International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Area and Delay Analysis of Modulo $2^n \pm 1$ Adder Subtractor Using Prefix Adder on Weighted One and Diminished-1

Kishore Kunal[1], Ghanshyam Jangid[2]

[1]Master of Technology (Scholar), Suresh Gyan Vihar University, Jaipur, India

[2]Assistant Professor, Suresh Gyan Vihar University, Jaipur, India

Abstract: ***Arithmetic architectures for modulo 2n+1 and 2n-1 adders and Subtractor are introduced in this paper. The first one is built around a sparse carry computation unit that computes only some of the carries of the modulo 2n + 1 and 2n-1 addition. Second is Apart from addition a 2s compliment methodology has been introduced fro subtraction concept. The results are focusing on area and timing delay. These results is also being comparing in diminished-1 and weighted one for the individually adder and Subtractor and, while maintaining a high operation speed.***

Keywords: Residue number system, Parallel Algorithm, Modular arithmetic, Weighted – one, diminished-1

1. Introduction

Arithmetic modulo operators has application within the fields starting from pseudorandom variety generation and cryptography up to convolution computations. Modulo 2n+1 one operators square measure unremarkably enclosed in residue mathematical notation applications. RNS is AN arithmetic system that decomposes variety into elements and performs arithmetic operations in parallel and it conjointly don't would like of carry computation for every residue.RNS is employed in addition/subtraction and multiplication operations and conjointly the planning of digital signal processors. The RNS supported the modulo set (2n -1, 2n, 2n +1) is most often used to attain a superior RNS application since the ensuing RNS design performs quick residue arithmetic. The selection of moduli set is extremely necessary and necessary for nearly equal delay of the channels. Special moduli sets are used extensively to cut back the hardware quality within the implementation of converters and arithmetic operations. Among that the triple moduli set has some advantages. Thanks to quantity lengths of those moduli the operation delay of this technique is set by the modulo 2n - one channel. this suggests that if we tend to weigh down the time needed for m1odulo 2n - one addition we tend to conjointly weigh down the RNS addition time. In order to hurry up the modulo 2n ± one arithmetic operations the diminished-1 illustration of binary numbers has been introduced.[1]

The complexness of a modulo 2n+1 arithmetic unit is set by the illustration of the input operands. 3 representations are thought of the traditional weighted one the diminished-1and the signed-LSB representations. We tend to solely contemplate the primary 2 illustrations here as a result of the signed-LSB representation isn't additional economical in delay or space terms. once we perform arithmetic operations modulo 2n + one the input operands and also the results ought to be in between zero and 2n .In the traditional weighted illustration every quantity needs n + one bits for its illustration however use 2n + one representations out of the 2n+1. diminished-1 illustration provides a dense cryptography of the input operands and simplified arithmetic operations modulo 2n+1. Within the diminished-1 illustration A is drawn as azA* wherever az may be a single bit, known as the zero indication bit, associate degreed A* is an n-bit vector known as the amount half. If A > 0, then az = zero and A* = A – one. Whereas for A =0; az = one, and A* = 0. as an example, the diminished-1 illustration of A = five modulo seventeen is 00100.[1]

Here we are designing modulo $2^n \pm 1$ adder- Subtractor with the help of parallel prefix adder. After designing the adder-Subtractor we also implement these designs on FPGA. For adder Subtractor we use weighted-one and diminished- one representation. Then we design both adder subtractors in both representation forms then compare the results in terms of area and time.

For this architecture first we design parallel prefix adder using verilog after design adder we design weighted-one adder-subtractor. For Subtractor we use the 2's complement method. Then Diminished –one adder –Subtractor is designed. Finally FPGA implementation is done for adder-subtractor.

2. Residue number system

A residue system of numeration (RNS) represents an oversized whole number employing a set of smaller integers, in order that computation could also be performed additional with efficiency. It depends on the Chinese remainder theorem of standard arithmetic for its operation.

