A novel modulo $(2^n + 1)$ multiplication approach for IDEA cipher

Sourav Mukherjee and Bibhudutta Sahoo

***Abstract*—** This paper covers the FPGA implementation of the International Data Encryption Algorithm (IDEA) using Very Large Scale Integrated Circuits Hardware Description Language (VHDL) with device as Vertex II Pro XC2VP30 using Xilinx – ISE 10.1. IDEA is very much fast and entirely based on internal group operations-XOR, modulo addition and modulo multiplication. So unlike other symmetric key block ciphers like AES or DES, there is no need for S-Boxes or P-Boxes in round operations. To use an encryption algorithm in real time applications like Cable TV, Video conferencing, the speed i.e. the data throughput rate needs to be high. The multiplication modulo $(2^n + 1)$ is the main module of this IDEA block cipher, as this module is highly computation intensive and consumes a lot of time. Due to regularity of IDEA, it has been implemented in hardware several times using different architectures. This paper mainly focuses on implementing a new algorithm and architecture for modulo $(2^n + 1)$ multiplication which takes the input in a diminished-1 form [2] and produces the product in the same form. This is a new modulo $(2^n + 1)$ approach for implementing IDEA in hardware. The proposed multiplier optimizes the time by producing n/2 partial products and handles zero values very efficiently. The performance of the proposed multiplier is analyzed in terms of time delay and circuit complexity and is compared with some existing schemes of diminished-1 modulo multipliers like Zimmerman [15], Sousa and Chaves [10] and Efstathiou [15][3].

***Keywords*—**Diminished-1 representation, IDEA cipher, Hardware Implementations, Modulo Multiplier, Partial Products.

I. Introduction

Cryptography is the art of keeping data secure from unauthorized access so as to ensure that only the intended users can access it. As computer technologies are getting advanced, more and more cryptographic applications are used in day-to-ay life. They are mainly used to support other applications which are very much sensitive to data security such as smart cards and commercial data exchange over a network. Not only for personal use but cryptographic algorithms are also very important in every aspect of professional activities. A cryptographic algorithm generally consists of some specialized arithmetic computations which are complicated in terms of time complexity. It is because of the fact that these algorithms work with large amount of data either in blocks or simply in streams. Although a single traditional CPU is enough for performing these computations, but for a machine which works as a server in a huge network gets millions of client requests for performing cryptographic operations for them individually. This makes the workload huge. The computational resources may also be limited for example in smartcards, mobile phones, handheld computers, etc. Moreover if the associated network is of high speed, the speed of the necessary cryptographic computations also needs to be taken into account. For example in transmitting audio and video data for cable TV, pay TV, video conferencing and sensitive financial and commercial data, the speed of the cryptographic module to be embedded ,needs to be very high. Moreover for security related issues in wireless and sensor networks, there is a need for separate hardware device with very high processing rate because of limited battery of the nodes and for optimizing the bandwidth efficiency. So from the viewpoint of high speed and throughput, traditional software implementations of these complicated cryptographic algorithms are not efficient in real time applications like ATM, VPN, etc. This forces the system designers to go for hardware implementation of the cryptosystems [8]. Traditionally hardware implementations are based on ASIC technology, but they are not quite affordable every time especially in monetary terms. Moreover these ASICs are not adaptable to new changes once the hardware is built. The more efficient and convenient method is to use FPGA platforms [5] which provides sufficient logics and storage elements on which any complex algorithm can be implemented. They are adaptable to new changes and their granularity matches quite well with the cryptographic algorithms.

In this paper, the cipher used is a symmetric key block cipher named IDEA. It takes its input as 64 bit plain text and gives a 64 bit cipher text as output using a 128 bit key. While working on plain text, it divides the input data into 16 bit sub-blocks and operates on each block. It is described as one of the most secure block algorithm due to its high immunity to attacks.

In this paper, we have discussed about hardware implementation of IDEA block cipher using VHDL. The main objective here is to design an efficient and fast modulo multiplier which is to be used in the entire IDEA algorithm. The organization of the rest of the paper is as follows. The previous hardware and software implementations are

covered in section II. Section III describes the IDEA cipher and its detailed operations as well as modules. Section IV describes basic idea of diminished-1 representation and the algorithm for proposed modulo (2^n+1) multiplication. In Section V, the architecture of the proposed multiplier is described. Section VI describes the performance reviews and comparisons with previous schemes and finally section VII concludes the paper.

