Modulo Multiplier by using Radix-8 Modified Booth Algorithm

B.Sreekanth
Post Graduate Scholar, Indur institute of Engineering & Technology, Siddipet

K.Padmavathi
Associate Professor, Indur institute of Engineering & Technology, Siddipet
*

**Abstract - Modular arithmetic operations (inversion, multiplication and exponentiation) are used in several cryptography applications. RSA and elliptic curve cryptography (ECC) are two of the most well established and widely used public key cryptographic (PKC) algorithms. The encryption and decryption of these PKC algorithms are performed by repeated modulo multiplications. These multiplications differ from those encountered in signal processing and general computing applications in their sheer operand size. Key sizes in the range of 512~1024 bits and 160~512 bits are typical in RSA and ECC, respectively. Hence, the long carry propagation of large integer multiplication is the bottleneck in hardware implementation of PKC. The residue number system (RNS) has emerged as a promising alternative number representation for the design of faster and low power multipliers owing to its merit to distribute a long integer multiplication into several shorter and independent modulo multiplications. RNS has also been successfully employed to design fault tolerant digital circuits. A special moduli set of forms {2n-1, 2n, 2n +1} are preferred over the generic moduli due to the ease of hardware implementation of modulo arithmetic functions as well as system-level inter-modulo operations, such as RNS-to-binary conversion and sign detections. To facilitate design of high- speed full-adder based modulo arithmetic units, it is worthwhile to keep the moduli of a high-DR RNS in forms of {2n-1, 2n, 2n +1}.The modulo 2n-1 multiplier is usually the noncritical datapath among all modulo multipliers in such high-DR RNS multiplier. With this precept, a family of radix-8 Booth encoded modulo 2n-1 multipliers, with delay adaptable to the RNS multiplier delay, is proposed. The modulo 2n-1 multiplier delay is made scalable by controlling the word-length of the ripple carry adder, employed for radix-8 hard multiple
generation.**

Key Words — Booth Algorithm, Radix-8, Prefix Adder, Modulo Arithmetic, Multiplier, Residue number system (RNS)

I. INTRODUCTION

RIVEST, Shamir, and Adleman (RSA) and elliptic curve cryptography (ECC) are two of the most well established and widely used public key cryptographic (PKC) algorithms. The encryption and decryption of these PKC algorithms are per- formed by repeated modulo multiplications. The Residue Number System (RNS) is a non-weighted number system that can map large numbers to smaller residues, without any need for carry propagations. Its most important property is that additions, subtractions, and multiplications are inherently carry-free. These arithmetic operations can be performed on residue digits concurrently and independently. Thus, using residue arithmetic, would in principle, increase the speed of computations RNS has shown high efficiency in realizing special purpose applications such as digital filters , image processing , RSA cryptography and specific applications for which only additions, subtractions and multiplications are used and the number dynamic range is specific. Special moduli sets have been used extensively to reduce the hardware complexity in the implementation of converters and arithmetic operations. Among which the triple moduli set {2n+1,2n,2n-1} have some benefits. Since the operation of multiplication is of major importance for almost all kinds of processors, efficient implementation of multiplication modulo 2n-1 is important for the application of RNS. A residue number system is characterized by a base that is not a single radix but an N-tuple of integers (mN,mN-1 … m1). Each of these mi (i = 1, 2, … N) is called a modulus. An integer

"X" is represented in the residue number system by N-tuple (xN, xN-1… x1) where xI is a nonnegative integer satisfying

$$X = m_I * q_I + x_I \ , \ldots\ldots\ldots\ldots \quad (1)$$

where q_I is the largest integer such that $0 <= x_I <= (m_I - 1)$. x_i is known as the residue of X modulo m_i, and notations X mod m_i and $|X|_{mi}$ are commonly. Modular Multiplication is the key algorithm of RSA and other public key cryptosystems, and so provides an indication of the efficiency of the RNS implementation. The majority of the currently established Public-Key Cryptosystems (RSA, Difie-Hellman, Digital Signature Algorithm (DSA), Elliptic Curves (ECC), etc.) require modular multiplication in finite fields as their core operation which accounts for up to 99% of the time spent for encryption and decryption. In order to improve the performance of the overall cryptosystem, it is therefore crucial to optimize modular multiplication.