A residue mathematical notation is outlined by a collection of N whole number constants, , spoken because the moduli. Let M be the smallest amount integer of all the mi. Any whimsical whole number X smaller than M may be drawn within the outlined residue mathematical notation as a collection of N smaller integers {x1, x2, x3, ... , xN} with

xi = X modulo mi

representing the residue class of X to that modulus.

Licensed Under Creative Commons Attribution CC BY

3. Parallel Algorithm

Steps for calculative prefix total victimization parallel Algorithm:

-Compute the sums of consecutive combines of things within which the primary item of the pair has a fair index: z0 = x0 + x1, z1 = x2 + x3, etc.

- Recursively reckon the prefix total w0, w1, w2, ... of the sequence z0, z1, z2, ...
- Express every term of the ultimate sequence y0, y1, y2 ... because the total of up to 2 terms of those intermediate sequences: y0 = x0, y1 = z0, y2 = z0 + x2, y3 = w0, etc.
- When the primary worth, every consecutive range Yi is either traced from a grip 0.5 as so much through the w sequence, or is that the previous worth more to 1 worth within the x sequence.

4. Modular Arithmetic

This is a system of arithmetic for integers, wherever numbers "wrap around" on reaching a particular worth: the modulus. Standard arithmetic outlined mathematically by introducing a congruity relation on the integers that's compatible with the operations of the ring of integers: subtraction, addition and multiplication. For a positive whole number n, 2 integers a and b ar same to be congruent modulo n

$$a \equiv b \pmod{n},$$

Their distinction a − b is Associate in Nursing whole number multiple of n (or n divides a − b). The amount n is termed the modulus of the harmony.

Foundation:
Binary numbers with n bits are denoted as A = an-1an-2……. a0 then

$$A = \sum_{i=0}^{n-1} (2^i a_i)$$

Reduction of a number A modulo a number M ("A mod M") can be represented by a division (with the remainder as result) or by subtracting the modulus until A <M. For the moduli $(2^n - 1)$ and $(2^n + 1)$, the modulo reduction of a number A with at most 2^n bits can be computed by an addition or subtraction.[3]

$2^n \bmod (2^n - 1) = 2^n - (2^n - 1) = 1$

the reduction modulo $(2^n - 1)$ can be represented as

$A \bmod (2^n - 1) = (A \bmod 2^n + A \operatorname{div} 2^n) \bmod (2^n - 1)$

where the modulo operation on the right hand side is used for final correction if the addition yields a result > $2^n - 1$ (2^n-1 has to be subtracted once). Then the modulo (2^n-1) reduction is computed by adding the high n bit word (A div 2^n) to the low n - bit word (A mod 2^n) and then subtracting $2^n - 1$.

Similarly,

$2^n \bmod (2^n + 1) = 2^n - (2^n + 1) = -1$

the reduction modulo (2n + 1) can be represented as

$A \bmod (2^n + 1) = (A \bmod 2^n - A \operatorname{div} 2^n) \bmod (2^n + 1)$

where the modulo operation on the right hand side is used for final correction if the subtraction yields a negative result ($2^n + 1$ has to be added once). Then the modulo (2^n+1) reduction is computed by subtracting the high n bit word from the low n bit word and then adding $2^n + 1$ Also the modulo operator has the property that a sum modulo M is equivalent to the sum of its operands modulo M:

(A + B) mod M=(A mod M+B mod M) mod M

4.1 Modulo (2n ± 1) Adder

Modulo-m addition of mod-m residues A and B (0 ≤ A, B < m) is defined as:

S=|A+B|m= {A + B –m if A + B > m }
A + B otherwise

Replacing m in this equation with $2^n - 1$ or $2^n + 1$ shows the corresponding equations for mod-(2^n–1) or mod-(2^n+1) addition, respectively. Because comparing A + B with 2^n-1 or $2^n + 1$ is nontrivial, so this equation is modified into new equations which use much simple comparisons with 2^n. Here, W = (wn wn–1 . . . w1 w0) = A + B is the true sum of A and B, which can be decomposed into a single bit wn and an n-bit number $|W|2^n$. Similarly, W ′ = (w′nw′n–1 . . . w′1w′0) = A + B – 1 is the diminished sum of A and B, with its associated decomposition into a single bit w′n and an n-bit number $|W'|2^n$ [4]