II. PREVIOUS WORK

In spite of the fact that IDEA works with 16 bit word blocks, software implemented IDEA cannot reach the speed that is required for online encryption in high speed networks. IDEA was implemented in software by Ascom, the patent holder of IDEA, and it achieved an encryption rate of 23.53 Mbps. Helger [16] proposed an approach using the Intel Pentium II 233MHz machine and achieved an encryption rate of 32.9 Mbps. Mencer [18] proposed a design of IDEA processor which achieved 528 Mbps on 4 XC4020XL devices. The first VLSI implementation of IDEA was developed and verified by Bonnenberg [17] using a CMOS technology with an encryption rate of 44 Mbps. With a system clock frequency of 25 MHz, Curiger et al. performed 177 Mbps VLSI implementation of IDEA [4]. Wolter reported a 355 Mbps VLSI implementation [9] in 1995. This is followed by Salomao's approach of single round implementation on chip with 424 Mbps data conversion rate. In another approach [7], the modulus multiplier is optimized using temporal parallelism and implemented with VHDL with a data conversion rate of 522 Mbps with comparatively less area requirements. Later Leong proposed a 500 Mbps bit serial implementation of IDEA on Xilinx Virtex XCV300 -6 FPGA which is followed by Goldstein's approach with conversion rate of 1013 Mbps. Finally Ascom developed IDEACrypt Kernel with a speed of 720 Mbps [5]. Recently Thaduri [7] implemented IDEA cipher having a throughput of 700 Mb/s.

III. THE IDEA BLOCK CIPHER

In this section, the entire algorithm for the IDEA block cipher is elaborated. IDEA is a symmetric key cipher which was proposed by Lai and Massey [4]. The block size of data on which IDEA operates, is of 64 bit and the key size is of 128 bits. But all data operations in IDEA cipher are in 16 bit unsigned integers. The length of the incoming data should be either in normal in integer multiple of 64 bits or if not, is made by using padding bits. At the end of the algorithm, a 64 bit cipher text is created.

A. Basic structure of IDEA cipher.

IDEA is based on mixing operation of three different algebraic groups which are

- XOR (bitwise).
- Addition modulo 2^n.
- Multiplication modulo $(2^n + 1)$.

The security of IDEA depends on these three operations. The basic structure of IDEA cipher [13] is shown in Figure 1. The IDEA cipher consists of 8 rounds which are identical in nature and a last output transformation round which is similar to upper half of any round. Before the starting of 1st round, the input 64 bit plain text is divided into four 16 bit sub-blocks, X1, X2, X3 and X4 respectively. At the end of encryption phase, four 16 bit sub-blocks of cipher text is created. Each round uses six 16 bit sub-key blocks $Z_1^{(n)}$, $Z_2^{(n)}$, … , $Z_6^{(n)}$ which are made from the input 128 bit key. . The super-script n denotes the n^{th} round. The output transformation phase, which is considered as 9^{th} or the last round, uses 4 sub-keys, $Z_1^{(9)}$, $Z_1^{(9)}$, $Z_1^{(9)}$, $Z_1^{(9)}$. Every round except the 1st round uses the output sub-blocks produced from the previous round. In between every round, the 2nd and the 3rd sub-blocks are swapped. The entire algorithm uses only three different algebraic group operations which are XOR, addition modulo 2^{16} and multiplication modulo $(2^{16} + 1)$.