Modulo arithmetic is also widely used in various applications such as digital signal processing where the residue arithmetic is used for digital filter design. Also, the number of wireless and internet communication nodes has grown rapidly. The confidentiality and the security of the data transmitted over these channels has becoming increasingly important.

Cryptographic algorithms like International Data Encryption Algorithm (IDEA) are frequently used for secured transmission of data. Modulo 2^n+1, 2^n, 2^n-1 addition and multiplication are the crucial operations in the IDEA algorithm and also modulo 2^n+1 arithmetic operations are used in Fermat number transform computation. Moduli choices of the forms $\{2^n+1, 2^n, 2^n-1\}$ have received significant attention because they offer very efficient circuits when considering the area time^2 product and efficient converters from and to the binary system. Therefore, designing efficient modulo 2^n-1 multipliers is an interesting issue. Modulo 2^n-1 multiplication is used extensively in Residue Number System (RNS) based Digital Signal Processing (DSP) and cryptography units.

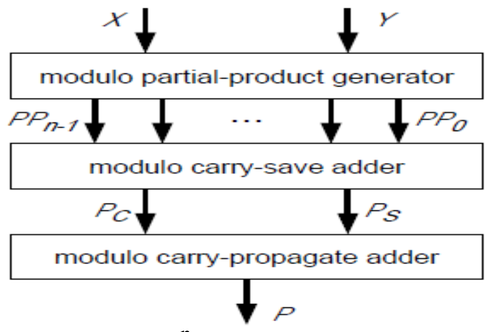

X
Y
modulo partial-product generator
PP_{n-1}
...
PP_0
modulo carry-save adder
P_C
P_S
modulo carry-propagate adder
P

Figure 1 : Modulo (2^n-1) multiplier architecture

II. General Classification of RNS-based Multipliers:-

Modular RNS-based multipliers can be classified into three main groups. I. The first group deals with specific moduli, i.e. 2n-1, 2n, or 2n+1.

II. The second type uses any moduli value and utilizes special ROM architectures in order to implement the multiplier and since a ROM is involved, this type of multipliers tend to be very impractical since memory size tend to be very huge for very large moduli values.

III. The third group of multipliers handle medium to large values of moduli but it uses mainstream arithmetic components that have been developed beforehand, thus facilitating the job of the hardware designer by reducing the overall project lifespan. These components could be regular binary multipliers, adders, sub tractors, logic components and small size ROM architectures.

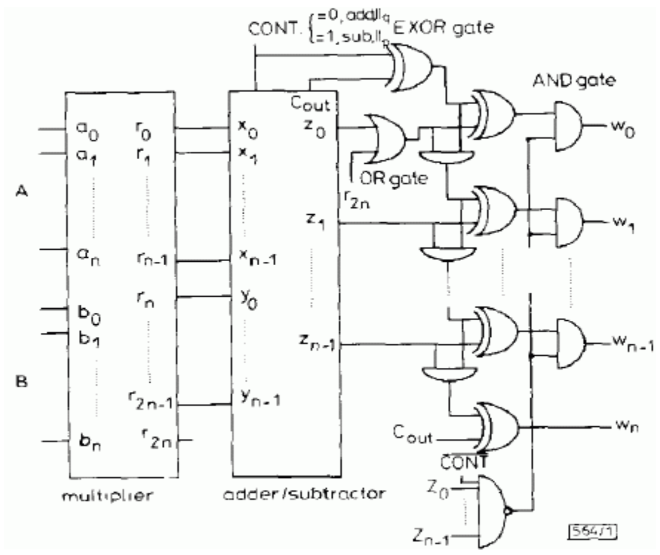


Figure 2: Type-1 RNS-based modular multiplier architecture

III. RADIX-8 BOOTH ENCODED MODULO 2N-1 MULTIPLICATION ALGORITHM

To ensure that the radix-8 Booth encoded modulo multiplier does not constitute the system critical path of a high-DR moduli set based RNS multiplier, the carry propagation length in the hard multiple generation should not exceed n-bits. To this end, the carry propagation through the Has can be eliminated by making the end-around-carry bit c7 a partial product bit to be accumulated in the CSA tree. This technique reduces the carry propagation length to n bits by representing the hard multiple as a sum and a redundant end-around-carry bit pair. The resultant end-around-carry bits in the partial product matrix may lead to a marginal increase in the CSA tree depth and consequently, may aggravate the delay of the CSA tree. In which case, it is not sufficient to reduce the carry propagation length to merely bits using the above technique.

