Analysis of Power Efficient Modulo 2^n+1 Adder Architectures

M.Parimaladevi
Assistant Professor
Velalar College of Engineering and Techonology,
Erode, Tamilnadu, India.

R.Karthi
P.G Scholar
Velalar College of Engineering and Techonology,
Erode, Tamilnadu, India.

ABSTRACT

Two modified architectures for modulo 2^n+1 adders are introduced in this paper. Only some of the carries of modulo 2^n+1 addition are computed in sparse carry computation unit present in first architecture. This sparse approach is introduced by inverted circular idempotency property of the parallel-prefix carry operator and in this modified pre-processing stage and carry select blocks are combine the multiplexer operation of a diminished-one adder can be implemented in smaller LUT's and less consumes power, while maintain the same operating speed and delay. The modulo adder 2^n+1 adders can be easily derived by adding extra logic of modulo 2^n-1 adders present in second architecture.

Keywords

Parallel-Prefix-Addition, IEAC, Modulo- Arithmetic, Boolean Expression, VLSI.

1. INTRODUCTION

Modulo 2^n+1 arithmetic has been used in cryptography [1], [2].Cryptography is the art of protecting information by transforming encryption into an unreadable format called chipper text. Decryption posses a secret key to change the message into plain text. Generally cryptography systems can be classified into symmetric-key system and public-key system. Symmetric-key system uses single key to key both sender and recipient and public-key systems are used two keys, a public key known to everyone and a private key that only the recipient of message uses.

The modulo 2^n+1 arithmetic unit complexity is determined by chosen for the operands representation. Three representations are considered namely, the normal weighted-one, diminished-one [3], and the signed-LSB representation [4]. In above only we consider the first two representations, since the adoption of the signed-LSB representation does not lead to more efficient circuits in delay or area terms. In every case, when performing modulo 2^n+1 arithmetic operation, the input operands and results are limited between 0 and 2^n. In the normal-weighted representation, each operand requires n+ 1 bit for its representation but only utilizes 2^n+1 representation out of the 2^{n+1} that these can provide. A denser encoding of the input operands and simplified arithmetic operations modulo 2^n+1 are offered by the diminished-1 representation. In the diminished-1 representation, each number Z IS represented by as $a_z Z^*$, where a_z is a single bit, often called the zero indication bit, and Z^* is an n-bit vector, often called the number part. If Z>0, then a_z=0 and Z^* = Z-1, Where for A=0, a_z = 1 and Z^* = 0.

1.1 Related Work

Many papers have attacked the problem of designing efficient diminished adders. The majority of them rely on the use of an inverted end around carry (IEAC) n-bit adder, which is an adder that accepts two n-bit operands provides a sum increased by one compared to their integer sum if their integer addition does not result in a carry output. Although an IEAC adder can be implemented by using an integer adder in which carry output is connected back to its carry input via an inverter, such a direct feedback is not a good solution. Since the carry input, a direct connection between them forms a combinational loop that may lead to an unwanted race condition [5].Considering diminished-1 representation foe modulo 2^n+1 arithmetic operation, [3], [4], used an IEAC adder which is based on an integer adder along with an extra carry look ahead (CLA) unit. The CLA unit computes the carry output which is then inverted used as the carry input of the integer adder. Zimmerman [6], proposed IEAC adders that makes use of a parallel-prefix computation unit along with an extra prefix level that handles the inverted end around carry. Although these architectures are faster than the carry look ahead ones proposed in [7], for sufficiently wide operands, they are slower than the corresponding parallel-prefix integer adders because of the need for the extra prefix level. In [7] it has been shown that the recirculation of the inverted end around carry can be performed within existing prefix levels, that is, in parallel with the carries computation. In this way, the need of the extra prefix level is canceled and parallel-prefix IEAC adders are derived that can operate as fast as their integer counter parts. Unfortunately, this level of performance require more area then the solutions of [6], since a double parallel-prefix computation tree is required in several levels of the carry computation unit. For reducing area complexity of the parallel-prefix by select-prefix [8] and circular carry select [9] IEAC adders can be proposed. Although a modulo 2^n+1 adder follows the (n+1)-bit weighted representation can be designed following principles of generic modulo adder design [10], [11]. However, it has been recently shown [12] that weighted adder can be designed efficiently by using an IEAC one and a carry save adder (CSA) stage. As a result, improving the design for an IEAC adder would improve weighted adder design as well.

2 Parallel-Prefix Addition Basics

Generally parallel-prefix n-bit adder considered as a three stage circuit. They are pre-processing-stage, carry-computation-unit and post-processing-stage. Suppose that $A = A_{n-1}A_{n-2} \ldots A_0$ and B = B_{n-1} B_{n-2} . . . B_0 represent the two numbers to be added and S = S_{n-1} S_{n-2} . . . S_0 denotes their sum.

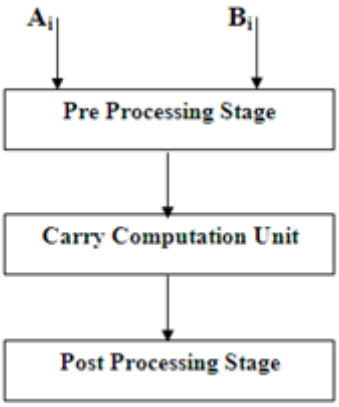

Fig. 1. Parallel-Prefix Addition Basics

2.1 Pre Processing Stage

The pre processing stage computes three type of signal bits. They are carry-generate bits G_i, the carry-propagate bits P_i, and the half-sum bits H_i, for every I, $0 \le i \le n-1$, according to

$$G_i = A_i \cdot B_i P_i = A_i + B_i H_i = A_i \oplus B_i$$

Where ·, +, ⊕ denote the logical AND, OR, and EXCLUSIVE-OR, respectively. The pre-processing-stage is shown in the figure 2.