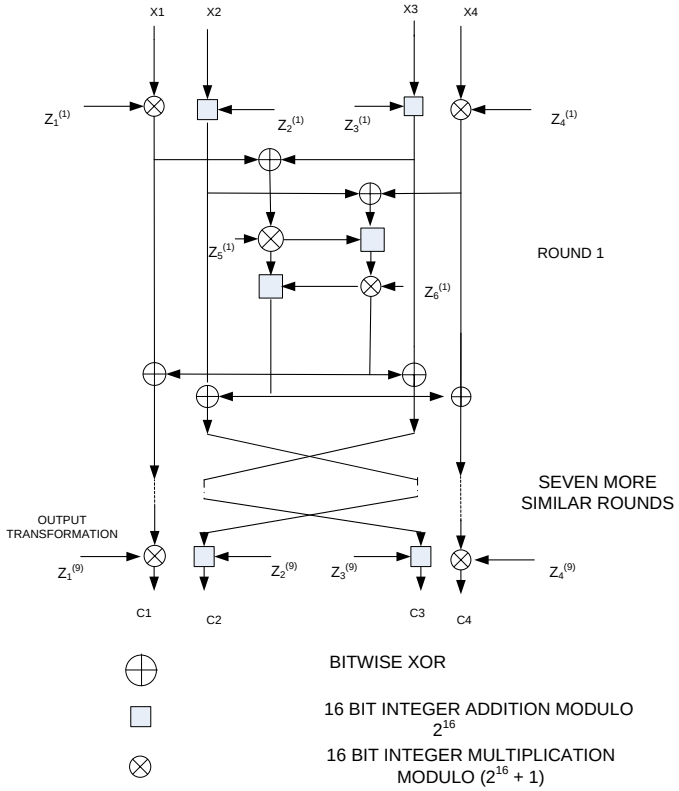

Figure 1. Basic structure of IDEA Cipher and its data flow.

The encryption phase of IDEA thus uses [(8*6) + 4] i.e. 52 sub-key blocks, which are made from the 128 bit input key. As IDEA involves only algebraic operations, no look-up tables or S-Boxes are used like DES or AES[14].

The decryption phase of IDEA is identical to that of the encryption phase. It uses the same sequence of operations as in the encryption phase. The only change is that the sub-keys are reversed and slightly different. That means the sub-keys which are used in round 1 during encryption phase are manipulated during last round of decryption phase. The sub-keys used in decryption are either additive or

multiplicative inverse of the sub-keys used in the encryption phase.

B. Key Generation

The key generation phase of IDEA generates 52 sub-keys from the 128 bit input key. The basic steps of generating the encryption keys are:

- All the sub-keys are named as $Z_1^{(1)},\ldots Z_6^{(1)}$, $Z_1^{(2)},\ldots Z_6^{(2)}$, … , $Z_1^{(8)},\ldots Z_6^{(8)}$, $Z_1^{(9)},\ldots Z_4^{(9)}$.
- From the input 128 bit key, eight sub-blocks of 16 bits are partitioned and are assigned to $Z_1^{(1)},\ldots Z_2^{(2)}$ directly.
- Now the original 128 bit key block is rotated by 25 bits and a new 128 bit block is formed. Now another eight sub-blocks are generated from this new block.
- The rotation procedure is repeated until and unless sub-blocks used in previous rounds are found.

Once the encryption keys are generated, the decryption keys can be generated directly by taking their additive or multiplicative inverses as required.

IV. DIMINISHED-1 MODULO MULTIPLIER

Multiplication modulo a Fermat prime (p) is used as an important operation in many algorithms and it is crucial in various applications like pseudorandom number generation, Arithmetic processing and Cryptography. In IDEA algorithm, this modulo multiplier plays a very important role in the throughput and speed of IDEA. More over the data flow path also contains several such multipliers. This needs an efficient design of such multiplier in hardware for FPGA implementation of IDEA.

In modulo $(2^n + 1)$ multiplication, the basic operations which are involved are *regular binary multiplication of 2 numbers* and *modulo correction*. But while realizing this operation in hardware, an inefficient usage of area and time is involved. In residue number system arithmetic and in Fermat Number Transform (FNT), the operands are of (n+1) wide when the concerned modulus is $(2^n + 1)$, because the maximum value allowed in these types of arithmetic is 2n which is normally represented by n+ 1 bit. But it appears inefficient to work with an extra bit in case of n bit operands in weighted binary representation. So in order to transform the computation units to a bit width of n bits, a new approach is introduced which maps the numbers from normal weighted binary representation into a modified binary representation, which ultimately brings down the operand width to n bits. Such modified binary representations are regarded as diminished-1 number representation. Although an extra overhead is there in such multipliers to convert from weighted to diminished-1 form, it is still efficient when a large number of modulo multiplications and additions are involved in any algorithm.

