LOW LATENCY DIGIT-SERIAL IN-CIRCUIT CONFIGURABLE MONTGOMERY MULTIPLIER IN GF(2^M) FOR SPECIAL CLASSES OF POLYNOMIAL

[1]Sabir Akhtar Mallick,[2]Somsubhra Talapatra
[1]Assistant Professor, [2]Assistant Professor
[1] Electronics and Commucation Engineering, Dream Institute of Technology,West Bengal, *India*
[2] *Electronics and Commucation Engineering, Aliah University, West Bengal, India*

ABSTRACT
The In-Circuit Configurable of a Montgomery Multiplication based on Toeplitz matrix-vector representation is proposed. The proposed architecture is design based on to programmable the Montgomery Multiplication. So, we don't waste the hardware which does not required for that design. In our proposed architecture if we increase the element of any digit size (m) the parameters of architecture is not increase but the throughput increases. This is the advantage in our design.

KEYWORDS;- *In-Circuit Configurable,Montgomery Multiplication, Toeplitz Matrix-Vector*

1. INTRODUCTION

Due to rapid growth in secure data communications and internet technology, the public key cryptosystems are extensively used for authentication, data integrity, and data security. Cryptosystems, such as RSA [1-4] and Elliptic Curve Cryptography (ECC) [3], and communication devices employing error control code [2] are mainly based on prime GF(p) [1, 8], GF(p^m) [6, 7] and GF(2^m) [5], where p is a positive prime and m is the positive integer, the arithmetic operations in GF(2^m) can be easily realized in hardware. Also, the electronics information are stored and transmitted in binary format. Therefore, for low cost applications, such as smart cards and mobile personal gadgets, and in transmitters and receivers operating in the physical layer of OSI architecture, the GF(2^m) is the preferred finite field. The popular public-key cryptosystem: ECC [3] require extensive Galois field arithmetic operations e.g. addition, multiplication, inversion and exponentiation. The addition of two m-bit elements in GF(2^m) requires m bitwise EXOR operations. The multiplication of the same operands requires partial products generation using m AND operations for each partial product and then addition of partial products. In the above two operations there is no carry propagation.

The area and time complexities of VLSI realization of finite field multiplier depend on four aspects- (a) the basis representation of the field elements, (b) the generator polynomial, (c) the algorithm of multiplication, and (d) the architecture of the multiplier. The elements of GF(2^m) can be represented either in polynomial basis (PB), or in normal basis(NB) or in dual basis. The last two require basis conversion between PB and them as pre and post processing steps. The PB representation naturally occurs during construction of an extension field. A good multiplication algorithm over Galois fields needs suitable presentation of a polynomial for construction of the field. The generator polynomial is required during reduction of the partial products by way of appropriate shifting and then adding to a partial product. GF(2^m) multipliers based on some popular polynomials such as AOP and trinomials have a low circuit complexity [22].

In [22], a formulation, exploiting the simplicity of the formulation for binomials, was given for trinomials. Of the algorithms available for large number multiplications, such as direct multiplication algorithm(DMA), Montgomery multiplication algorithm (MMA), Karatsuba-Ofman algorithm(KOA), and FFT based multiplication, only first two algorithms are useful for modular multiplication in GF(2^m) for moderate values of m [28]. All three algorithms, additionally, require to apply reduction while applied in the context of finite field. The FFT based approach is applicable to only GF(p) and GF(p^m) [7,28].

MM algorithm introduce by P.L. Montgomery [17] speeds up the modular multiplication and squaring required for exponentiation operation in the public-key cryptosystems. MM algorithm computes the Montgomery product $C=AB\,R^{-1}$ mod G where {A, B < G} and R such that gcd(G, R)=1. In traditional GF(2) multiplication algorithm, every partial product needs to be reduced by the generator polynomial G(x) before being added. This increases the complexity of the architecture. The reduction process is not necessary for the Montgomery multiplication algorithm over finite field [18], with the suitable choice of Montgomery polynomial R(x). This improves both the software and VLSI architectures. In Montgomery multiplication algorithm, each operand is transformed by the multiplying R(x). The output is the required product polynomial multiplied by R(x). So the output has to be divided by R(x). With choice of $R(x)=x^k$, where k is a positive integer, this division becomes k-bit shift towards LSB.

The advantage of MMA is that it does not require post-multiplication reduction steps of direct multiplication, which involves time consuming trial divisions. The formulation of MMA in GF(2^m) [20] requires to calculate only x^{-1}, which equals to the generator polynomial G(x) shifted by a bit towards the LSB, in order to reduce the partial products. This, along with the choice of Montgomery polynomial $R(x)=x^m$, yields efficient LSB-first multiplication algorithms for binary and prime fields [18-19].