IV GENERATION OF PARTIALLY-REDUNDANT HARD MULTIPLE

Let |X2n-1 and 2X2n-1be added by a group of M=(n/k) k-bit RCAs such that there is no carry propagation between the adders. Below figure shows this addition for n=8 and k=4.

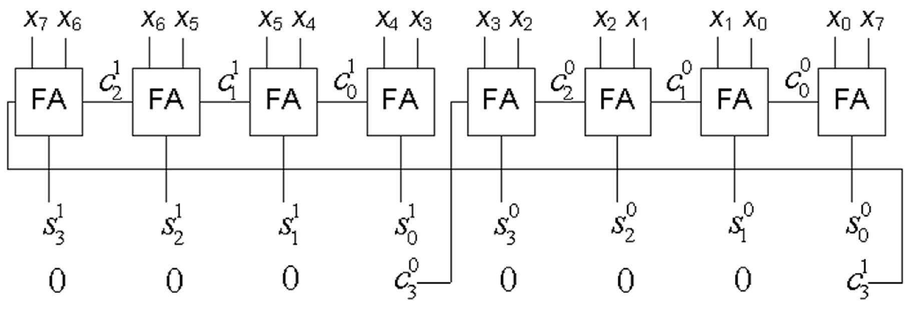


Figure 3: Generation of partially-redundant |+3X 2 -1 using n k-bit RCAs.

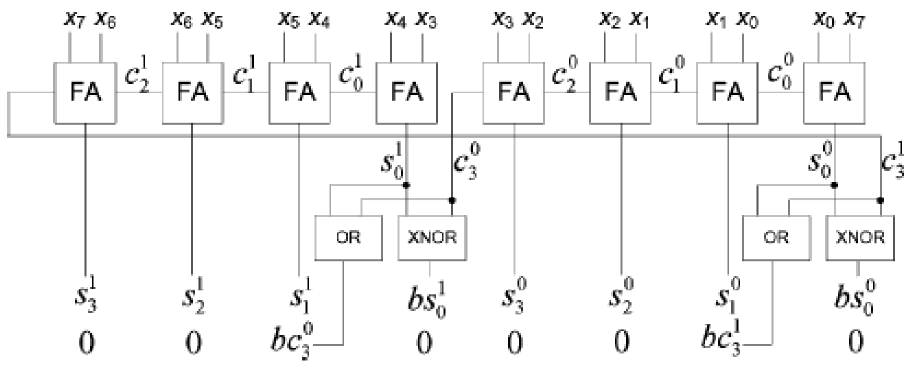


Figure 4: Generation of partially-redundant |B+3X 2n -1 |

Where the c_3^1 sum and carry-out bits from the RCA block are represented as s_i^j and c_i^j for $i \in [0, k-1]$ and $j \in [0, M-1]$ respectively. In Fig. 6, the carry-out of RCA 0, c_3^0 is not propagated to the carry input of RCA 1 but preserved as one of the partial product bits to be accumulated in the CSA tree. The binary weight of the carry-out c_3^1 of RCA 1 has, however, exceeded the maximum range of the modulus and has to be modulo reduced before it can be accumulated by the CSA tree.By Property 2, the binary weight of c_3^1 can be reduced from 28 to 20. Thus, is inserted at the least significant bit (lsb) position. It should be stressed that the carry-out is a partial carry propagated through only most significant FAs and hence, is different from the end-around-carry bit in the modulo 2n-1addition of X and 2X i.e., c7 , the partially-redundant form of |+3X 2n-1 is given by the partial-sum and partial-carry pair (S, C) where

$$S = s_{k-1}^{M-1} s_{k-2}^{M-1} \cdots s_0^{M-1} \cdots s_{k-1}^{0} s_{k-2}^{0} \cdots s_0^{0} \qquad \bar{S} = \bar{s}_{k-1}^{M-1} \bar{s}_{k-2}^{M-1} \cdots \bar{s}_0^{M-1} \cdots \bar{s}_{k-1}^{0} \bar{s}_{k-2}^{0} \cdots \bar{s}_0^{0}$$

$$C = \underbrace{0\cdots0}_{k-1} c_{k-1}^{M-2} \cdots \underbrace{0\cdots0}_{k-1} c_{k-1}^{0} \underbrace{0\cdots0}_{k-1} c_{k-1}^{M-1}. \qquad \bar{C} = \underbrace{1\cdots1}_{k-1} \bar{c}_{k-1}^{M-2} \cdots \underbrace{1\cdots1}_{k-1} \bar{c}_{k-1}^{0} \underbrace{1\cdots1}_{k-1} \bar{c}_{k-1}^{M-1}.$$