A. Basic Idea

The diminished-1 representation of binary numbers was first proposed by Leibowitz [2], which is found to be a very convenient and efficient form of representation for modulo $(2^n + 1)$ operations on binary numbers. For normal (n + 1) bit representation under modulo (2^n+1), the number 2n is expressed as -1 mod (2^n+1), which requires an extra bit. To get rid of this, this diminished-1 representation is introduced. This modified binary representation is achieved by subtracting 1 from the normal binary representation of any number. Thus if A is a normal binary number and d[A] be the diminished-1 representation of A , then

$$d[A] = (A-1) \bmod (2^n + 1) \quad \ldots\ldots (1)$$

Thus when A ϵ [1, 2^n] and A $\neq$ 0 (an n + 1 bit number), then d[A] ϵ [0; 2^n - 1],which is an n bit number. However when A = 0, d[A] = d[0] = (0 - 1) mod $(2^n + 1)$ = (-1) mod $(2^n + 1)$ which is equal to 2^n, an (n + 1) bit number.

The diminished-1 arithmetic has the following operations:-

$$d[-A] = \overline{d[A]} \quad \ldots\ldots (2)$$

$$d[A+B] = (d[A] + d[B] + 1) \bmod (2^n + 1) \quad \ldots\ldots (3)$$

$$d[A-B] = (d[A] + \overline{d[B]} + 1) \bmod (2^n + 1) \quad \ldots\ldots (4)$$

$$d[AB] = (d[A] \times d[B] + d[A] + d[B]) \bmod (2^n + 1)$$
$$= (d[A] \times B + B - 1) \bmod (2^n + 1) \quad \ldots\ldots (5)$$

$$d[2^k A] = inCLS(d[A], k) \quad \ldots\ldots (6)$$

$$d[2^k A] = inCLS(\overline{d[A]}, k) \quad \ldots\ldots (7)$$

Where (d[A])' is one's complement of d[A] and inCLS(x , k) is the k bit circular shift of x in which the circulated k bits are complemented.

B. Proposed Modulo $(2^n + 1)$ Multiplication algorithm

As the modulo multiplier is the most important module for the IDEA cipher, the basic goal is to speed up the modulo multiplication technique, which can be done in two ways:

- To reduce the number of partial products.
- To accelerate the addition of partial products.

The number of partial products can be reduced by applying Booth's recoding algorithm and the addition of these partial products can be accelerated by using Wallace trees (Carry Save Adder trees). Zimmerman [14][6] proposed modulo $(2^n + 1)$ adders which used diminished-1 number system where the normal Carry Save Adders(CSA) are replaced by Inverted End Around Carry CSA tree for modulo $(2^n + 1)$ correction

Previously, few diminished-1 modulo $(2^n + 1)$ modulo multipliers [12][11][14] were used in IDEA but handling of zero results was not considered. For some of them, a new combinational circuit was used for treatment of zeroes which was a correction term generator. But the circuit complexity was complex. . The multiplication approach by Zimmerman [14] consists of an n-bit unsigned multiplication followed by an n-bit modulo correction. Booth's recoding algorithm is

used to reduce the number of partial products. But the correction term is generated with substantial gate delay and this scheme cannot handle 2^n value efficiently. The multiplication approach of Sousa and Chaves [10] used Modified Booth's algorithm but the circuit complexity was very high. Moreover, for diminished-1 addition, carry propagate adder is used. Efstathiou [11][15][3] proposed a diminished-1 multiplier but it is without Booth recoding algorithm. It uses an $n \times n$ array of partial products and moreover there is no special treatment of zero.

In our proposed multiplier, radix 4 booth encoding is used to reduce the number of partial products to n/2. In this multiplier, both the inputs and the result are in diminished-1 form. Moreover the correction term generator is also very simple. A separate correction term is generated for each partial product. The partial product reduction is done by using an inverted end around carry (EAC) adder tree and the adder used is a carry save adder (CSA). A final diminished-1 adder is used for generating the product.

Let the multiplicand be d[A] and multiplier be d[B] where d[A] and d[B] are two n bit numbers. Let d[A] = ($a_{n-1}a_{n-1},\ldots a_1a_o$) and d[B] = ($b_{n-1}b_{n-1}\ldots b_1b_0$).Since the product is also an n bit number, let the product be d[AB] = ($p_{n-1}p_{n-2}p_{n-3}\ldots..p_1p_0$). Now in IDEA, the operand 0 is represented as 2^n. So in diminished-1 representation, 0 is treated as $2^n - 1$.