For high speed vlsi implementation, the preferred architecture for polynomial basis (PB) multiplier is systolic array architecture in which, basic cell is repeated in an array and signals flow unilaterally between neighbours. Architecture presented in [16, 18] both use identical cells and require a latency of 3m clock cycles. For fully pipelined operation, these architectures require large area and latency. PB systolic array multipliers over GF(2^m) can be classified into four categories: bit-serial, bit-parallel,hybride and digit-serial. In digit serial architecture, an m-bit word broken into $N=\lceil m/L \rceil$ digits. In every clock cycle, 'L' bits of multiplier and multiplicand are processed to produce one m-bit product every N cycles.

The architecture of multiplier in GF(2^m) can be semi-systolic [22], [25], [33], systolic [11]-[16], [26], and super-systolic [27]. Depending on the mathematical formulation of product coefficients, systolic architecture may be either of MSB-first or of LSB-

first types [5, 7], with the later rendering less complex architectures.

The reduction of area complexity generally results in increase of timing complexities (clock period and/or cycle latency). Bit-serial multipliers [5] have the lowest area complexity of (O(l)) but worst timing complexity of ($O(m^2)$). If one operand is kept constant over many multiplications, it can be fed in parallel. This technique improves timing complexity of (O(m)) of serial/parallel multipliers [35] at the cost of area complexity (O(m)). Bit-parallel multipliers [10, 12, 23, 33] have the worst area complexity of ($O(m^2)$) and the clock period of O(l), if bit-level pipelined, otherwise O(l). For moderate latency and moderate area multipliers, the digit-serial architecture s are proposed [11-16, 24-26]. If L is the digit size and N_{PE} is the number of Processing Elements (PEs) employed, which normally are pipelined, and $N = \lceil m/L \rceil$, then area and timing complexities are $O(N_{PE}L^2)$ and O(N), respectively.

In [23], Montgomery multiplication is represented by two Henkel matrix-vector multiplications when $R=x^k$ where $GF(2^m)$ is generated from irreducible trinomials of form (x^m+x^k+1). A bit-parallel systolic Montgomery multiplication architecture over $GF(2^m)$ based on irreducible AOPs and trinomials has been proposed in [22]. The multiplier has a latency of (m+1) clock cycles. A scalable and systolic MM architecture has been developed in [24] which uses a fixed (L×L) bit-parallel Henkel matrix multiplier iteratively in order to achieve {m×m} in [25]. Its latency is {L+N(N+1)} clock cycles.

Contribution: The digit-serial architecture is the most suitable to support DMA and MMA over $GF(2^m)$ with programmable value of m for realization in ASIC. The digit-serial multipliers presented in [11-16,24-26] are parameterized and scalable and are suitable for FPGA, but not for ASIC. In this paper, we propose an In-Circuit Configurable Montgomery Multiplier (ICCMM) in $GF(2^m)$ generated by AOP and trinomials for any m, subjected to some maximum value determined by the memory resource of the ASIC. Moreover, the circuit is parameterized and scalable. The latency can be scaled down to low value without sacrificing too much area. The architecture consists of a core systolic digit-serial MM and four memory systems, together having 4SL bits, where S is the maximum supported value of N. The ICCMM has cycle latency of $\{N+NN_C+N_{PE}+2\}$ and clock period of $O(\lceil \log_2 L \rceil)$, where N_{PE} is the number of pipelined PEs.

2. MATHEMATICAL MODEL

2.1. PRELIMINARIES

Depending on the mathematical formulation of product coefficients, systolic architecture may be either of MSB-first or of LSB-first types with the later rending less complex architecture. The i-th partial product (P_i) in MSB-first DMA and the P_i in LSB-first DMA, for i=1,2,…m, can be generated and reduced as shown following equations respectively.