$S^- = |A+B|2^n - 1 = \{W - 2^n + 1$ if $W > 2^n\}$
W otherwise
$S^+ = |A+B|2^n + 1 = \{W' - 2^n$ if $W' > 2^n\}$
$W' + 1$ otherwise

4.2 Modulo (2n -1) addition:

Modulo (2^n -1) addition can be formulated as
$(A+B) \bmod (2^n-1) = \{A+B-(2^n+1) = (A+B+1) \bmod 2^n$ if $A+B > 2^n - 1\}$
A + B otherwise

This equation can also be written as:
$(A+B) \bmod (2^n-1) = \{A+B-(2^n - 1) = (A+B+1) \bmod 2^n$ if $A+B > 2^n\}$
A + B otherwise

4.3 Modulo 2n+1 Subtractor

Subtraction is an operation which is widely used in digital signal processing applications for calculating mean error estimation, mean square error estimation and calculation of sum of absolute differences. Modulo arithmetic is also used in these types of applications like efficient modulo subtraction circuits are mostly used.

5. Weighted – One Representation

This representation is also called integer representation. In this each operand requires n+ 1 bits for its representation but only utilizes $2^n + 1$ representations out of the 2^{n+1}[1]

Licensed Under Creative Commons Attribution CC BY

6. Diminished-1 Representation

In reduced 1 representation every operand is spoken to diminished by one contrasted with its weighted representation.[1] Zero operands are not utilized within the reckoning. The results are inferred then again when any operand or the result is zero. In this way just n-bit operands are utilized within a lessened 1 channel prompting more diminutive and speedier segments.

In the decreased 1 representation An is spoken to as aza*, where az is a solitary bit, frequently called the zero sign bit, and A* is a n-bit vector, regularly called the number part. On the off chance that A > 0, then az = 0 and A* = A – 1. while for A =0; az = 1, and A* = 0. Case in point, the lessened 1 representation of A = 5 modulo 17 is 0010

7. Parallel Prefix Operation

7.1 Addition

Suppose that A= An_1 An_2 . . .A0 and B= Bn_1 Bn_2 ... B0 represent the 2 numbers to be value-added and S=Sn_1 Sn_2 . S0 denotes their add. Associate in Nursing adder will be thought-about as a three-stage circuit. The preprocessing stage computes the carry-generate bits Gi, the carry-propagate bits Pi, and therefore the half-sum bits Hi for each i, zero ≤ i ≤ n − one, according to:

gi = ai . bi
pi = ai + bi
hi =ai ⊕ bi

Where . , +, and ⊕ denote logical AND, OR, and exclusive-OR, respectively. The second stage of the adder, called the carry computation unit, computes the carry signals Ci using the carry generate and carry propagate bits Gi and Pi. The third stage computes the sum bits according to

si = hi ⊕ ci-1

The computation of the carries *ci* can be performed in parallel for each bit position $0 \le i \le n-1$ using the formula[5]

$$ci = gi + \sum_{j=-1}^{i-1} \left(\prod_{k=j+1}^{i} pk \right) gi$$

where the bit g−1 represents the carry-in signal cin. Based on (3), each carry ci can be written as

ci = gi + Ki

It can be easily observed from that the switching activity of the bits ci equally depends on the values assumed by the carry-generate bits gi and the term Ki.

$$Ki = \sum_{j=-1}^{i-1} \left(\prod_{k=j+1}^{i} pk \right) gi$$

7.2 Background on Parallel-Prefix Addition

Many solutions are conferred for the carry-computation drawback. Carry computation is remodeled to a prefix drawback by victimization the associative operator ◦, that associates pairs of generate and propagate bits and it's outlined as

