the elaborated structure matched the intended architecture, and the synthesis report was used to read off area and timing.

VII. Simulation Results And Discussion

The simulation result for the three-operand binary adder is shown in Fig. 7, and for the MAC unit built with it in Fig. 8.

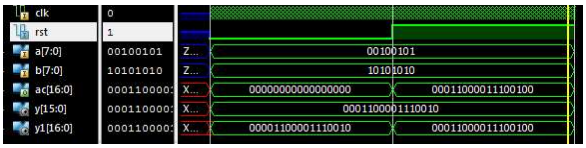

Fig. 7. Simulation result of the three-operand binary adder.

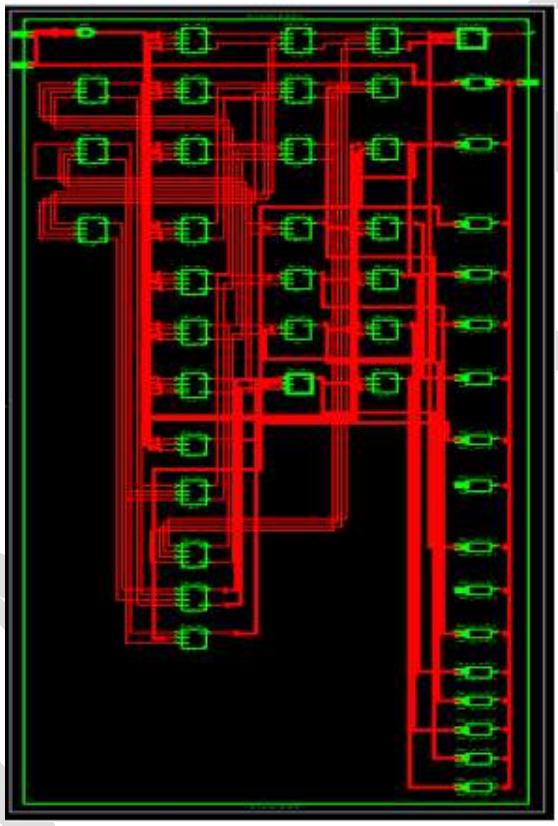

Fig. 8. Simulation result of the MAC unit with the proposed adder.

```
=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 304 / 17
-------------------------------------------------------------------------
Delay:               12.045ns (Levels of Logic = 10)
  Source:            a<1> (PAD)
  Destination:       sum<15> (PAD)

  Data Path: a<1> to sum<15>
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             3   1.222   0.898  a_1_IBUF (a_1_IBUF)
     LUT4:I0->O            2   0.203   0.721  Mxor_sum<2>_xo<0>11 (Mxor_sum<2>_xo<0>1)
     LUT5:I3->O            3   0.203   0.755  gc2/g1 (w<15>)
     LUT5:I3->O            3   0.203   0.755  gc3/g1 (w<27>)
     LUT5:I3->O            3   0.203   0.755  gc5/g21 (gc5/g2)
     LUT5:I3->O            3   0.203   0.755  gc6/g11 (gc6/g1)
     LUT5:I3->O            3   0.203   0.651  gc7/g21 (gc7/g2)
     LUT5:I4->O            3   0.205   0.755  gc7/g1 (gc8/g)
     LUT5:I3->O            1   0.203   0.579  Mxor_sum<15>_xo<0>1 (sum_15_OBUF)
     OBUF:I->O                 2.571          sum_15_OBUF (sum<15>)
    ----------------------------------------
    Total                     12.045ns (5.419ns logic, 6.626ns route)
                                       (45.0% logic, 55.0% route)

=========================================================================
```

Fig. 9. RTL view of the three-operand binary adder.

The simulations confirm functional correctness of both the adder and the MAC unit across the applied test vectors. The synthesis results show the proposed adder using less area, dissipating less power and having a smaller delay than HC3A, and achieving the lowest area-delay product and power-delay product among the three-operand techniques compared.

The reason for the improvement is structural rather than incidental. CS3A spends its second stage rippling a carry across n positions, so its delay is bounded below by that ripple regardless of how fast the individual full adders are. HC3A avoids the ripple but pays for it by instantiating two complete Han-Carlson networks. The proposed design collapses the three operands to two in a single parallel stage and then runs one prefix network rather than two, which is what gives both the logarithmic delay and the area saving at the same time. The gap over CS3A therefore widens with word length, and the reported speed advantage grows accordingly at 32, 64 and 128 bits.