$T_i(x)=xP_{i-1}(x)+A(x)b_i$ (1a)

$P_i(x)=T_i(x) \bmod G(x)=t_{i,m}G(x)+T_i(x)$ (1b)

$P_i(x)=P_{i-1}(x)+A_{i-1}(x)b_i$ (2a)

$A(x)=[A_{i-1}(x)x] \bmod G(x)=A_{i-1}(x)x+a_{i-1,m-1}G(x)$ (2b)

The j-th coefficient of P_i and A_i has dependency on signals from the less significant terms, such as $P_{i-1,j-1}$ and $a_{i-1,j-1}$ and the most significant terms, such as t_{i-1} and $a_{i-1,m-1}$. This is a problem for LSB-first digit-serial or bit-serial multipliers, where the least significant bits are processed first. Moreover, in MSB-first formulation, the modular reduction in (1b) is dependent on the partial product generation in (1a) in the same iteration. The delay of the critical path of the resultant multiplier is also larger than that of the LSB-first DMA multiplier. Therefore, transformations at the interconnect level are required for both MSB and LSB first multipliers.

Montgomery Multiplication in $GF(2^m)$

Montgomery multiplication (MM) was propose by P. L. Montgomery [17] for integer multiplication and was later modified in [18-19] for application to the finite field. The bit level MM in $GF(2^m)$ is simpler than word level MM in GF(p) with p being prime [18]. The bit-level MMA and word level MMA have been discussed briefly as below.

Algorithm 1: Bit level MM in $GF(2^m)$

Input: $A(x) \equiv \{ a_{m-1}, a_{m-2}, \ldots, a_0 \}$, $B(x) \equiv \{ b_{m-1}, b_{m-2}, \ldots, b_0 \}$, $G(x) \equiv \{ 1, g_{m-1}, g_{m-2}, \ldots, 1 \}$,

$R(x) = x^m$

Output: A(x) B(x) mod G(x)

Init: $P^{(0)} = 0$

1. for i in 0 to m-1

1.1. $T(x) = P^{(i)}(x) + A(x)\, b_i$

1.2. for j in 0 to m-1 / * $P_{i-1} = (T(x) + t_0\, G(x)) >> 1$ * /

1.2.1. $P_j^{(i+1)} = t_j^{(i)} + t_0\, g_{j+1}$ / * $t_m^{(i)} = 0$ * /

1.2.2. end for

1.3. end for

2. return $P(x) = P^m(x)$

For special G(x), such as the AOP of degree m-1 for $GF(2^{m-1})$ and trinomial $G(x) = x^m + x^k + 1$for $GF(2^m)$, simplified mathematical formulation exists in literature [22] , [26] . The AOP, $G(x) = x^{m-1} + x^{m-2} + \ldots + 1$, can be transformed in to binomial ($x^m + 1$) by multiplying with (x+1). Therefore, any A(x) in $GF(2^{m-1})$, can have isomorphic representation $A(x) = a_{m-1}\, x^{m-1} + a_{m-2}\, x^{m-2} + \ldots + a_0$ in the polynomial ring GF(2) [x]/($x^m + 1$) . Discussions on similar transformation on general irreducible polynomials over GF(2) are given in [21]. For bit-level MMA modulo ($x^m + 1$) and modulo trinomial ($x^m + x^k + 1$) , the step 1.2.1 in algorithm 1 should be replaced by the following steps.

1.2.1.1. if (j= m-1) then

1.2.1.2. $P_j^{(i+1)} = t_0$
1.2.1.3. else if (j=k-1) then /* Not relevant for binomial */
1.2.1.4. $P_j^{(i+1)} = t_k^{(i)} + t_0\, g_k$
1.2.1.5. else
1.2.1.6. $P_j^{(i+1)} = t_{j+1}^{(i)}$
This is simplify the iterative bit-level MMA of Algorithm 1 for AOPs , trinomials and low weight polynomials .

A.Toeplitz Matrix Multiplication (TMM) based on MM algorithm for AOP in GF(2^{m-1})

If $A(x)=a_{m-1}x^{m-1}+ \ldots +a_2x^2+a_1x+a_0$ and $B(x)=b_{m-1}x^{m-1}+ \ldots +b_2x^2+b_1x+b_0$ are two elements in GF(2)[x]/(x^m+1), and the Montgomery reduction polynomial $R(x)=x^k$, then the Montgomery multiplication is given below.
$A(x)\, B(x)\, x^{-k} \bmod (x^m+1) = \sum_{i=0}^{m-1} pi\, x^i$
Where $p_i = \sum_{j=0}^{m-1} \left(a_j\, a_{i\oplus inv(j\oplus k)}\right)$
Each product coefficient can be expressed as the product between two vectors. If $\overline{A}=[a_0,\ldots,a_{m-2},a_{m-1}]^T$ and $\overline{W_i}$ is a row vector whose j-th element is $b_{i\oplus inv(j\oplus k)}$, then from eqn (6), p_i can be written as $\{p_i=\overline{W_i}.\overline{A}\}$, where product is done over vector space V_m in GF(2^m) . Therefore, P=[p_i]=[W_i] . $\overline{A}$= $W.\overline{A}$, where W is defined as a column matrix $W=\left[\overline{W_0}\cdots\overline{W_{m-2}}\,\overline{W_{m-1}}\right]^T$, and is given in eqn. (7).