Since modulo negation is equivalent to bitwise complementation by Property 1, the negative hard multiple in a partially-redundant form, $|-3X|_{2^n-1} = (\bar{S}, \bar{C})$ is computed as follows:

To avoid having many long strings of ones in $\bar{C}$ an appropriate bias, B, is added to the hard multiple such that both C and $\bar{C}$ are sparse. The value of is chosen as

$$B = \sum_{j=0}^{M-1} 2^{k\cdot j} = \overbrace{\underbrace{0\cdots01}_{k}\cdots\underbrace{0\cdots01}_{k}}^{n}.$$

The addends for the computation of the biased hard multiple, |B+3X 2n-1 in a partially-redundant form are X2n-1 and 2X2 -1 and B or equivalently S , C and B. Since B is chosen to be a binary word that has logic ones at bit positions 2kj , and logic zeros at other bit positions. B+3X 2n-1 can be generated by simple XNOR and OR operations on the bits of and at bit positions 2kj. Illustrates how these bits in the sum and the carry outputs of RCA 0 and RCA 1 are modified. In general |B+3X 2n-1, is given by the partial-sum and partial-carry pair (BS, BC) such that

$$BS = s_{k-1}^{M-1}s_{k-2}^{M-1}\cdots bs_0^{M-1}\cdots s_{k-1}^{0}s_{k-2}^{0}\cdots bs_0^{0}$$
$$BC = \underbrace{0\cdots0}_{k-2}bc_{k-1}^{M-2}0\cdots\underbrace{0\cdots0}_{k-2}bc_{k-1}^{0}0\underbrace{0\cdots0}_{k-2}bc_{k-1}^{M-1}0$$

Wher

$$bs_0^j = \begin{cases}\overline{s_0^j \oplus c_{k-1}^{j-1}} & \text{when } j \neq 0\\ \overline{s_0^0 \oplus c_{k-1}^{M-1}} & \text{when } j = 0\end{cases} \qquad bc_{k-1}^j = \begin{cases} s_0^{j+1} + c_{k-1}^{j} & \text{when } j \neq M-1\\ s_0^0 + c_{k-1}^{M-1} & \text{when } j = M-1\end{cases}$$

For j= 0, 1…….M-
1. Let

$$\overline{BS} = \overline{s_{k-1}^{M-1}s_{k-2}^{M-1}}\cdots\overline{bs_0^{M-1}}\cdots\overline{s_{k-1}^{0}s_{k-2}^{0}}\cdots\overline{bs_0^{0}}$$
$$\overline{BC} = \underbrace{0\cdots0}_{k-2}\overline{bc_{k-1}^{M-2}}0\cdots\underbrace{0\cdots0}_{k-2}\overline{bc_{k-1}^{0}}0\underbrace{0\cdots0}_{k-2}\overline{bc_{k-1}^{M-1}}0$$

$$(\overline{BS}, \overline{BC})$$

It can be easily verified that the sum of (BS, BC) and modulo 2n-1 is |2B|2n-1. Therefore, $(\overline{BS}, \overline{BC})$represents the partially-redundant form of |B-3X|2n-1.

V. RADIX-8 BOOTH ENCODED MODULO MULTIPLICATION WITH PARTIALLY- REDUNDANT PARTIAL PRODUCTS

The i-th partial product of a radix-8 Booth encoded modulo 2n-1 multiplier is given by

$$PP_i = \left|2^{3i}\cdot d_i \cdot X\right|_{2^n-1}.$$

To include the bias B necessary for partially-redundant representation of PPi, (12) is modified to

$$PP_i = \left|2^{3i}\left(B + d_i \cdot X\right)\right|_{2^n-1}.$$

Using Property 3, the modulo 2n-1 multiplication by 23i , in (13) is efficiently implemented as bitwise circular-left-shift of the biased multiple,(B+ di . X). For n=8, k=4, Fig. 9 illustrates the partial product matrix of |X .Y|28-1 with $(\lfloor n/3\rfloor + 1)$ partial products in partially- redundant representation. Each PPi consists of an n-bit vector, ppi7, ppi1, ppi0 and a vector of n/k=2, redundant carry bits qi1,qi0 . Since qi0 and qi1 are the carry-out bits of the RCAs, they are displaced by k-bit positions for a given PPi. The bits, qij is displaced circularly to the left of q(i-1)j by 3 bits, i.e., q20 and q21 are displaced circularly to the left of q10 and q11 by 3 bits, respectively q10 and q11 are in turn displaced to the left of q00 and q01 by 3 bits, respectively. The last partial product in the Compensation Constant (CC) for the bias introduced in the partially- redundant representation.