As the multiplicand and the multiplier are operands of n bits wide, d[B] can be written as

$$d[B] = (\sum_{i=0}^{n-1} b_i 2^i) \bmod (2^n + 1)$$

In radix 4 booth encoding format (where the triplets of multiplier are encoded), hence d[B] can be written as

$$d[B] = ((b_0 - 2b_1) + \sum_{i=1}^{\lfloor n/2 \rfloor} (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}) \bmod (2^n + 1)$$

…….. (8)

Now

$$B = (d[B] + 1) \bmod (2^n + 1)$$

So substituting the value of d[B], we get,

$$B = ((1 + b_0 - 2b_1) + \sum_{i=1}^{\lfloor n/2 \rfloor} (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}) \bmod (2^n + 1)$$

…….. (9)

Substituting (9) in equation (5), we get,

$$d[AB] = (d[A] \times (1 + b_0 - 2b_1) + d[A] \times \sum_{i=1}^{\lfloor n/2 \rfloor} (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}$$
$$+ \sum_{i=1}^{\lfloor n/2 \rfloor} (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i} + (1 + b_0 - 2b_1) - 1) \bmod (2^n + 1)$$

$$= (d[A] \times (1 + b_0 - 2b_1) + (1 + b_0 - 2b_1) - 1$$
$$+ \sum_{i=1}^{\lfloor n/2 \rfloor} d[A] \times (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i} +$$
$$(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}) \bmod (2^n + 1)$$

$$= (d[A(1 + b_0 - 2b_1)]$$
$$+ \sum_{i=1}^{\lfloor n/2 \rfloor} d[A] \times (b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i} +$$
$$(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}) \bmod (2^n + 1)$$

$$= (d[A(1 + b_0 - 2b_1)]$$
$$+ \sum_{i=1}^{\lfloor n/2 \rfloor} (d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i})] + 1)) \bmod (2^n + 1)$$

$$= (d[A(1 + b_0 - 2b_1)]$$
$$+ \sum_{i=1}^{\lfloor n/2 \rfloor} (d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i})] + \lfloor n/2 \rfloor) \bmod (2^n + 1)$$

…….. (10)

Now in IDEA, the operand size(n) is of 16 bit wide. So n is even and equation (10) can be written as

$$d[AB] = (d[A(b_{n-1} + b_n - 2b_{n+1})2^n] +$$
$$d[A(1 + b_0 - 2b_1)] +$$
$$\sum_{i=1}^{\frac{n}{2}-1} d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}] + n/2) \bmod (2^n + 1)$$

$$d[AB] = (d[-A(b_{n-1} + b_n - 2b_{n+1})] + d[A(1 + b_0 - 2b_1)] +$$
$$\sum_{i=1}^{\frac{n}{2}-1} d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}] + 1 + n/2 - 1) \bmod (2^n + 1)$$

$$= (d[-A(b_{n-1} + b_n - 2b_{n+1}) + A(1 + b_0 - 2b_1)] +$$
$$\sum_{i=1}^{\frac{n}{2}-1} d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}] + n/2 - 1) \bmod (2^n + 1)$$

$$= (d[A(1 + b_0 - 2b_1 - b_{n-1})] + n/2 - 1 +$$
$$\sum_{i=1}^{\frac{n}{2}-1} d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}]) \bmod (2^n + 1)$$

$$= (d[A(\bar{b}_{n-1} + b_0 - 2b_1)] + n/2 - 1 +$$
$$\sum_{i=1}^{\frac{n}{2}-1} d[A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}]) \bmod (2^n + 1)$$

…….. (11)

As this algorithm follows radix 4 recoding scheme, the value of the sub-expression ($b'_{n-1} + b_0 - 2b_1$) and ($b_{2i-1} + b_{2i} - 2b_{2i+1}$) are elements of the set {-2, -1, 0, +1, +2}.When the value is zero, a separate correction term is introduced. Now according to Chen and Yao [1], equation

(11) can be represented in terms of a summation expression containing partial products and correction terms, which can be written as

$$d[AB] = (\sum_{i=0}^{\frac{n}{2}-1} partialproduct_i - \sum_{i=0}^{\frac{n}{2}-1} c_i + \frac{n}{2} - 1) \bmod (2^n + 1)$$

…….. (12)