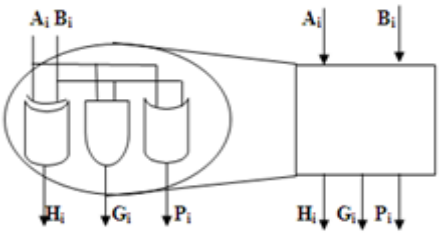

Fig. 2. Pre processing stage

2.2 Carry Computation Unit

The second stage of the adder, here after called the carry computation unit, computes the carry signals C_i, for $0 \le i \le n-1$ using the carry generate and carry propagate bits G_i and P_i. Carry computation transformed into a parallel prefix problem using the ◦ operator, which associate pairs of generate and propagate signals and defined as

$(G, P) \circ (G', P') = (G + P \,.\, G', P \cdot P')$

In a serious of associations of consecutive generate/propagate pairs (G, P) , the notation $(G_{k:j}, P_{k:j})$ with k>j, used to denote the group generate/propagate term produced out of bits k, k-1, . . .j, that is,

$(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \ldots \circ (G_j, P_j)$

Since every carry $C_i = G_{i:0}$, a number of algorithms have been introduced for computing all the carries using only ◦ operator. The prefix operator is shown in the figure 3.

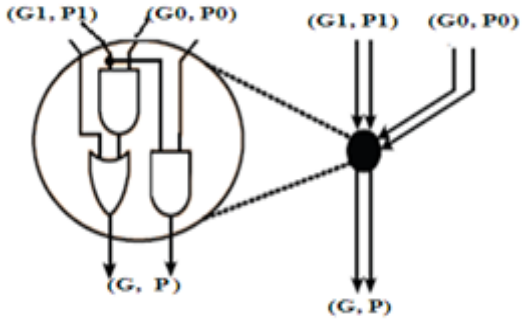

Fig. 3. Carry computation unit

2.3 Post Processing Stage

The third computes the half sum bits according to

$S_i = H_i \oplus C_{i-1}$

The post processing stage is shown in the figure 4.

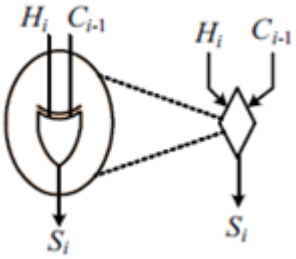

Fig. 4. Post Processing Stage

3 Modulo $2^n \pm 1$ Addition Basics

3.1 Modulo 2^n–1 Adders

The computation of modulo $2^n - 1$ addition is, in fact, a conditional operation defined as,

$$(A+B) mod\ (2^n - 1) = \begin{cases} (A+B), & A+B < 2^n \\ (A+B+1) mod\ 2^n, & A+B \ge 2^n \end{cases} \quad \ldots (1)$$

A modulo $2^n - 1$ adder can be implemented using an integer adder that increments also its sum when the carry output is one, that is, when $A + B \ge 2^n$. the conditional increment can be implemented by an additional carry incremental stage as shown in figure 5. In this case, one extra level of • cells driven by the carry output of the adder, is required.

Depending on the implementation of the modulo $2^n - 1$ adder, for bitwise-complementary inputs, i.e., when $A + B = 2^n + 1$, the adder may produce an all 1s output vector, in place of the expected result which is equal to zero. In most applications, this is acceptable as a second representation for zero.

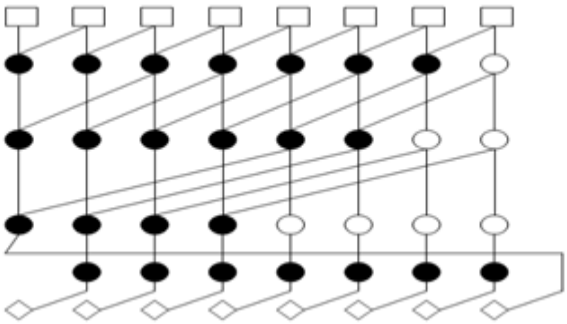

Fig. 5. Design of a prefix modulo $2^8 - 1$ adder

The implementation of a modulo $2^n - 1$ adder requires the connection of the carry output $C_{n-1} = G_{n-1:0}$ of a integer adder to its carry-input port. The carries of the modulo $2^n - 1$ adder C^-i that take place also a carry-input port are equal to $C_i^- = G_{i:0} + P_{i:0} \cdot C_{in}$. Therefore, connecting the carry output to the carry input leads to $C_i^- = G_{i:0} + P_{i:0} \cdot G_{n-1:0}$. This relation contains many redundant terms and according and simplified to

$$C_i^- = G_{i:0} + P_{i:0} \cdot G_{n-1:i+1} \ldots . (2)$$

The simpler equation can be equivalently expressed using the ◦ operator as follows:

$$C_i^- \leftrightarrow (G_i, P_i) \circ \ldots (G_0, P_0) \circ (G_{n-1}, P_{n-1}) \circ \ldots \circ (G_{i+1}, P_{i+1}) \ldots (3)$$