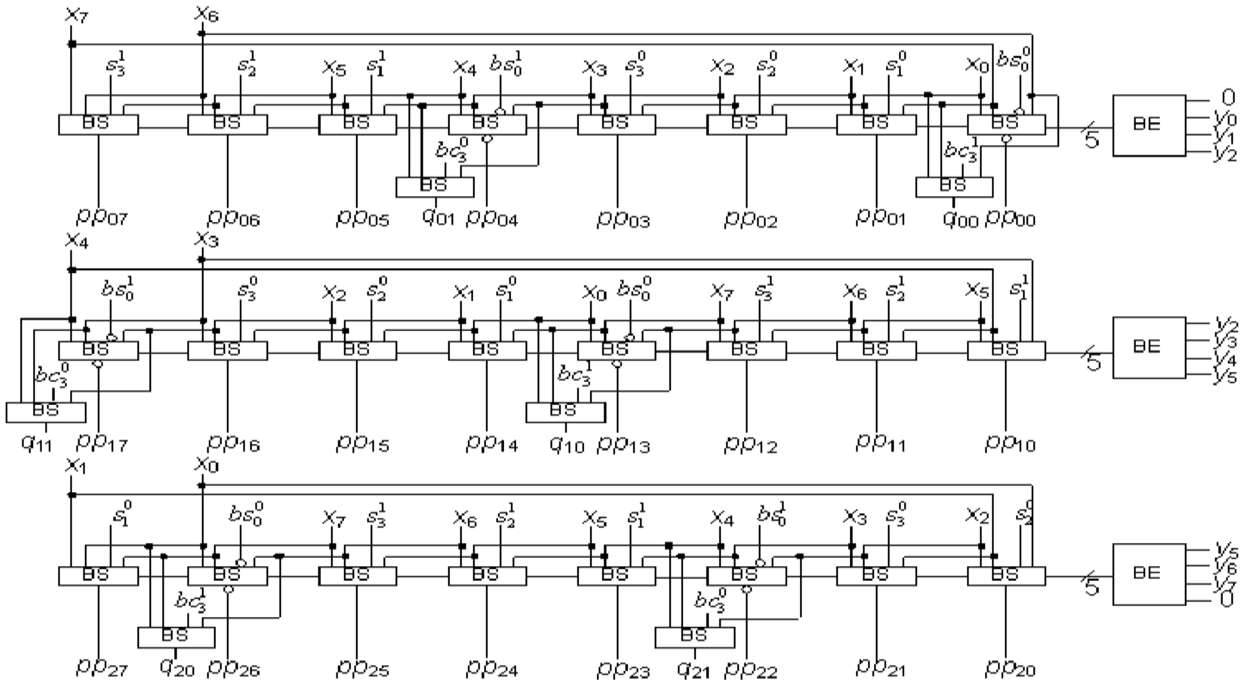


Figure. 5. Modulo-reduced partial product generation.

The generation of qij the modulo-reduced partial products, PP0, PP1, and PP2, in a partially-redundant representation using Booth Encoder (BE) and Booth Selector (BS) blocks are illustrated in Fig. 5. The BE block produces a signed one-hot encoded digit from adjacent overlapping multiplier bits as illustrated in Fig. 6(a). The signed one-hot encoded digit is then used to select the correct multiple to generate PPi. A bit-slice of the radix-8 BS for the partial product bit, ppij is shown in Fig. 6(b). As the bit positions of do not overlap, as shown in Fig. 5, they can be merged into a single partial product for accumulation. The merged partial products, PPi and the constant CC are accumulated using a CSA tree with end-around-carry addition at
each CSA level and a final two-operand modulo 2n-1 adder