VIII. Conclusion And Future Work

A three-operand binary adder using pre-computed bitwise addition followed by carry-prefix computation was implemented at 16 bits and used to build an 8-bit input, 16-bit output MAC unit. The four-stage structure reduces adder delay to $O(\log_2 n)$, avoiding the linear ripple delay of the carry-save approach without duplicating the prefix network as the Han-Carlson-based approach does. Simulation confirmed correct operation and synthesis confirmed the area, power and delay advantages.

The work also surveyed five parallel prefix adder topologies — Sklansky, Brent-Kung, Kogge-Stone, Han-Carlson and Ladner-Fischer — and their relative merits, which is the background against which the proposed structure was designed. The MAC unit built on the adder is applicable to digital signal processing, digital filtering and neural network hardware.

Further work could develop a hybrid prefix structure, using the associative property of prefix addition to keep the number of computation nodes to a minimum by eliminating the overlap between prefix sub-terms being computed. Implementing and characterizing the adder at 32, 64 and 128 bits would also confirm directly that the advantage over CS3A scales as the analysis predicts.

References

[1] N. H. E. Weste and D. M. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed. Addison-Wesley, 2010.

[2] G. Dimitrakopoulos and D. Nikolos, “High-speed parallel-prefix VLSI Ling adders,” IEEE Transactions on Computers, vol. 54, no. 2, pp. 225–231, 2005.

[3] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” IEEE Transactions on Computers, vol. C-31, no. 3, pp. 260–264, 1982.

[4] P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Transactions on Computers, vol. C-22, no. 8, pp. 786–793, 1973.

[5] J. Sklansky, "Conditional-sum addition logic," IRE Transactions on Electronic Computers, vol. EC-9, no. 2, pp. 226–231, 1960.

[6] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders," in Proc. 8th IEEE Symposium on Computer Arithmetic, 1987, pp. 49–56.

[7] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," Journal of the ACM, vol. 27, no. 4, pp. 831–838, 1980.

[8] S. Knowles, "A family of adders," in Proc. 15th IEEE Symposium on Computer Arithmetic, 2001, pp. 277–281.

[9] B. Parhami, Computer Arithmetic: Algorithms and Hardware Design. New York: Oxford University Press, 2000.

[10] N. Zamhari, P. Voon, and K. Kipli, "Comparison of parallel prefix adder," in Proc. World Congress on Engineering (WCE), vol. 2, 2012.

[11] A. Beaumont-Smith and C.-C. Lim, "Parallel prefix adder design," in Proc. 15th IEEE Symposium on Computer Arithmetic, 2001, pp. 218–225.

[12] A. K. Panda and K. C. Ray, "Modified dual-CLCG method and its VLSI architecture for pseudorandom bit generation," IEEE Transactions on Circuits and Systems I, vol. 66, no. 3, pp. 989–1002, 2019.

[13] A. K. Panda and K. C. Ray, "A coupled variable input LCG method and its VLSI architecture for pseudorandom bit generation," IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 4, pp. 1011–1019, 2020.

[14] R. S. Katti and S. K. Srinivasan, "Efficient hardware implementation of a new pseudo-random bit sequence generator," in Proc. IEEE Int. Symposium on Circuits and Systems, 2009, pp. 1393–1396.

[15] P. L. Montgomery, "Modular multiplication without trial division," Mathematics of Computation, vol. 44, no. 170, pp. 519–521, 1985.

[16] S.-R. Kuang, K.-Y. Wu, and R.-Y. Lu, "Low-cost high-performance VLSI architecture for Montgomery modular multiplication," IEEE Transactions on VLSI Systems, vol. 24, no. 2, pp. 434–443, 2016.

[17] M. M. Islam, M. S. Hossain, M. K. Hasan, M. Shahjalal, and Y. M. Jang, "FPGA implementation of high-speed area-efficient processor for elliptic curve point multiplication over prime field," IEEE Access, vol. 7, pp. 178811–178826, 2019.

[18] Z. Liu, D. Liu, and X. Zou, "An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor," IEEE Transactions on Industrial Electronics, vol. 64, no. 3, pp. 2353–2362, 2017.