Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

High Speed Area Efficient VLSI Architecture of Three Operand Binary Adder

P. Lohith Chowdary, Doddapaneni Satish, Balagala Chethan

Research Scholar, Department of Electronics and Communication Engineering, Amrita school of engineering, India.
Email: Lohitheers@gmail.com

The three-binary adder in operation has several uses and is the fundamental building block for computation using modules in many cryptographic and pseudorandom bit generator (PRBG) methods. Most often, the preserve the beetle (CS3A) is used to do the three-operand addition. There will be a major important route delay the reason behind the adder that carries ripples, found in last rescue the beetle. For addition with three operands, a functional prefix It is possible to utilize a two-operand adder such as Han-Carlson (HCA). Which drastically decreases calculating the time required for the critical route as the area complexity increases. An innovative adder design is developed that utilizes calculate bitwise addition in advance and carry prefix calculation logic to accomplish adding three binary operands with much reduced time and space consumption. The suggested architecture saves time and space compared to current designs such as the three-manifold With Han-Carlson adder and the carry save adder, which both use two operands. Utilizing Equipped with the The The Xilinx ISE 14.7 environment Tool, both the simulation and synthesis have been validated.
Keywords: modular arithmetic, three-operand adder, carry-save adder (CSA), and Han-Carlson adder (HCA).

1. Introduction

Hardware implementation of cryptographic techniques is necessary for physical security preservation and system performance optimization [1]-[3]. Many mathematical operations used in cryptography make use of modular arithmetic, including multiplication, addition, and exponentiation [4]. Success or failure of the encryption method is therefore dependent on how well congruential modular mathematics works process. As the cornerstone of the critical operation, the Montgomery algorithm [5]-[7] is the best way to do modular multiplication and

exponentiation. One of the primary mathematical operations of a PRBG based on a linear congruential generator, which linked such are LCG (CLCG) [9], MCLCG [10], along with connected variable input LCG (CVLCG) [11] is binary addition. At the moment, the safest and most random PRBG technique is the modified dual-CLCG (MDCLCG) approach, which uses LCG-based methods. The security and unexpectedness are both achieved in polynomial time. if the bit size is more over 32 bits. Consequently, Safeguarding the MDCLCG is enhanced as the operand size increases. But as mentioned in previous works, both crucial and the region route delay increase linearly due to the hardware design's consisting of four multiplexers, two comparators, and four three-operand modulo-2n adders areas. Thus, the MDCLCG's performance may be improved with the help of the adder with three operands.

2. LITERATURE REVIEW

The term "parallel prefix adders" among the greatest common types of mathematical units. After careful analysis at the architectural, RTL, gate, circuit, and layout levels, several mathematical formulations, topologies, and implementations emerged. In two ways, this study significantly expands our understanding of these parallel prefix adders. Initially, it seeks to provide sophisticated and rational terminology for characterizing various parallel prefix adders. Our second proposal is an architecturally-level latest generation of parallel prefix adders. Additionally, we provide estimations area-throughput relationship properties for a specific instance of this family. Their area characteristics are better than existing ones, and even if their speeds are equivalent to the most modern adders, the hardware complexity is still greater.

Summary of literature: Numerous parallel prefix adders, including ling adders with various topologies and Weinberger-Smith Recurrence, which has more space and less time, have been shown in this work.

3. EXISTING METHOD

1. Carry Save Adder:

A contemporary approach that can execute three-operand binary addition—an essential operation in the framework of congruential modular mathematics systems—is an LCG-based PRBG method, among many others. This kind of method includes *CLCG*, *MDCLCG*, and *CVLCG* are all recognized in the literature. A one-stage or two-stage two-operand adder may be used in the implementation, depending on the number of operands. From [9] to [14], the CSA (carry-save adder) was the go-to for three-operand binary addition. It just takes two steps to obtain the total of the three operands. It is recommended to begin by creating arrays of full adders. You can calculate the "carry" and "sum" bits with any full adder. At the same time, all three binary inputs (bi, and, and ci) are taken. The second step, the added-carry ripple, discovers the n-bit long final "sum" signal and the individual bit long "carry-out" signal at the output of the three-operand addition. A total of n full adders process the "carry-out" signal during the ripple-carry stage. Because of this, the latency increases linearly with the bit length. An example of a three-operand carry-save adder is shown in the figure below, with the

needed route delay represented by the dotted line. The following computation shows that the delay in the critical route is governed by carried out via the ripple carry stage.