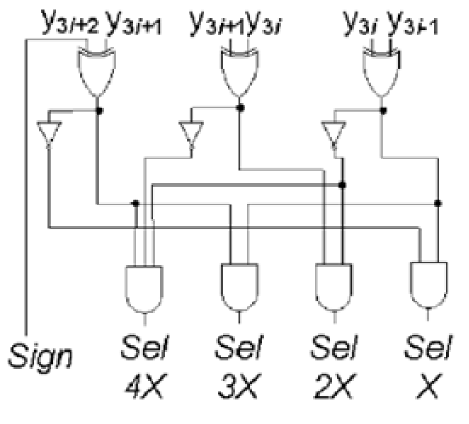


Figure. 6(a) Bit-slice of Booth Encoder (BE)

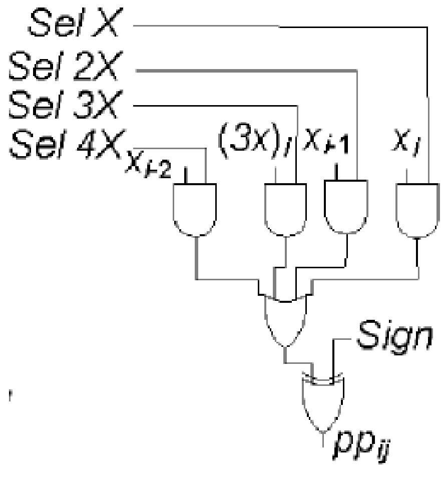


Figure. 6(b) Bit-slice of Booth Selector (BS

VI. PARALLEL PREFIX ADDER

To humans, decimal numbers are easy to comprehend and implement for performing arithmetic.
However, in digital systems, such as a microprocessor, DSP (Digital Signal Processor) or ASIC (Application-Specific Integrated Circuit), binary numbers are more pragmatic for a given computation. This occurs because binary values are optimally efficient at representing many values. Binary adders are one of the most essential logic elements within a digital system.

VI.I Modulo (2n-1) addition: -
Modulo (2n-1) addition is the same as one's complement addition can be formulated as

$$(A+B) \bmod (2^n-1) = \begin{cases} A+B-(2^n-1) \\ \quad = (A+B+1) \bmod 2^n \\ \qquad \text{if } A+B \geq 2^n-1 \\ A+B \quad \text{otherwise} \end{cases}$$

The modulo 2n reduction is automatically performed if an n-bit adder is used. Note that the value "11…..1" never occurs and that only one single representation "00…..0" of zero exists. The equation (14) can be rewritten using the condition A+B ≥ 2n

=

$$(A+B) \bmod (2^n-1) = \begin{cases} A+B-(2^n-1) \\ \quad = (A+B+1) \bmod 2^n \\ \qquad \text{if } A+B \geq 2^n \\ A+B \quad \text{otherwise} \end{cases}$$

Now, zero has a double representation ("00…..0" and "11…..1")..Since the new condition A+B ≥ 2n is equivalent to cout=1,where cout is the carryout of the addition A + B, equation (5.3.2) can be rewritten as (A+B) mod=(A+B+ cout) mod2n Therefore, modulo (2n-1) addition with a double representation of zero can be realized by the n-bit end-around-carry parallel-prefix adder of with cin = cout .The additional condition of A + B = 2n-1 = 11 …. 1 found in is equivalent to $P^m_{n-1:0} = 1$

6.1. A PARALLEL PREFIX ADDER CAN BE SEEN AS A 3-STAGE PROCESS

Pre-computation:
In pre-computation stage, each bit computes its carry generate (g)/propagate (p) signals and a temporary sum as below. These two signals are said to describe how the Carry-out signal will be handled.

$$g_i = a_i \cdot b_i$$
$$p_i = a_i \oplus b_i$$
$$c_i + 1 = g_i + p_i \cdot c_i$$

Prefix:
In the prefix stage, the group carry generate/propagate signals are computed to form the carry
chain and provide the carry-in for the adder below. Various signal graphs/architectures can be used to calculate the carry-outs for the final sum those are Sklansky,Kogge-Stone prefix tree, Brent-kung, Ladner-Fischer,Han-Carlson Prefix Tree

$$s_i = p_i \oplus c_i$$

Post-computation:
In the post-computation stage, the sum and carry-out are finally produced. The carry-out can be omitted if only a sum needs to be produced.