This can be further written as

$$d[AB] = (\sum_{i=0}^{K-1} partialproduct_i - \sum_{i=0}^{K-1} c_i + K - 1) \bmod (2^n + 1)$$

$$= (\sum_{i=0}^{K-1} partialproduct_i - C + K - 1) \bmod (2^n + 1)$$

$$= (\sum_{i=0}^{K-1} partialproduct_i + \overline{C} + 2 + K - 1) \bmod (2^n + 1)$$

$$= (\sum_{i=0}^{K-1} partialproduct_i + \overline{C} + d[1] + K + 1) \bmod (2^n + 1)$$

…….. (13)

Where

$$K = \frac{n}{2}$$

$$C = \sum_{i=0}^{K-1} c_i$$

The entire multiplication algorithm is stated in brief steps as follows:-

- Convert the multiplicand (A) and the multiplier (B) to their corresponding diminished-1 representation. If either of the multiplicand or the multiplier is zero, first the number is represented as 2^n then represented in diminished-1 form. So d[0] becomes $2^n - 1$.
- An extra bit is appended with the LSB of the multiplier such that the appended bit is complement of the MSB of the multiplier. The resultant number becomes an n+1 bit wide number as shown below.

b_{n-1}	b_{n-2}	b_{n-3}	……	…..	….	b_1	b_0	b'_{n-1}

- From the (n+1) bit number, the codes are formed as per radix 4 encoding algorithm, which gives n/2 partial products. The partial products are formed as per the algorithm proposed by Chen and Yao [2], which is given in Table 1. The corresponding correction terms are also calculated having zero values against non zero codes and non zero values for zero codes.

Table 1
Partial products and respective correction terms.

b_{2i+1}	b_{2i}	b_{2i-1}	code	pp_i	c_i
0	0	0	0	$2^{2i} - 1$	2^{2i}
0	0	1	1	$d[2^{2i} A]$	0
0	1	0	1	$d[2^{2i} A]$	0
0	1	1	2	$d[2^{2i+1} A]$	0
1	0	0	-2	$d[-2^{2i+1} A]$	0
1	0	1	-1	$d[-2^{2i} A]$	0
1	1	0	-1	$d[-2^{2i} A]$	0
1	1	1	0	$2^{2i} - 1$	2^{2i}

Here the range of i is $0 < i < K$. When i = 0 then the corresponding triplet is $(b_1 b_0 \overline{b}_{n-1})$

- The K partial products, C' and d[1] are added using inverted EAC CSA tree [3]. So the K+1 term in (13) are naturally added. The final two operands formed by the adder tree is added using diminished-1 adder to form the product (d[AB]). It is to be noted that no separate modulo reduction step is need in this algorithm.

V. PROPOSED MULTIPLIER ARCHITECTURE

The modulo multiplier proposed in this paper consists of a partial product generator, a correction term generator, an inverted end around carry (EAC) adder tree for reducing K+2 operands to a final sum and carry vector and a final diminished-1 modulo $(2^n + 1)$ adder.

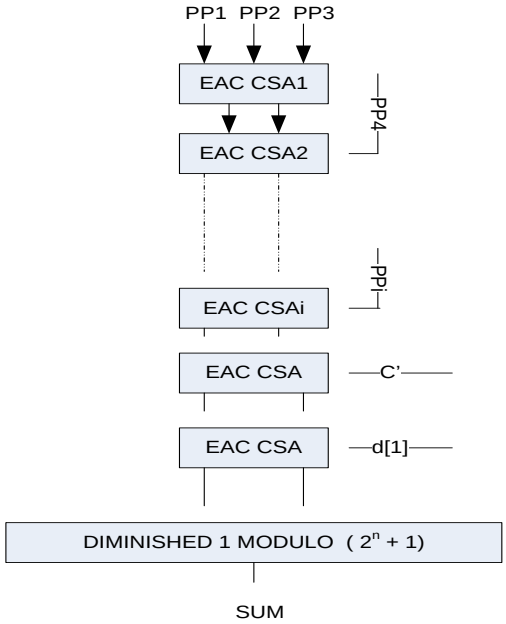

Figure 2: Proposed Multiplier Architecture

An extra module is necessary for zero checking and conversion of diminished-1 to weighted form. The block diagram of the proposed multiplier is given in Figure .2.