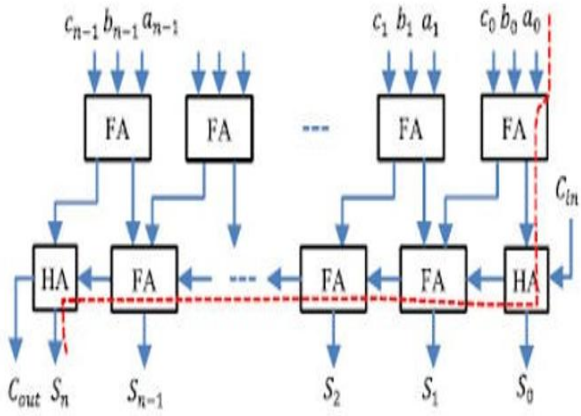


Figure 1: A carry-save adder with three operands critical route delay (CS3A).

Drawback:

One major issue with This is the CS3A. longer critical path time that comes from its greater This is the CS3A.

2. Han-Carlson Adder:

In order to drastically cut down on essential route time, the MR2MMM employs Hc2A, a parallel prefix Han-Carlson two-operand adder, operates at high speeds. Radix-2 Montgomery's revised modular technique employs two two-operand Han-Carlson adders that the three-operand addition should be performed. The following image depicts the three-operand adder based on Han-Carlson, which takes its cues from the HC2A's first-order architecture and characteristics diagram. Combining three operands is possible with two-stage Hanna-Carlson beetles like HC2A-1 and HC2A-2. A unique implementation of the Han-Carlson adder (HC2A) with two operands. Proliferate and produce (PG), base, and sum make up its trinity of levels [16]. Here we may see a base logic cell and a sum logic cell shown side by side.

Drawback:

The critical route delay is substantially increased by an adder with three operands that uses carry-save, as opposed to the three-operand binary adder [15] that is based on HCA. Unfortunately, the area increases in proportion to the adder's bit length.Consequently, a new technique for quickly, efficiently use space three-operand adders is created using an efficient VLSI design to lessen the area-latency trade-off.

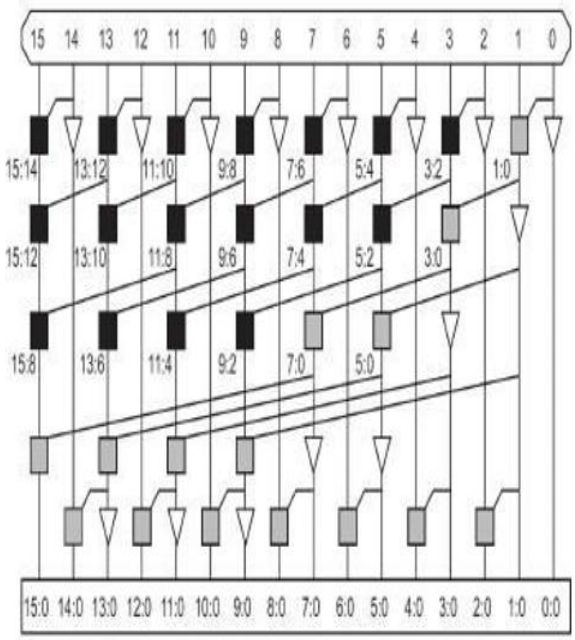


Figure2: Han-Carlson adders' carry generation structure

4. PROPOSED METHOD

So as to include addition with three operands into modular arithmetic, we developed a novel adder mechanism and an architecture for its very large-scale integration. An adder method referred to as an adder with a parallel prefix is suggested in this article. To calculate the addition for three operands that take binary inputs, several prefix adders use alternate four-stage frameworks of three-stage ones. These include PG, sum, bit-addition, and base protocols are all forms of logic. Each of these four stages may be thought of in this way:

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: Stage

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

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

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

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

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

Stage-4: Sum Logic:

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

Our modular arithmetic three-operand addition was accomplished by the introduction of a new adder mechanism and the design of its very large scale integration (VLSI). An adder technique that is being proposed here is the parallel prefix adder. Although most prefix adders employ a three-phase architectures to determine the sum of three operands that are supplied as binary numbers, there are a few that use there are four steps: "base logic," "bit-addition logic," Reasoning based on PG (propagate and generate) and sum. The following is a suitable method for representing each of these four procedures:

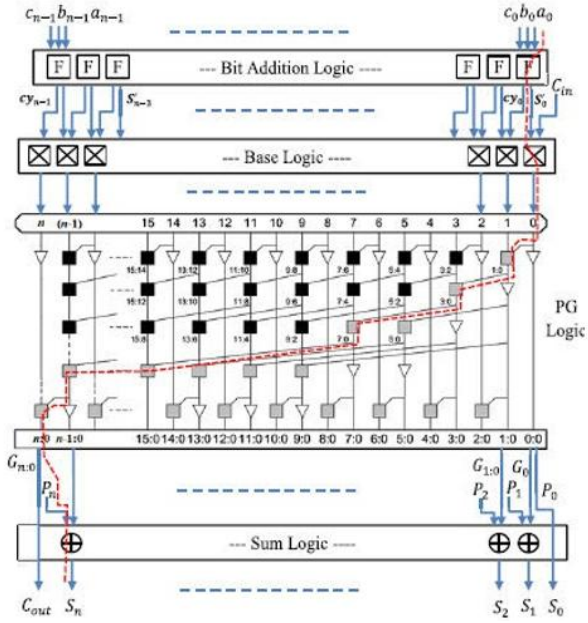


Fig.3: adder with three operands proposal

To establish the create as well as transmit (Pi) signals in the subsequent phase, which constitutes the fundamental logic, the starting point combines the output signals "sum (S)" and "carry" from the present full adder and the full adder to its right. The "squared saltire-cell" is used to represent computed signals of Gi and Pi in the base logic stage. Figure 3(a) shows that this level is composed of n+1 saltire-cells. As shown in Figure 3(b), the following logical equation allows for the realization of the saltire-cell logic diagram.

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

Following that, third step, mentioned by "generate and propagate logic" (PG), is the carry

calculation stage. The black in addition to the gray matter logics are combined to precalculate little for carrying. Picture 3(b) shows the flowchart for the black and grey cells. It uses the following logical formula to calculate the carry create Gi: "and spread" Pi: j signals:

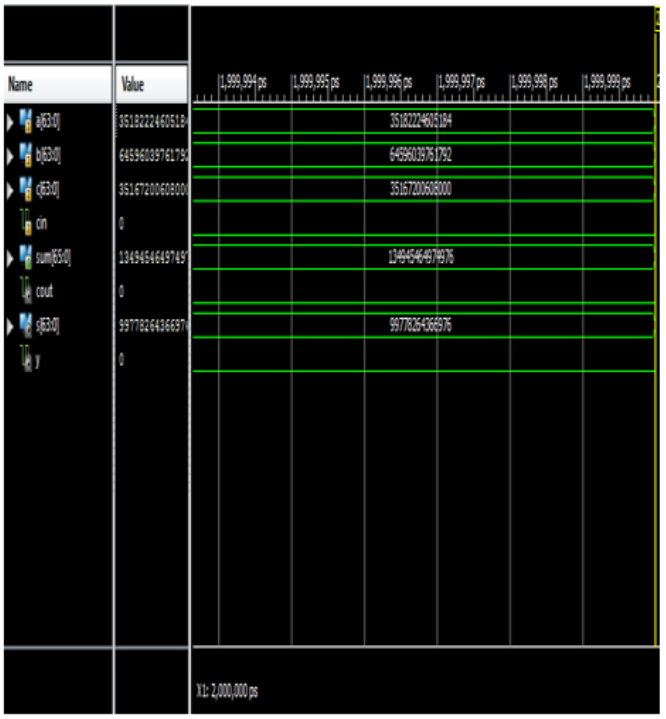
$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j},$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

Due to its (log2 n+1) steps of calculating prefixes Most of the impact on the key path comes from this carry propagate chain. latency of the suggested adder. Sum logic is used to depict the last phase, where a rational equation This is the equation for Si: $(Pi \oplus Gi - 1: 0)$ is used to compute "sum (Si)" parts of the vehicle produce You may propagate Pi bits using Gi: j and carry. Carry create a random number Gn is the immediate source of the carryout signal ($Cout$):

5. METHODS OR TECHNIQUES USED

The suggested adder is developed using A14.5 of the Xilinx ISE Design Suite by stimulating Verilog HDL. The Verilog HDL language is used to describe hardware. This term refers to network systems and other digital systems. It is quite simple to create and debug.