The above equation (3) that computes the modulo $2^n - 1$ carries has a cyclic form and, in contrast to integer addition, the number of generate and propagate pairs (G_i, P_i) that need to be associated for each carry is equal to n. This means that the parallel-prefix carry computation unit of a modulo 2^n -1 adder has significantly increased area complexity than that of a corresponding integer adder. In terms of delay, the carries C^- can be computed in $\log_2 n$ levels using regular parallel-prefix structures using end around technique. At each level of the parallel-prefix structure, larger groups of (G_i, P_i) are progressively associated and the carries C^- are computed at the last level. The final sum bits S^-_i are equal to $H_i \oplus C_{i-1}^-$. The above from of modulo $2^n - 1$ adder suffers from the double representation of zero. Few solutions have been reported on the design of a modulo $2^n - 1$ adder with a single zero representation. Those proposed by [13] have an increased delay compared to those with a double zero representation since they rely on using H_i instead of P_i as the carry propagate signal, while those proposed in [14] compute the modulo carries C^- as

$$C_i^- \leftrightarrow (G_i, P_i) \circ \ldots (G_0, P_0) \circ (G_{n-1}, P_{n-1}) \circ \ldots \circ (P_{i+1}, P_{i+1})$$

that is, by using P_{i+1} instead of G_{i+1}. Although this change seems minor, it ruins the regularity of the adders, and the interconnect area. In the rest of this paper, we consider modulo $2^n - 1$ adder with a double representation for zero.

3.2 Modulo 2^n+1 Adder

Diminished-1 modulo $2^n + 1$ addition is more complex since special care is when at least one of the input operand is zero (1 00…..0). The sum of a diminished-1 modulo adder is derived according to the following cases:

(1) When none of the input operand is zero (a_z, bz $\neq 0$) their number parts A^* and B^* are added modulo $2^n + 1$. This operation as discussed in the following can be handled by an IEAC adder.

(2) When one of the two input's are zero, the result is equal to the non zero operand.

(3) When both operands are zero, the result is zero.

In any case that the result is equal to zero (case 1 or 3), the zero-indication bit of the sum needs to be should be equal to the all-zero vector. According to above, a diminished-1 adder is needed only in case 1, while in the other cases the sum is known in advance.

When none of the input operands is zero, a_z, $b_z \neq 1$, the number part of the diminished-1 sum is derived by the number parts A^* and B^* of the input operand as follows:

$S^+ = (A^* + B^*) \bmod (2^n + 1)$

$$= \begin{cases} (A^* + B^*) mod 2^n, & A^* + B^* < 2^n \\ (A^* + B^*) mod 2^n, & A^* + B^* \geq 2^n \end{cases} \ldots (4)$$

In analogous way to that of the modulo $2^n - 1$ case, [8] has shown that the carry C_i^+ at the i^{th} bit position of an IEAC adder, when feeding the carry input $C_{in} = C_{-1}^+$ with the inverted carry out $\overline{C_{n-1}} = \overline{G_{n-1:0}}$, can be computed more simply by

$$C_i^+ = G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}} \ldots . (5)$$

Equivalently, using the ◦ operator the IEAC addition carries can be expressed as

$$C_i^+ \leftrightarrow (G_i, P_i) \circ \cdots (G_0, P_0) \circ \overline{(G_{n-1}, P_{n-1}) \circ \cdots \circ (G_{i+1}, P_{i+1})},$$

Where by definition, $\overline{(g,p)}$ is equal to $(\overline{g}, p)$, and the final sum bits S_i^+ are equal to $H_i \oplus C_{i-1}^+$. Using the above simplified carry equations.

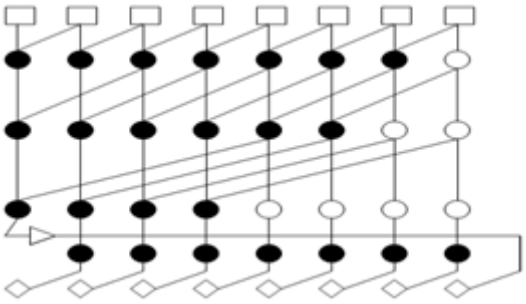

Fig. 6. Design of a prefix modulo $2^8 + 1$ adder

4 NEW SPARSE MODULO 2^n+1 ADDERS

In this section, we focus on the design of diminished modulo adders with a sparse parallel-prefix carry computation stage that can use the same carry-select block as the sparse integer adders.

4.1Partially Regular Sparse Parallel-Prefix Adders

The carries of the diminished-1 modulo $2^n + 1$ addition are associated in the very same way as the carries of the integer addition. To this end, the inverted circular idempotency property is introduced by the following Theorem:

Theorem 1.

$$(G_{i:0}, \quad P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})} \circ (G_{i:0}, \quad P_{i:0})$$

$$=(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})}$$

With $\overline{(G,P)} = (\overline{G}, P)$, according to the definition given in [8].

Proof: At first, we ungroup the $\circ$ operator with their equivalent Boolean relation as

$$(G_{i:0}, \quad P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})} \circ (G_{i:0}, \quad P_{i:0})$$

$$= (G_{i:0}, P_{i:0}) \circ (\overline{G_{n-1:i+1} + P_{n-1:i+1} \cdot G_{i:0}}, P_{n-1:i+1} \cdot P_{i:0})$$

$$= \left(G_{i:0} + P_{i:0} \cdot \overline{(G_{n-1:i+1} + P_{n-1:i+1} \cdot G_{i:0})}, P_{i:0} \cdot P_{n-1:i+1} \cdot P_{i:0}\right).$$

In the following, in the generate part of the prefix relation we expand the inversion operation, While in the propagate part we simplify the double appearance of the term $P_{i:0}$ as

$$(G_{i:0}, \quad P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})} \circ (G_{i:0}, \quad P_{i:0})$$

$$= \quad (G_{i:0} + P_{i:0} \cdot (\overline{G_{n-1:i+1}} \cdot (\overline{P_{n-1:i+1}} + \overline{G_{i:0}})), P_{i:0} \cdot P_{n-1:i+1}).$$

$$=\left(G_{i:0} + \left(P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{P_{n-1:i+1}}\right) + \left(P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{G_{i:0}}\right), P_{i:0} \cdot P_{n-1:i+1}\right).$$