The partial product generator consists of two sub-blocks, (i) Booth Encoder (BE) block for generating the codes from the triplets and (ii) Booth Selector (BS) block for generating the partial products. The BE block takes three consecutive bits of the multiplier as inputs and provides the

corresponding code as per radix 4 booth recoding scheme. The BS block takes the code and the multiplicand as the input and produces the corresponding partial product. The correction term generator generates a vector C' which is of the form

$$(\ldots..x'_{i+1}1x'_i1\ldots\ldots1x'_11x'_0)_2$$

The EAC adder tree consists of Carry Save Adders (CSAs) which is constructed using Full Adders (FA).
The modulo $(2^n + 1)$ multiplier proposed in this paper follows a similar structure as that of Chen and Yao [1]. The difference is that, as in IDEA, the operand zero is treated as 2^n, the zero checking using n^{th} bit of the multiplicand and the multiplier is not required. When a zero value is encountered, before converting it to diminished-1 form i.e (-1) mod $(2^n + 1)$, it is treated as 2^n i.e. $d[0]=d[2^n]= 2^n -1$.

VI. Performance and Observation

This new modulo multiplication technique is quite efficient in the sense that it reduces the number of partial products and there is no separate modulo reduction phase. The only overhead is the delay in producing the partial products and correction term and the delay in the adder tree. The delay in adder tree depends on the depth of the tree which is fixed for a given n. So the delay in the partial product generator is also fixed for a given n.
The performance estimation is done qualitatively using unit gate model. According to this model, all 2 input monotonic gates count is considered as one gate delay and all 2 input XOR/XNOR gate count as 2 gate delays, the entire delay van be calculated as:

Time delay for a CSA is the time delay of the Full Adder * depth of the (K + 2) operand adder tree. As the last operand is zero (d[1]), the last full adder can be replaced by a half adder.

CSA (time) = FA (time) * h (K +1) + HA (time).

Where h(X) is the height of the adder tree of X number of operands.
Now according to Efstathiou [11, 15], the delay in the final diminished-1 modulo adder is given by $2\lceil \log n \rceil + 3$
Here the value of n is (4, 6, 10, 16, 24, 36, etc)

So the total delay is

PPG (time) + CSA (time) + $2\lceil \log n \rceil + 3$

=Constant value + FA (time) * h (K +1) + HA (time) + $2\lceil \log n \rceil + 3$

The delay is a Full Adder as per the unit gate model is 4. A tabular comparison of delay of existing schemes with other schemes are as follows:

Table 2: Comparison with other diminished-1 schemes

Multiplier	Delay
Proposed multiplier	Constant(C_0) + 4* h(n/2 + 1) + $2\log_2 n$
Zimmermann [14]	C_2 + 4* h(n/2 + 1) + $2\log_2 n$
Souse,chaves[10]	C_1 + 4* h(n/2 + 1) + $2\log_2 n$
Efstathiou[11]	C_4 + h(n + 3) + $2\log_2 n$

In this context, the values of C_0, C_1, C_2, and C_4 are constant delays of the partial product generator (PPG), in various schemes. However the delay in PPG depends on the number of partial products and the depth of the Adder Tree used.
The needful results are given in table 3. The design is synthesized using VHDL code, and then it is realized in Virtex II Pro –XC2VP30 -7.

Table 3: Synthesis Report

Device used	Virtex II Pro – XC2VP30
Language	VHDL
Input size (n=16)	Delay 3.293 ns
Total Gate Delay	4.883 ns
System Clock Frequency	100 MHz

Although the comparisons are made in terms of time delay, the area requirements are now not taken into account. Moreover, the conversion delay from weighted form to diminished-1 form is also not considered. The proposed design is incorporated in IDEA cryptosystem in order to speed up the multiplication process.

VII. Conclusion

This newly proposed multiplier is efficient when it is used many times in a single algorithm. As is uses a pure radix 4 booth recoding, the computation is faster. To make the design more efficient, a pipelining approach may be incorporated to reduce time delay but it will increase the area requirements. This new multiplier when used in IDEA, reduces the time delay and increases the data throughput rate. In all the computations, the conversion cost in hardware and delay from weighted normal form to diminished-1 form has not been considered for simplicity. The delay may vary according to the platform i.e. for ASICs the delay will be less compared to FPGAs.

Acknowledgment

The authors are thankful for the support provided by VLSI Laboratory developed as part of Special Manpower development programme in VLSI by The Ministry of Communication and Information Technology Govt. of India at ECE Department, NIT Rourkela.