6. SIMULATIONRESULTS



Existing method

HC3A Adder

Area report:

```
Device utilization summary:
---------------------------

Selected Device : 7vx330tffg1157-2


Slice Logic Utilization:
 Number of Slice LUTs:                  444  out
    Number used as Logic:               444  out

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    444
   Number with an unused Flip Flop:     444  out
   Number with an unused LUT:             0  out
   Number of fully used LUT-FF pairs:     0  out
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                         259
 Number of bonded IOBs:                 259  out
```

Delay Report:

```
Data Path: a<1> to sum<65>
                           Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             3   0.000   0.534  a_1_IBUF (a_1_IBUF)
   LUT4:I0->O            4   0.043   0.422  g1/gc1/G1 (g1/c<1>)
   LUT5:I3->O            7   0.043   0.529  g1/gc2/G1 (g1/c<3>)
   LUT5:I2->O            7   0.043   0.529  g1/gc4/G11 (g1/gc4/G1)
   LUT5:I2->O            5   0.043   0.518  g1/gc4/G1 (g1/c<7>)
   LUT6:I3->O            4   0.043   0.367  g1/gc8/G1 (g1/gc8/G)
   LUT5:I4->O            3   0.043   0.417  g1/gc8/G2 (g1/c<15>)
   LUT6:I4->O            2   0.043   0.410  g1/gc16/G11 (g1/gc16/G11)
   LUT4:I2->O            5   0.043   0.518  g1/gc16/G12 (g1/gc16/G1)
   LUT6:I3->O            4   0.043   0.367  g1/gc28/G111 (g1/gc28/G111)
   LUT5:I4->O            5   0.043   0.626  g1/gc28/G112 (g1/gc28/G11)
   LUT6:I1->O            4   0.043   0.422  g1/gc28/G (g1/c<55>)
   LUT5:I3->O            4   0.043   0.630  g1/Mxor_sum<63:1>_56_xo<0>1 (s<57>)
   LUT6:I0->O            3   0.043   0.625  g2/bc28/P1 (g2/p1<27>)
   LUT6:I0->O            1   0.043   0.613  g2/gc32/G6_SW0 (N96)
   LUT6:I0->O            1   0.043   0.405  g2/gc32/G6 (g2/gc32/G5)
   LUT5:I3->O            2   0.043   0.618  g2/gc32/G7 (cout)
   LUT6:I0->O            1   0.043   0.339  g3/carry1 (sum_65_OBUF)
   OBUF:I->O                 0.000          sum_65_OBUF (sum<65>)
  ----------------------------------------
  Total                      9.622ns (0.731ns logic, 8.891ns route)
                                     (7.6% logic, 92.4% route)
```

Proposed three Operand Adder:

Area Report:

```
Slice Logic Utilization:
 Number of Slice LUTs:                  368  out of  204000     0%
    Number used as Logic:               368  out of  204000     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    368
   Number with an unused Flip Flop:     368  out of    368   100%
   Number with an unused LUT:             0  out of    368     0%
   Number of fully used LUT-FF pairs:     0  out of    368     0%
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                         259
 Number of bonded IOBs:                 259  out of    600    43%
```

Delay Report:

```
Data Path: b<0> to sum<65>
                           Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O             3   0.000   0.507  b_0_IBUF (b_0_IBUF)
   LUT3:I0->O            2   0.043   0.608  g1/carry1 (c1<0>)
   LUT5:I0->O            4   0.043   0.630  pg1/gc1/G1 (x<1>)
   LUT6:I0->O            2   0.043   0.618  pg1/gc33/G1 (x<2>)
   LUT6:I0->O            4   0.043   0.512  pg1/gc2/G1 (x<3>)
   LUT6:I3->O            3   0.043   0.534  pg1/gc4/G1 (x<7>)
   LUT6:I2->O            1   0.043   0.522  pg1/gc8/G1 (pg1/gc8/G)
   LUT4:I0->O            3   0.043   0.417  pg1/gc8/G2 (x<15>)
   LUT6:I4->O            1   0.043   0.405  pg1/gc16/G1_SW0 (N4)
   LUT6:I4->O            5   0.043   0.428  pg1/gc16/G1 (x<23>)
   LUT6:I4->O            1   0.043   0.405  pg1/gc28/G111 (pg1/gc28/G11)
   LUT5:I3->O            5   0.043   0.518  pg1/gc28/G112 (x<31>)
   LUT6:I3->O            2   0.043   0.527  pg1/gc32/G4 (pg1/gc32/G3)
   LUT6:I2->O            1   0.043   0.339  h1/carry1 (sum_65_OBUF)
   OBUF:I->O                 0.000          sum_65_OBUF (sum<65>)
  ----------------------------------------
  Total                      7.531ns (0.559ns logic, 6.972ns route)
                                     (7.4% logic, 92.6% route)
```