The term $G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}} \cdot \overline{G_{i:0}}$ are reduced to $G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}$ which simplifies the initial relation as follows:

$$(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})} \circ (G_{i:0}, \quad P_{i:0})$$

$$= \left(G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}, P_{i:0} \cdot P_{n-1:i+1}\right).$$

If we reuse the $\circ$ operator, the equation is rewritten in the following way:

$$(G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})} \circ (G_{i:0}, \quad P_{i:0})$$

$$= (G_{i:0}, P_{i:0}) \circ (\overline{G_{n-1:i+1}} \cdot P_{n-1:i+1})$$

$$= (G_{i:0}, P_{i:0}) \circ \overline{(G_{n-1:i+1}, P_{n-1:i+1})},$$

This concludes the proof.

The inverted circular idempotency indicates that we can repeat (G_i, P_i) terms that appear at the front of a prefix relation of the form suggested by (5) inverted at its tail. Armed with the inverted circular idempotency, we will present the modified proposed methodology by using as an example the design of a sparse-4 parallel-prefix modulo $2^{16} + 1$ adder. Since we assume a sparsity-4, only one every four carries is generated at positions 3, 7, 11, 15, and 31 or equivalently -1.

4.2 Totally Regular Parallel-Prefix Units

The methodology presented in [7] is the only approach known so far that can organize the computation of the carries C^+, in case of modulo $2^n + 1$ addition, in a parallel-prefix-like form with $\log_2 n$ prefix levels. As also shown in the figure some prefix operator are double up, since two carry computations need to be performed in parallel; one on normal propagate and generate signals, while the other on their complements. The problem gets worse when the input operands' width is not a power of two. Although, the sparse version of the parallel-prefix adders introduced in this paper alleviates a lot the regularity and the area-overhead problem, as it can be verified from figure 6, there is still a lot space for improvement.

In the following, this problem is solved by introducing a new prefix operator and an even simpler parallel-prefix carry computation unit. The new technique will be presented via an example. Let the design of a sparse-4 diminished-1 modulo $2^{16} + 1$ adder be considered. In this case, a carry computation unit is needed that implements the following prefix equation

$$C_{15}^+ \leftrightarrow \left(\overline{G_{15:0}, P_{15:0}}\right), C_{11}^+$$

$$\leftrightarrow (G_{11:0}, P_{11:0}) \circ \overline{(G_{15:12}, P_{15:12})},$$

$$C_7^+ \leftrightarrow (G_{7:0}, P_{7:0}) \circ \overline{(G_{15:8}, P_{15:8})},$$

$$C_3^+ \leftrightarrow (G_{3:0}, P_{3:0}) \circ \overline{(G_{15:4}, P_{15:4})},$$

To this end, a new operator called gray operator is introduced. The implementation of a gray operator is given in figure. It accepts five inputs and produces four outputs. Three of the inputs of a gray operator residing at prefix level j – 1, namely G_V^{j-1}, P_V^{j-1} and T_V^{j-1} form the operator's vertical input bus, while the rest two G_L^{j-1} and P_L^{j-1} form its lateral input bus. The lateral bus signals are driven inverted to the operator. The gray operator produces three signals for vertical successor of prefix level j (G_V^j, P_V^j and T_V^j) and one (c^j) for its lateral successor. Note that compared to the $\circ$ prefix operator, the gray one requires one extra gate, but does not require logic levels. Considering a sparse-2^k parallel-prefix carry computation unit, gray operators will not be used in the first k prefix levels, since these need only compute the group generate and propagate terms out of 2^k adjacent bit positions.

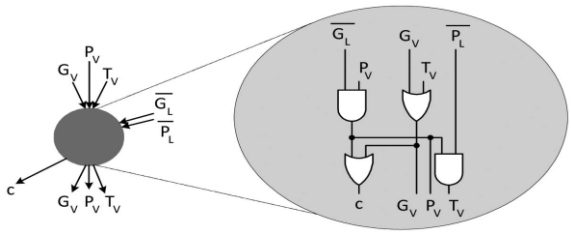

Fig. 7. Design of a gray operator

The logic equations performed by a gray operator residing at prefix level j – 1 are

$$G_V^j = G_V^{j-1} + T_V^{j-1},$$

$$P_V^j = P_V^{j-1} \cdot \overline{G_L^{j-1}},$$

$$T_V^j = P_V^j \cdot \overline{P_L^{j-1}},$$

$$c^j = G_V^j + P_V^j,$$

Consider now that we connect

T_V^{j-1} to 0,

G_V^{j-1} and P_V^{j-1} to $G_{k:0}$ and $P_{k:0}$, respectively, and G_L^{j-1} and P_L^{j-1} to $G_{n-1:r}$ and $P_{n-1:r}$, respectively.