(g , p) ◦ (g' , p') = (g + p · g', p · p')

By using consecutive associations of the generate and propagate pairs (g, p),we can compute carry ci according to[5]

ci=(gi,pi)◦ (gi−1, pi−1) ◦ . . . ◦ (g1, p1) ◦ (g0, p0)

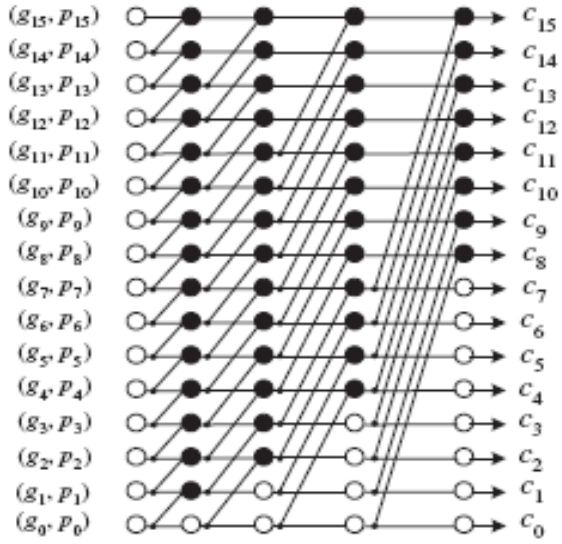


Figure 1: parallel-prefix carries computation in acyclic graph form

Here the operator ◦ as a node • and the signal pairs (g, p) as the edges of a graph and parallel-prefix carry computation units are in acyclic graphs form.

Parallel-prefix formulation of the computation of the bits Ki is represent here

K5 =p5 · g4 + p5 · p4 · g3 + p5 · p4 · p3 · g2 + p5 · p4 · p3 · p2 · g1+ p5 · p4 · p3 · p2 · p1 · g0 + p5 · p4 · p3 · p2 · p1 · p0 · cin

According to the definition pi · gi = gi. Then

K5=p5 · p4 · (g4 + g3) + p5 · p4 · p3 · p2 · (g2 + g1)+p5 · p4 · p3 · p2 · p1 · p0 · (g0 + cin)

Assuming that

Gi = gi + gi−1
Pi = pi · pi−1
G0 = g0 + cin
P0 = p0, then

K5=P5 · G4 + P5 · P3 · G2 + P5 · P3 · P1 · G0.

Computation of K5 can be transformed to a parallel-prefix problem by introducing the variable G∗i as follow-

G∗i = Pi · Gi−1
and G∗0 = p0 · cin.
K5 = G∗5 + P5 · G∗3 + P5 · P3 · G∗

which can be expressed in term of the ◦ operator as

K5 = (G∗5, P5) ◦ (G∗3 , P3) ◦ (G∗1 , P1).

In case of an 8-bit adder the bits Ki can be represent as

K7=(G∗7, P7)◦(G∗5,P5)◦ (G∗3, P3) ◦ (G∗1 , P1)
K6=(G∗6, P6)◦(G∗4,P4)◦ (G∗2, P2) ◦ (G∗0 , P0)
K5 = (G∗5, P5) ◦ (G∗3, P3) ◦ (G∗1, P1)
K4 = (G∗4, P4) ◦ (G∗2, P2) ◦ (G∗0, P0)
K3 = (G∗3, P3) ◦ (G∗1, P1)
K2 = (G∗2, P2) ◦ (G∗0, P0)
K1 = (G∗1, P1)
K0 = (G∗0, P0)

We can express the bits of Ki of odd and even indexed position as

K2k=(G∗2k,P2k)◦(G∗2k−2,P2k−2)◦ . . . ◦(G∗0, P0)
K2k+1 = (G∗2k+1, P2k+1) ◦ (G∗2k−1, P2k−1) ◦ . . . ◦ (G∗1, P1)