REFERENCES

[1] J.W. Chen and R.H. Yao. Efficient modulo $2^n + 1$ multipliers for diminished-1 representation. Circuits, Devices Systems, IET, 4(4):291 -300, jul. 2010.

[2] L. Leibowitz. A simplified binary arithmetic for the fermat number transform, Acoustics, Speech and Signal Processing, IEEE Transactions on, 24(5):356 - 359,oct. 1976.

[3] H. T. Vergos, D. Bakalis, and C. Efstathiou. Fast modulo 2^n+1 multi-operand adders and residue generators. Integr. VLSI J., 43(1):42- 48, 2010.

[4] X.Lai and J.L Massey "A Proposal for a New Block Encryption Standard," in advances in Cryptology – EUROCRYPT 90,Berlia,Germany: Springer Verlag pp. 389-404, 1990.

[5] Tsoi Kuen Hung,Leong," Cryptographic Primitives on Reconfigurable Platforms", PhD thesis,The Chinese University of Hong Kong,2002.

[6] R.Zimmermann,A.Curiger,H.Bonnenberg,H.Kaeslin,N.Felher,W.Fitchner,"A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm", IEEE Journal of Solid State Circuit,vol.29,110.3,pp.303-307,March 1994.

[7] Thaduri,M.,Yoo,S. and Gaede,R, " An Efficient Implementation of IDEA encryption algorithm using VHDL", ©2004 Elsevier.

[8] P. Kitsos *, N. Sklavos, M.D. Galanis, O. Koufopavlou , "64 Bit Block ciphers: Hardware Implementations and Comparison analysis",593-604,3rd November,2004,Elsevier.

[9] Stefan Wolter,Hogler Matz,Andreas Schubert and Ruiner Laur, " On the VLSI Implementation of International Data Encryption Algorithm.", © IEEE 1995.

[10] Sousa L , Chaves R : ' A universal architecture for designing efficient modulo $2^n + 1$ multipliers' , IEEE Trans. Circuits Syst. I., 2005, 52, (6), pp. 1166-1178.

[11] Efstathiou C, Vergos H.T. ,Dimitrakopoulos G., Nikolos D, : 'Efficient diminished-1 modulo $2^n + 1$ multipliers ', IEEE Trans. Comput., 2005, 54,(4), pp. 491-496.

[12] Curiger,Bonnenberg and Kaeslin,H., "Regular VLSI Architecture for Multiplication Modulo ($2^n + 1$).",IEEE Journal of Solid State Circuits,vol.27,NO. 7,July 1991,pp 990-994.

[13] Bruce Schneier,"Applied Cryptography", 2nd Edition,Wiley publications.

[14] Zimmermann, R.; , "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on* , vol., no., pp.158-167, 1999 doi: 10.1109/ARITH.1999.762841

[15] Efstathiou, C.; Voyiatzis, I.; , "Handling zero in diminished-1 modulo $2^n + 1$ subtraction," *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on* , vol., no., pp.1-6, 6-8 Nov. 2009 doi: 10.1109/ICSCS.2009.5414182.

[16] Helger Lipmaa. Idea: A cipher for multimedia architectures, In Selected Areas in Cryptography 98, pages 248{263. Springer-Verlag, 1998.

[17] H. Bonnenberg, Andreas Curiger, Norbert Felber, Hubert Kaeslin, and Xuejia Lai. Vlsi implementation of a new block cipher. In Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors, ICCD '91, pages 510–513, Washington, DC, USA, 1991. IEEE Computer Society.

[18] O. Mencer, M. Morf, and M.J. Flynn. Hardware software tri-design of encryption for mobile communication units. In Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, volume 5, pages 3045 –3048 vol.5, May 1998.

Sourav Mukherjee received the B.Tech. degree in Computer Science and Engineering from West Bengal University of Technology, Kolkata, India in 2008. He is currently pursuing the M.Tech. in Computer Science from National Institute of Technology, Rourkela, India. His current research interests include FPGA Implementation of cryptosystem and Information security

Bibhudatta Sahoo received the M.Sc. Engineering in Computer Science from National Institute of Technology Rourkela, INDIA, in 1999. He is currently an assistant professor in the Department of Computer Sc. & Engineering, NIT Rourkela, India. His interest include Parallel & Distributed Systems, Networking, Computational Machines, System Software, High performance Computing, VLSI algorithms He is a member of the IEEE Computer Society & ACM.