Then, the gray operator will provide $c^j = G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:r}}$ as its lateral output. More importantly, it will also provide at its vertical outputs:

$$G_L^j = G_{k:0},$$

$$P_L^j = P_{k:0} \cdot \overline{G_{n-1:r}},$$

$$T_L^j = P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}},$$

An information which as we will show in the following suffices for the vertical successor to compute $(G_{k:0}, P_{k:0}) \circ \overline{(G_{n-1:m}, P_{n-1:m})}$ out of $(G_{k:0}, P_{k:0}) \circ \overline{(G_{n-1:r}, P_{n-1:r})}$ and $\overline{(G_{r-1:m}, P_{r-1:m})}$. Consider the vertical successor of the aforementioned gray operator which resides in prefix level j. by using a gray operator in its place in which we connect G_L^j and P_L^j to $G_{r-1:rm}$ and $P_{r-1:m}$, respectively, and G_V^j, P_V^j and T_V^j to the vertical output of the gray operator of level j-1 mentioned above, that is,

$$G_V^j \; to \; G_{k:0},$$

$P_V^j \; to \; P_{k:0} \cdot \overline{G_{n-1:r}}$, and

$$T_V^j \; to \; P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}}.$$

The lateral output of the operator will be equal to

$$c^j = G_V^j + P_V^j = G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{P_{n-1:r}} + P_{k:0} \cdot \overline{G_{n-1:r}} \cdot \overline{G_{r-1:m}},$$

$= G_{k:0} + P_{k:0} \cdot \left(\overline{G_{n-1:r}} \cdot \left(\overline{P_{n-1:r}} + \overline{G_{r-1:m}}\right)\right),$

$= G_{k:0} + P_{k:0} \cdot (\overline{G_{n-1:r}} \cdot (\overline{P_{n-1:r} \cdot G_{r-1:m}})),$

$= G_{k:0} + P_{k:0} \cdot \overline{(G_{n-1:r} + P_{n-1:r} + G_{r-1:m})},$

$= G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:m}},$

And will provide at its vertical outputs:

$$G_V^{j+1} = G_{k:0},$$

$$P_V^{j+1} = P_{k:0} \cdot \overline{G_{n-1:m}},$$

$$T_V^{j+1} = P_{k:0} \cdot \overline{G_{n-1:m}} \cdot \overline{P_{n-1:m}},$$

Applying the same procedure recursively, the lateral output of the last vertical successor of a gray operator will be equal to

$$G_{k:0} + P_{k:0} \cdot \overline{G_{n-1:k+1}},$$

That is, equal to C_k^+.

From the above analysis, it is concluded that starting from a sparse architecture with doubled up operators, it suffices to

1 Remove the doubled up operators that associate inverted signals,

2 Replace the top operator of every column excluding the leftmost that accepts a feedback signal with a gray one, with its T_V input tied to zero, and

Replace every vertical successor of a gray operator introduced by the previous step with a gray one,

To attain a diminished-1 modulo $2^n + 1$ adder, in which two gray operator are used. The top one which resides at prefix level 3, accepts a feedback signal and therefore has its T_V^3 input tied to zero. This operator is used to compute $(G_{3:0}, P_{3:0}) \circ \left(\overline{G_{15:12}, P_{15:12}}\right)$, which is necessary for the computation of both $C_3^+ \; and \; C_{11}^+$. Its vertical successor is also replaced by a gray operator that computes the final:

$$C_3^+ \leftrightarrow (G_{3:0}, P_{3:0}) \circ \overline{(G_{15:12}, P_{15:12}) \circ (G_{11:4}, P_{11:4})}$$