Licensed Under Creative Commons Attribution CC BY

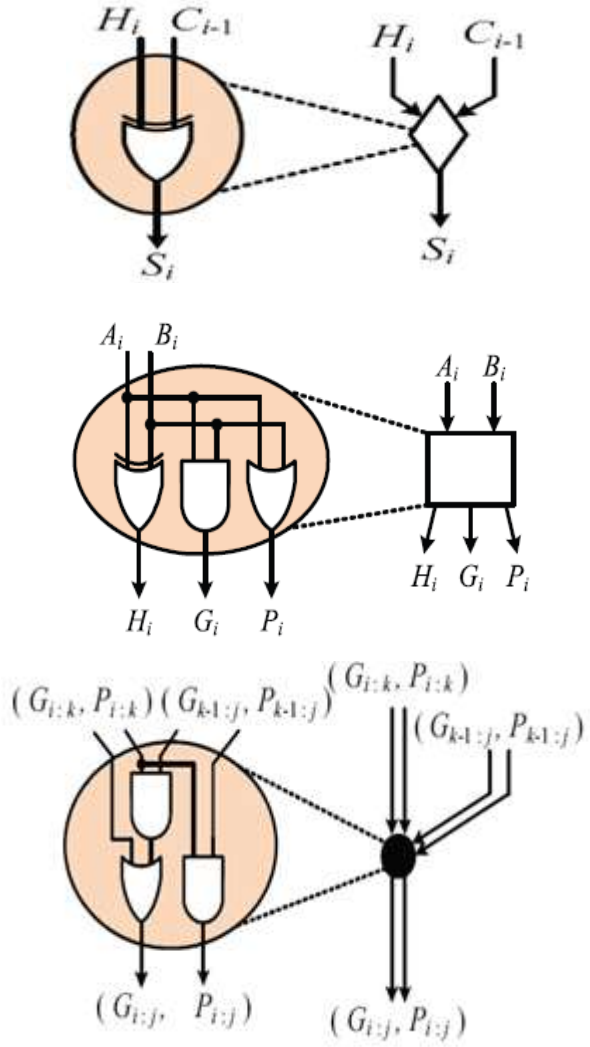


Figure 2: logic-level implementation of the basic cells

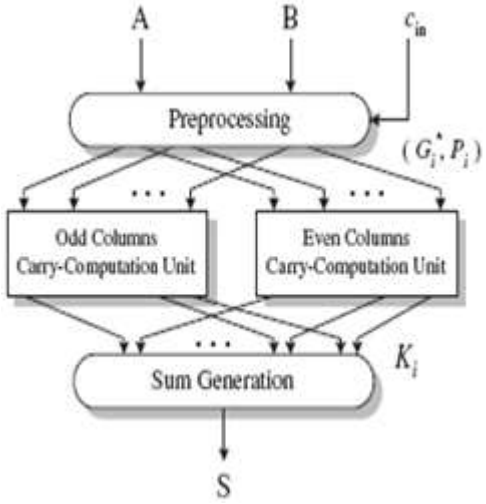


Figure 3: Architecture of the parallel-prefix adder

The computation of the bits Ki, instead of the normal carries ci, complicates the derivation of the final sum bits

si = hi ⊕ ci−1

si = hi ⊕ (gi−1 + Ki−1)

Steps used in the calculation of addition using parallel prefix algorithm

– Calculate the carry generate/propagate pairs (gi, pi)

– Combine the bits gi, pi, gi−1, and pi−1 in order to produce the intermediate pairs (G∗i,Pi)

Produce two separate prefix-trees one for the even and one for the odd indexed bit positions and compute the terms K2k and K2k+1 of equations.

8. Conclusion and Future Scope

Efficient modulo 2n+1 & 2n-1 adders and Subtractor area unit appreciated in a very style of pc applications together with all RNS implementations. during this paper, 2 contributions area unit offered to the modulo 2n+1 & 2n-1 addition and Subtractor downside that area unit weighted and diminished one. a completely unique design has been planned that uses a thin wholly regular parallel-prefix carry computation unit. This design was derived by proving the inverted circular idem potency property of the parallel-prefix carry operator in modulo 2n+1 & 2n-1 addition and by introducing a replacement prefix operator that eliminates the necessity for a double computation tree within the earlier quickest proposals whereas same parallel-prefix operation was designed for 2n+1 & 2n-1 subtraction with the extra 2s complement methodology.