COMPARISION OF EXISTING AND PROPOSED ADDERS IN TERMS OF AREA & DELAY:

	Area (in LUT's)	Delay(in ns)
CS3A	192	36.786
HC3A	444	9.622
Proposed Adder	368	7.531

7. CONCLUSION

This is suggests in which different cryptographic algorithms should use a Advanced logic design and high-speed area-efficient adder technology to perform binarization using three operands. A parallel prefix adder with three input operands is the basis utilizing the three-operand adder that has been suggested method, which employs structure with four stages to calculate the sum. This suggested architecture stands out due to its decrease in latency and area during the rationale for bit-addition and PG prefix calculation phases, which in turn lowers the total critical path delay. Additionally, contrasted with other related prefix three operand adders, its area complexity is reduced. In contrast up to the The The CS3A and HC3A families families adders with three operands, suggested three-operand binary adder is both smaller and has reduced latency, as shown above. Utilizing With the Xilinx ISE tool, the simulation and synthesis are validated.

FUTURE SCOPE

This article proposes an adder mechanism called a parallel adder for prefixes. Some bits and prefix adders-addition formulas, PG (propagate and generate), sum formulas, base formulas, employ constructs with four levels as opposed to three ones to calculate the sum of three operands that are supplied as binary numbers. Substituting other carry propagation stages for the PG (propagate and generate) could increase the total value. Changing these designs can add delay and area.

References

[1] 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.

[2] Z. Liu, J. GroBschadl, Z. Hu, K. Jarvinen, H.Wang, and I. Verbauwhede, "Elliptic curve cryptography with efficiently computable endomorphisms and its hardware implementations for the Internet of Things," IEEE Trans. Comput., vol. 66, no. 5, pp. 773–785, May 2017.

[3] Z. Liu, D. Liu, and X. Zou, "An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor," IEEE Trans. Ind. Electron., vol. 64, no. 3, pp. 2353–2362, Mar. 2017.
[4] B. Parhami, Computer Arithmetic: Algorithms and Hardware Design. New York, NY, USA: Oxford Univ. Press, 2000.
[5] P. L. Montgomery, "Modular multiplication without trial division," Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985.
[6] S.-R. Kuang, K.-Y. Wu, and R.-Y. Lu, "Low-cost high-performance VLSI architecture for montgomery modular multiplication," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 24, no. 2, pp. 434–443, Feb. 2016.
[7] S.-R. Kuang, J.-P. Wang, K.-C. Chang, and H.-W. Hsu, "Energy-efficient high-throughput montgomery modular multipliers for RSA cryptosystems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 11, pp. 1999–2009, Nov. 2013.
[8] S. S. Erdem, T. Yanik, and A. Celebi, "A general digit-serial architecture for montgomery modular multiplication," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 5, pp. 1658–1668, May 2017.
[9] R. S. Katti and S. K. Srinivasan, "Efficient hardware implementation of a new pseudo-random bit sequence generator," in Proc. IEEE Int. Symp. Circuits Syst., Taipei, Taiwan, May 2009, pp. 1393–1396.
[10] A. K. Panda and K. C. Ray, "Modified dual-CLCG method and its VLSI architecture for pseudorandom bit generation," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 66, no. 3, pp. 989–1002, Mar. 2019.
[11] A. Kumar Panda and K. Chandra Ray, "A coupled variable input LCG method and its VLSI architecture for pseudorandom bit generation," IEEE Trans. Instrum. Meas., vol. 69, no. 4, pp. 1011–1019, Apr. 2020.
[12] N. Weste and K. Eshraghian, Principles of CMOS VLSI Design—A Systems Perspective. Reading, MA, USA: Addison-Wesley, 1985.
[13] T. Kim, W. Jao, and S. Tjiang, "Circuit optimization using carry-saveadder cells," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 17, no. 10, pp. 974–984, Oct. 1998.
[14] A. Rezai and P. Keshavarzi, "High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 9, pp. 1710–1719, Sep. 2015.
[15] A. K. Panda and K. C. Ray, "Design and FPGA prototype of 1024- bit Blum-Blum-Shub PRBG architecture," in Proc. IEEE Int. Conf. Inf. Commun. Signal Process. (ICICSP), Singapore, Sep. 2018, pp. 38–43.