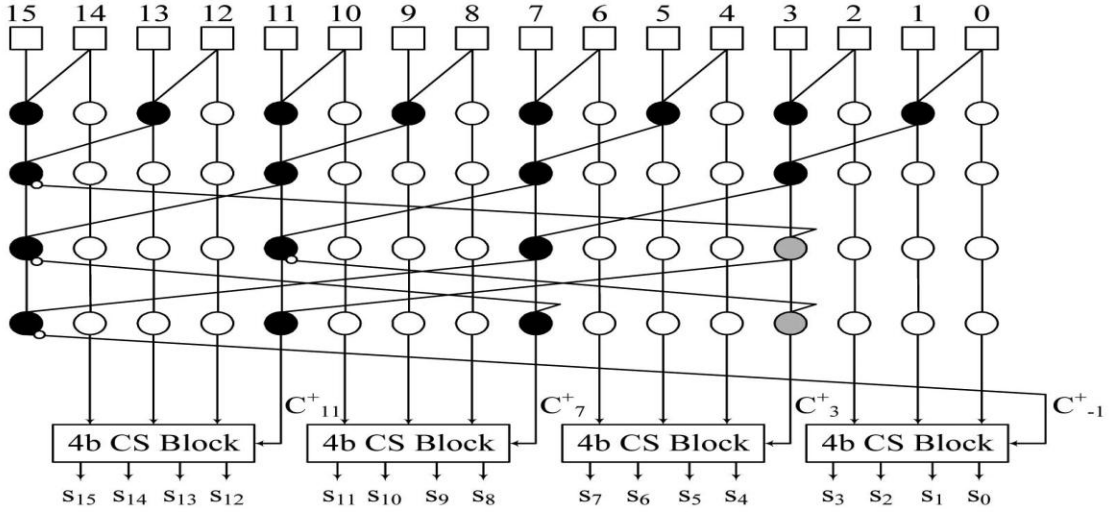

Fig. 8. Design of a Diminished-1 modulo $2^{16} – 1$ Adder Using CSB

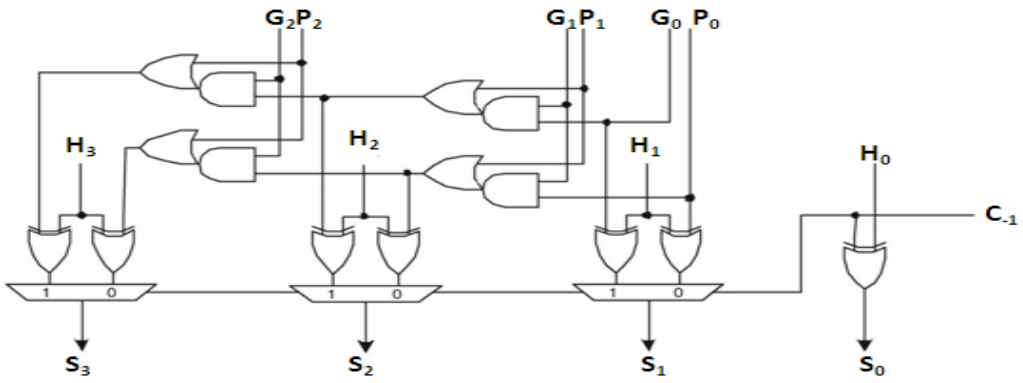

Fig. 9.Design of a 4-bit Carry Select Block

5 Unified Approaches to the Design of Modulo $2^n \pm 1$ Adder

Several area-time-power efficient architectures (for example, [16]) have been proposed for the simpler case of modulo $2^n - 1$ addition. These architecture preserve all the benefits of parallel-prefix carry computation units and can be easily designed for every n. more specifically, Dimitrakopoulos et al. [16] generalized the design of such units for all values of n and has provided easy-to-follow topographical design rules. The resulting structures for $n \neq 2^k$ save significant amount of area without scarifying delay. Therefore it is concluded that mapping the diminished modulo $2^n + 1$ adder design problem to that of modulo $2^n - 1$ addition, would beneficial given all the efficient architectures that have been proposed for the latter. In the following, it is shown that this mapping requires a constant time post processing stage and analyzes its area and time overhead.

5.1 Modulo $2^n - 1$ Unification Theory

In order to unify the parallel-prefix modulo $2^n - 1$ addition principles, there is a need to explore the relation between carries of these two addition operators, that is, between C_i^- and C_i^+.

The relationship that connects the recirculation carry-out bits C_{n-1}^- and C_{n-1}^+ that are employed for the derivation of the sum bits on the least-significant position zero is trivial

$$C_{n-1}^+ = \overline{C_{n-1}^-} = \overline{G_{n-1:0}} = C_{-1}^+. \quad \ldots(6)$$

A = 5Want to compute (A+B) mod 17 = (5+6)mod 17=11

B = 6 **Diminished-1 sum** = 11-1 = 10

Input Diminished-1 representations

A^* = A – 1 = 4 = 0 1 0 0

B^* = B – 1 = 5 = 0 1 0 1

Give operands to a modulo 15 adder

0 1 0 0 (4)

0 1 0 1 (5)

H_i $\overline{0\ 0\ 0\ 1}$

$H_{i:0}$ 0 0 0 1 XOR

Sum mod 15 1 0 0 1 (9)

Diminished-1 $\overline{1\ 0\ 1\ 0}$ INV (10)

Sum mod 17

Fig. 10. An arithmetic example of the new approach for the modulo $2^n + 1$ sum via the result of the corresponding modulo $2^n - 1$ addition.

In this case both carries are considered as incoming carries from bit position -1.For all other bit positions with 0≤ i ≤ n – 1, the relation between C_i^- and C_i^+ is given by the following theorem.

Theorem 2. $C_i^+ = C_i^- \oplus (\overline{G_{i:0}} \cdot P_{i:0})$, with i < n-1.

Proof: By the definition of the XOR operator $C_i^- \oplus (\overline{G_{i:0}} \cdot P_{i:0})$ can be rewritten as

$$C_i^- \cdot (G_{i:0} + \overline{P_{i:0}}) + \overline{C_i^-} \cdot \overline{G_{i:0}} \cdot P_{i:0}$$

Replacing C_i^- with its definition in the above relation we get

$$(G_{i:0} + P_{i:0} \cdot G_{n-1:i+1}) \cdot (G_{i:0} + \overline{P_{i:0}}) + \overline{G_{i:0}} \cdot (\overline{P_{i:0}} + \overline{G_{n-1:i+1}}) \cdot \overline{G_{i:0}} \cdot P_{i:0}$$

$= G_{i:0} + \overline{G_{i:0}} \cdot P_{i:0} \cdot \overline{G_{n-1:i+1}}$

$= G_{i:0} + P_{i:0} \cdot \overline{G_{n-1:i+1}}$.

Based on modulo $2^n - 1$ addition equation (5), the derived term is the definition of the carry signal C_i^+ in the case of diminished-1 modulo $2^n + 1$ addition.

The direct consequence of the newly derived relationship is that we can compute the carries for the case of modulo $2^n + 1$ adders directly from the a modulo $2^n - 1$ carry computation unit by a stage of XOR gates that will combine the carries C_i^- with terms $\overline{G_{i:0}} \cdot P_{i:0}$.

At first, it may seem complicated to compute $\overline{G_{i:0}} \cdot P_{i:0}$ since it requires a complete carry tree to be added for the computation of $G_{i:0}$. However, based on theorem 2 the computation of $\overline{G_{i:0}} \cdot P_{i:0}$ is straightforward and can be implemented at low cost.

Theorem 3 . $\overline{G_{i:0}} \cdot P_{i:0} = H_{i:0}$, where $H_{i:0} = H_i \cdot H_{i-1} \cdots H_0$.

Proof: By definition we know that $G_{i:0} = G_i + P_i \cdot G_{i-1:0}$ and that $P_{i:0} = P_i \cdot P_{i-1:0}$. Therefore, the term $\overline{G_{i:0}} \cdot P_{i:0}$ written as

$$\overline{G_i} \cdot (\overline{P_i} + \overline{G_{i-1:0}}) \cdot P_i \cdot P_{i-1:0} = \overline{G_i} \cdot P_i \cdot \overline{G_{i-1:0}} \cdot P_{i-1:0}.$$

The term $\overline{G_i} \cdot P_i$ can be easily proven that is equal to H_i.