9. Future Scope

Parallel-prefix structure is enticing for adders thanks to its exponent delay. The influence of style trade-offs may be simply discovered from adder styles. during this work, the radix is mounted as two. However, higher base could also be fascinating in some applications as indicated in thesis. With higher base, the prefix tree are a lot of distributed, permitting savings in space and power. With Doran's formula [39], the hybrid ripple-carry and prefix adders might have even higher performance. Another direction will result in the mixture of Ling's plan with carry-save notation. Considering all the chances, there are a unit still lots to hunt for adder style. Current adder style is obtaining a lot of advanced and adder's area unit having wider applications than ever before. Measurement exploitation bound technology is needed rather than straightforward gate model for performance analysis. Corner simulation is additionally fascinating for comprehensive characterization of adder architectures. Custom style techniques may be utilized for implementing adders. However, this sort of task is typically a lot of long. In further this modify design can be implement in TSMC 30nm for high precise data calculation of better result.

References

[1] Haridimos T. Vergos, Member, IEEE, and Giorgos Dimitrakopoulos, Member, IEEE,"On Modulo 2^n + 1 Adder Design" Feb.(2012)

[2] Wikipedia,http://en.wikipedia.org/wiki/Modular_arithmatic

[3] R.Zimmerman,"Efficient VLSI Implementation of Modulo $2^n \pm 1$ Addition and Multiplication," Proc. 14th IEEE Symp. ComputerArithmetic, pp. 158-167, Apr.(1999).

[4] G. Jaberipur and B. Parhami, "Unified Approach to the Design of Modulo-(2^n - 1)Adders Based on Signed-LSB Representation of Residues," Proc. 19th IEEE Symp. Computer Arithmetic, pp. 57-64,(2009).

Licensed Under Creative Commons Attribution CC BY

[5] G. Dimitrakopoulos1_, P. Kolovos1, P. Kalogerakis1, and D. Nikolos2, "Design of High-Speed Low-Power Parallel-PrefixVLSI Adders"
[6] H. T. Vergos and D. Bakalis, "On the Use of Diminished-1 Adders for Weighted Modulo $2n + 1$ Arithmetic Components"(2008)
[7] Tso-Bing Juang*, Member, IEEE, Pramod Kumar Meher**, Senior Member, IEEE,and Chin-Chieh Chiu*, "Efficient Weighted Modulo 2^n+1 Adders by Partitioned Parallel-PrefixComputation and Enhanced Circular Carry Generation"
[8] Tso-Bing Juang, Member, IEEE, Chin-Chieh Chiu, and Ming-Yu Tsai, "Improved Area-Efficient Weighted Modulo $2^n + 1$Adder Design With Simple Correction Schemes",Mar.(2010)
[9] Dina Younes and Pavel Steffan , "Novel Modulo $2^n +1$ Subtractor and Multiplier"(2011)
[10] G. Dimitrakopoulos, H.T. Vergos, D. Nikolos, and C. Efstathiou,"A Family of Parallel-Prefix Modulo $2^n - 1$ Adders," Proc. IEEEInt'l Conf. Application-Specific Systems, Architectures and Processors,pp. 326-336, (2003).

Author Profile



Kshore Kunal is a student of M.Tech (Dual Degree) Electronics & Communication + VLSI engineering at Suresh Gyan Vihar University, Jaipur. He is working on Verilog language.



Ghanshyam Jangid is Asst. Professor in Suresh Gyan Vihar University, Jaipur, Rajasthan. He has done his M. Tech from M.N.I.T, Jaipur, Rajasthan, India

Licensed Under Creative Commons Attribution CC BY