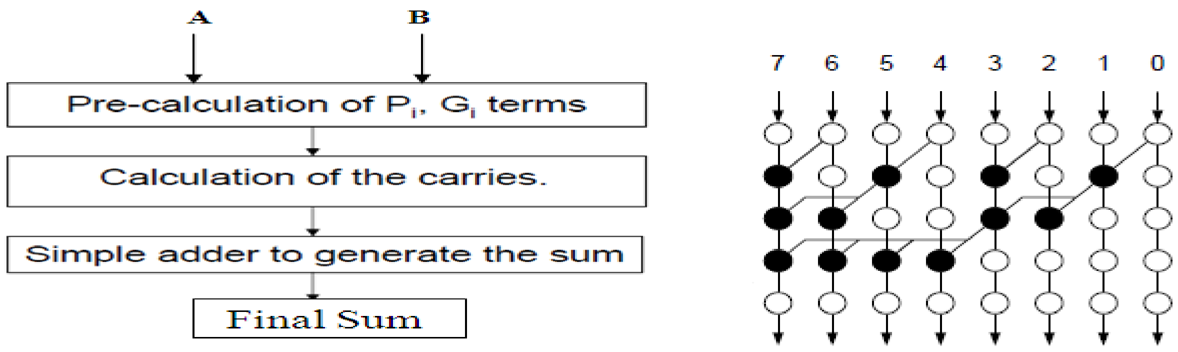

A
B
Pre-calculation of P_i, G_i terms
Calculation of the carries.
Simple adder to generate the sum
Final Sum
7
6
5
4
3
2
1
0

Figure 7: Parallel Prefix Addition Process Steps

Proposed Prefix Theme
In a prefix problem, n inputs xn-1, xn-2 ….. .x0 and an arbitrary associative operator " ● " are used compute n outputs $y_i = x_i \bullet x_{i-1} \bullet \cdots \bullet x_0$ for i=0,1,2….n-1.Thus each output yi is dependent on all inputs xj of same or lower magnitude (j ≤ i) .Carry propagation in binary addition is a prefix problem. The n-bit carry propagate addition with input operands A and B, carry-in cin, sum output S, and carry-out cout can be expressed by the logic equations:

$$(c_{out}, S) = 2^n c_{out} + S = A + B + c_{in}$$

6.2 PRE-PROCESSING:

PREFIX COMPUTATION:

$$\begin{aligned} g_i &= \begin{cases} a_0 b_0 + a_0 c_0 + b_0 c_0 & \text{if } i = 0 \\ a_i b_i & \text{otherwise} \end{cases} \\ p_i &= a_i \oplus b_i \end{aligned}$$

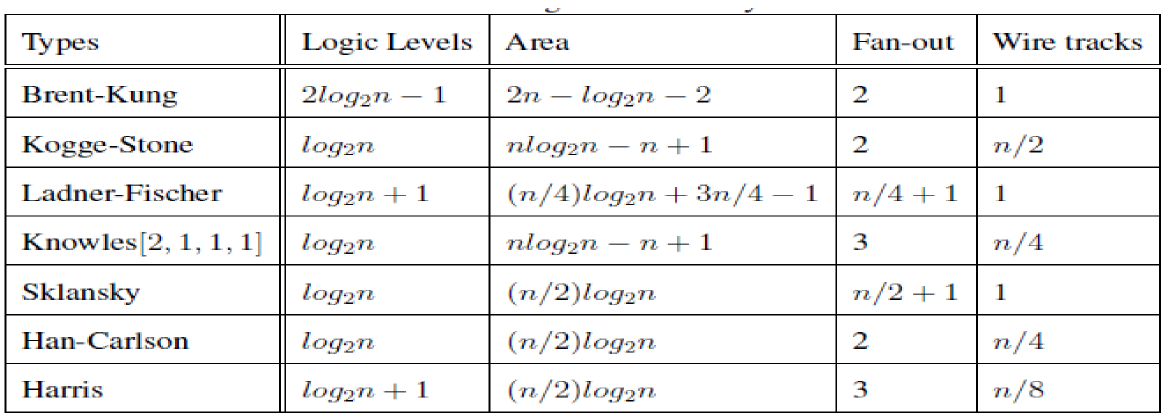
$$\begin{aligned} (G^0_{i:i}, P^0_{i:i}) &= (g_i, p_i) \\ (G^l_{i:k}, P^l_{i:k}) &= (G^{l-1}_{i:j+1}, P^{l-1}_{i:j+1}) \bullet (G^{l-1}_{j:k}, P^{l-1}_{j:k}) \\ &= (G^{l-1}_{i:j+1} + P^{l-1}_{i:j+1} G^{l-1}_{j:k}, P^{l-1}_{i:j+1} P^{l-1}_{j:k}) \end{aligned}$$