Hence,

$$\overline{G_{i:0}} \cdot P_{i:0} = H_i \cdot (\overline{G_{i-1:0}} . P_{i-1:0}).$$

In the manner, the term in the parentheses $\overline{G_{i-1:0}} \cdot P_{i-1:0} = H_{i-1} . \overline{G_{i-2:0}} . P_{i-2:0}$leading to

$$\overline{G_{i:0}} \cdot P_{i:0} = H_i \cdot H_{i-1}(\overline{G_{i-1:0}} \cdot P_{i-1:0}).$$

Applying the same rule recursively n times we get

$$\overline{G_{i:0}} \cdot P_{i:0} = H_i \cdot H_{i-1} \cdot \ldots \cdot H_0 = H_{i:0}.$$

Therefore, from the two namely introduced theorems, the diminished-1 modulo $2^n + 1$ sum can be derived from the corresponding modulo $2^n - 1$ sum as follows: by definition, we know that

$$S_i^+ = H_i \oplus C_{i-1}^+.$$

Replacing $C_{i-1}^+$with its new value $C_{i-1}^- \oplus H_{i-1:0}$ we get that

$$S_i^+ = H_i \oplus C_{i-1}^- \oplus H_{i-1:0}.$$

We identify that $H_i \oplus C_{i-1}^-$ is the corresponding sum S_i^-.

Thus, it holds that

$$S_i^+ = S_i^- \oplus H_{i:0} \text{ for } i \neq 0 \quad \ldots(7)$$

Also based (6), the sum bit S_0^+ is simply equal to $\overline{S_i^-}$. An arithmetic example illustrating the derivation of a diminished-1 modulo 17 sum via a modulo 15 adder and some extra logic is given in figure.

According to section 3, one or both the input operands in case of diminished-1 representation may be equal to zero. It is decided to handle this case by setting the corresponding diminished-1 carries C_i^+ to zero. However, when using a modulo $2^n - 1$ adder for implementation of a diminished-1 adder an even simpler approach can be employed: when at least of the input operand is zero, i.e., $a_z = 1$ or $b_z = 1$, then we ignore the term $H_{i-1:0}$ for the derivation of the bit S_i^+ and keep the original sum of the modulo $2^n - 1$ adder. This simple condition can be efficiently implemented by the following equations:

$$S_i^+ = S_i^- \oplus H_{i-1:0} \cdot \overline{(a_z + b_z)} \text{ for } i \neq 0 \text{ and } \ldots(8)$$

$$S_0^+ = S_0^- \oplus \overline{(a_z + b_z)}.$$

5.2 Modified Pre Processing Stages in Both Architectures

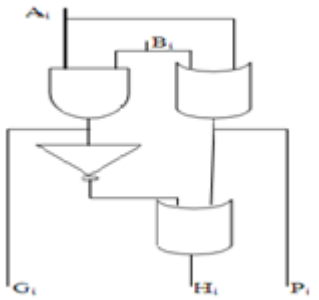

Fig. 11. Design of a Modified Pre Processing Stage

Here Boolean expressions are used to reduce the gate count. In normal pre processing stage contain 7 gates and modified pro processing stage are reduced to 3 gates and it contain only 4 gates to produce generate, propagate, and half sum bits.

6 Simulation And Implementation Results

Initially the VHDL coding is performed to the design and it is then implemented in XILINX ISE 9.1E kit. The implementation is performed with 8-bit input. The experimental results for the parameters namely power, delay, frequency and LUT count obtained for modulo 2^8+1 design are obtained. For the parallel addition operation, three stages are used. Thus the first stage(pre processing stage) are modified has less amount of power consumption compared to the earlier method. The results are obtained and are tabulated as follows.

PARAMETER	EXISTING PRE PROCESSING STAGE	MODIFIED PRE PROCESSING STAGE
POWER(mw)	3740	3726
GATE	7	4
DELAY(ns)	6.320	6.320
FERQUENCY(hz)	450	450

Table 1: Implementation for Pre Processing Stage

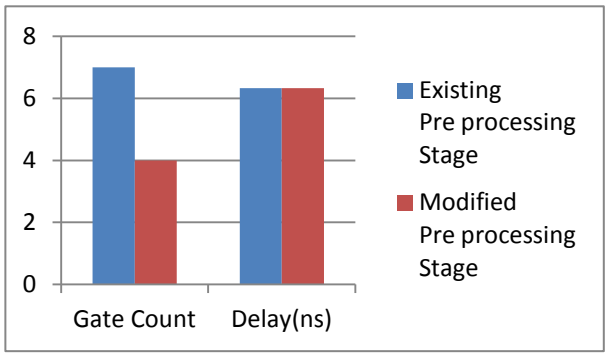

Table 2: Comparision of Gate count and delay for Existing and Modified Pre-processing stage

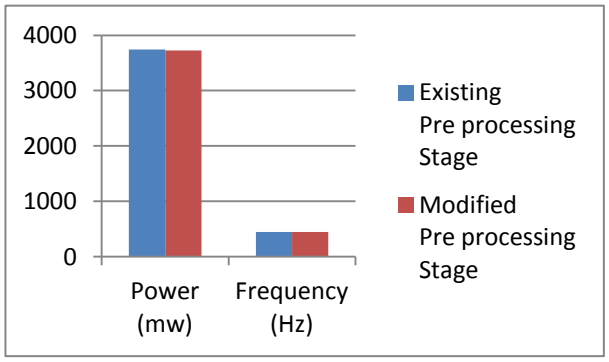

Table 3: Comparison of Power and Frequency for Existing and Modified Pre-processing stage