POST- PROCESSING :

$$\begin{aligned} c_{i+1} &= G^m_{i:0} \\ s_i &= p_i \oplus c_i \end{aligned}$$

For i=0, 1,2….n-1, l=1,….m, and 0≤ k≤ j≤ i where ai and bi are the operand input signals, gi and pi the generate and propagate, ci the carry, and si the sum output signals at bit position i. c0 and cn correspond to the carry-in and carry-out cout , respectively $G^l_{i:k}$ and $P^l_{i:k}$ denote the group generate and propagate signals for the group of bits i,…., k at level l. The "●" operator is repeatedly applied according to a given prefix structure of m levels in order to compute the group generate signal $G^m_{i:0}\ (= c_{i+1})$ for each bit position i

6.3 PARALLEL PREFIX ADDER ALGORITHMIC ANALYSIS



Types	Logic Levels	Area	Fan-out	Wire tracks
Brent-Kung	$2log_2n - 1$	$2n - log_2n - 2$	2	1
Kogge-Stone	log_2n	$nlog_2n - n + 1$	2	$n/2$
Ladner-Fischer	$log_2n + 1$	$(n/4)log_2n + 3n/4 - 1$	$n/4 + 1$	1
Knowles[2, 1, 1, 1]	log_2n	$nlog_2n - n + 1$	3	$n/4$
Sklansky	log_2n	$(n/2)log_2n$	$n/2 + 1$	1
Han-Carlson	log_2n	$(n/2)log_2n$	2	$n/4$
Harris	$log_2n + 1$	$(n/2)log_2n$	3	$n/8$

VII. CONCLUSION

In conclusion, a new approach for multiplication modulo (2n- 1) is proposed. Similar to the binary multiplier, the generation of the partial products is accomplished by AND gates. The Wallace tree is applied to reduce the speed for compression of column size from N to two. To completely utilize the unequal delay of a full adder, an algorithm for delay optimization of the Wallace tree is developed. The proposed approach exhibits superior performance, in terms of either speed of hardware requirement, in comparison with a recent counterpart for the same purpose. In addition, the proposed multiplier modulo (2n- 1) shows an extremely regular structure and is very suitable for VLSI implementation.

REFERENCES

[1] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public key cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] V. Miller, “Use of elliptic curves in cryptography,” in Proc. Advances in Cryptology- CRYPTO’85, Lecture Notes in Computer Science, 1986, vol. 218, pp. 417–426.

[3] R. Zimmermann, “Efficient VLSI implementation of modulo 2^n-1/2^n+1 addition and multiplication,” in Proc. 14th IEEE Symp. Computer Arithmetic, Adelaide, Australia, Apr. 1999, pp. 158–167.

[4] C. Efstathiou, H. T. Vergos, and D. Nikolos, “Modified Booth modulo 2^n- multiplier,” IEEE Trans. Comput., vol. 53, no. 3, pp. 370–374, Mar. 2004.

[5] B. S. Cherkauer and E. G. Friedman, “A hybrid radix-4/radix-8 low power signed multiplier architecture,” IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 44, no. 8, pp. 656–659, Aug. 1997.

[6] G. Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos, and C. Efstathiou, “Newarchitectures for modulo 2^n-1 adders,” in *Proc. 12th IEEE Int. Conf. Electronics, Circuits and Systems*, Gammarth, Tunisia, Dec. 2005, pp. 1–4.

[7] R. A. Patel, M. Benaissa, and S. Boussakta, “Fast Parallel-prefix architectures for modulo 2^n-1 addition with a single representation of zero,” *IEEE Trans. Comput.*, vol. 56, no. 11, pp. 1484–1492, Nov. 2007.

[8] R. Muralidharan and C. H. Chang, “Fast hard multiple generators for radix-8 Booth encoded modulo 2^n-1and modulo 2^n+1multipliers,” in *Proc. 2010 IEEE Int. Symp. Circuits and Systems*, Paris, France, Jun. 2010, pp. 717–720.

[9] G. W. Bewick, “Fast multiplication: Algorithms and implementation,” Ph.D. dissertation, Stanford Univ., Stanford, CA, 1994.