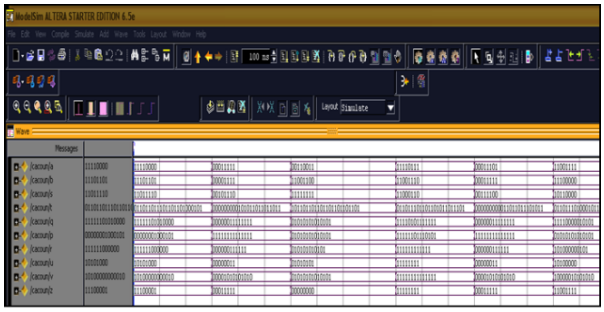

Fig. 12.Unified Design for a Prefix Modulo 2^8 – 1 Adder

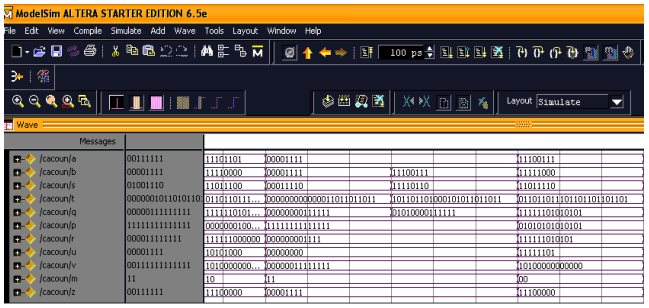

Fig. 13. Unified Design for a Prefix Modulo 2^8 + 1 Adder

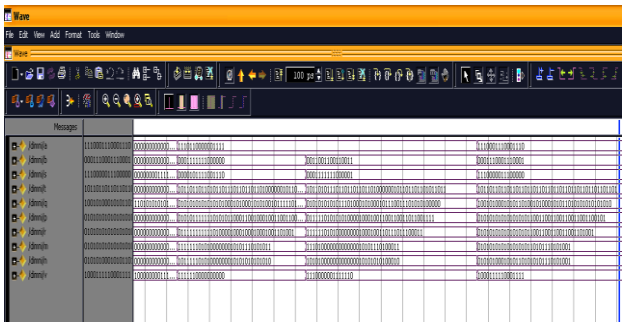

Fig. 11. Design for a diminished-one modulo 2^{16}+1adder Using CSB

7 CONCLUSIONS

Power efficient modulo 2^n + 1 adders are appreciated in a variety of computer applications such as cryptography. In this paper, two modified contributions are offered to the modulo 2^n + 1 addition problem.

A novel architecture has been modified the sparse totally regular parallel-prefix carry computation unit. This architecture was modified by using the Boolean expressions are reduced to parallel-prefix carry computation unit in modulo 2^n + 1 addition. The experimental results indicate that the modified architecture approximately decrease or increase the earlier solutions in implementation LUT'S and power consumption, while maintain the same operating speed and delay.

The modulo 2^n + 1 addition problem was also shown to be related to the modulo 2^n – 1 addition problem. The unified theory presented in this paper shown that a simple post processing stage composed of an XOR gate for each output bit needs to be added to a modulo 2^n – 1 adder for attaining a modulo 2^n + 1 adder.

8. REFERENCES

[1] www.cs.kent.edu/~rothstei/modular_arith.ppt

[2] http://www.cs.odu.edu/~cs772/fall04/lectures /public_key_cryptography.html

[3] G. Dimtra Kopulous, D.G. Nikolos, D. Nikolos, H.T. Vergos, and C. Efstathiou,' New Architectures for Modulo 2^n-1 Adders," proc. IEEE Int'1 Conf. Electronics, Circuits, and Systems, 2008.

[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.

[5] R. Zimmerman," Efficient VLSI Implementation of Modulo (2^n±1) Addition and Multiplication," proc. 14^{th} IEEE symp. Computer Arithmetic, pp. 158-167, Apr. 1999.

[6] R. Zimmerman,"Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," PhD dissertation, Swiss Fed. Inst of Technology, 1977.

[7] H.T. Vergos, C. Efstathiou, and D. Nikolos," Diminished-One Modulo 2^n+1 Adder Design," IEEE Trans. Computers, vol. 51, no. 12, pp. 1389-1399, Dec. 2002.

[8] C. Efstathiou, H.T. Vergos, and D. Nikolos," Modulo 2^n±1 Adder Design Using Select Prefix Blocks," IEEE Trans. Computers, vol.52, no. 11, pp. 1399-1406, nov. 2003.

[9] S.-H. Lin and M.-H. Sheu," VLSI Design of Diminished-One Modulo 2^n+1 Adder Using Circular Carry Selection," IEEE Trans. Circuits and Systems II, vol. 55, no. 9, pp. 897-901, Sept. 2008.

[10] A. Hissat, "High-Speed and Reduced-Area Modular Adder Structures for RNS," IEEE Trans. Computers, vol. 53, no. 1, pp. 84-89, Jan. 2002.

[11] C. Efstathiou, H.T. Vergos, and D. Nikolos," Fast Parallel-Prefix Modulo 2^n+1 adders," IEEE Trans. Computers, vol. 53, no. 9, pp. 1211-1216, Sept. 2004.

[12] H.T. Vergos, and C. Efstathiou," A Unifying Approach for Weighted and Diminished-1 Modulo 2^n+1 Addition," IEEE Trans. Circuits and Systems II, vol.55, no. 10, pp. 1041-1045, Oct. 2008.

[13] L. kalampoukas et al., "High-speed Parallel-Prefix Modulo 2^n-1 Adders," IEEE Trans. Computers, vol. 49, no.7, pp. 673-680, July 2000.

[14] R.A. Patel, M. Bonaissa, and S. Boussakta,"Fast Parallel-Prefix Architecture for Modulo 2^n-1 Addition with a Single Representation of Zero," IEEE Trans. Computers, vol. 56, no. 11, pp. 1484-1492, Nov. 2007.