$$W=\begin{bmatrix} b_k & b_{k-1} & \cdots & \cdots & b_0 & b_{m-1} & \cdots & \cdots & b_{k+1} \\ b_{k+1} & b_k & \cdots & \cdots & b_1 & b_0 & \cdots & \cdots & b_{k+2} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ b_{m-2} & b_{m-3} & \cdots & \cdots & b_{k-1} & b_{k-2} & \cdots & \cdots & b_{m-1} \\ b_{m-1} & b_{m-2} & \cdots & \cdots & b_k & b_{k-1} & \cdots & \cdots & b_0 \\ b_0 & b_{m-1} & \cdots & \cdots & b_{k+1} & b_k & \cdots & \cdots & b_1 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ b_{k-2} & b_{k-3} & \cdots & \cdots & b_{m-2} & b_{m-3} & \cdots & \cdots & b_{k-1} \\ b_{k-1} & b_{k-2} & \cdots & \cdots & b_{m-1} & b_{m-2} & \cdots & \cdots & b_k \end{bmatrix}$$

Therefore, P = [p_i] = [W_i] . $\bar{A}$ = W . $\bar{A}$, where W is defined as a column matrix W=[$\bar{W}_0 \ldots \bar{W}_{m-2}\ \bar{W}_{m-1}]^T$, is given above equation.
$A(x)\, B(x) = t_0 + t_1x + \ldots + t_{k-1}x^{k-1} + \ldots + t_{k+m-1}x^{k+m-1} + \ldots + t_{2m-1}x^{2m-1} = T_1(x)+T_2(x)x^k + T_3(x)x^{m+k}$,Where $T_1(x) = t_0 + t_1x + \ldots + t_{k-1}x^{k-1}$, $T_2(x) = t_k + t_{k+1}x + \ldots + t_{k+m-1}x^{m-1}$ and $T_3(x) = t_{k+m} + t_{k+m+1}x + \ldots + t_{k+2m-1}x^{m-1}$ are part of the product polynomial. If R(x) = x^k , then (x^m +1) and R(x) have no common root. Then Montgomery multiplication of A(x) and B(x) is given below,
$A(x)\, B(x)\, R^{-1}(x) \bmod (x^m+1) = [\, T_1(x)x^{-k} + T_2(x) + T_3(x)x^m \,] \bmod (x^m+1) = T_1(x)x^{m-k} + T_2(x) + T_3(x)$,Where the product is in GF(2)[x]/(x^m+1).

B.Toeplitz Matrix Multiplication (TMM) based on MM algorithm for Trinomials in GF(2^m)

If $G(x) = x^m + x^k + 1$, where 1 < k < m , is a degree-m irreducible trinomial over GF(2) and $R(x) = x^k$, then
$R^{-1}(x) = x^{-k} \bmod (G(x)) = x^{m-k} + 1$.
$A(x)\, B(x)\, R^{-1}(x) \bmod (G(x)) = [\, T_1(x)\, x^{-k} + T_2(x) + T_3(x)\, x^m \,] \bmod (G(x))$
$= [\, T_1(x)\, (x^{m-k}+1) + T_2(x) + T_3(x)\, (x^k+1) \,] \bmod (G(x))$
$A(x)\, B(x)\, R^{-1}(x) \bmod (G(x)) = A(x)\, B(x)\, R^{-1}(x) \bmod (x^m+1) + (T_1(x)+T_3(x)x^k)$
$= W\,.\,\bar{A} + W_k\,.\,\bar{A} = W_{Mk}\,.\,\bar{A}$

$$\begin{bmatrix} p_k \\ p_{k+1} \\ \vdots \\ \vdots \\ p_{m-2} \\ p_{m-1} \\ p_0 \\ p_1 \\ \vdots \\ \vdots \\ p_{k-1} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 & 0 & b_{m-1} & b_{m-2} & \cdots & \cdots & b_{k+1} \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & b_{m-1} & \cdots & \cdots & b_{k+2} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 & b_{m-1} \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ b_0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & b_0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ b_{k-1} & b_{k-2} & \cdots & \cdots & b_0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_{m-k-2} \\ a_{m-k-1} \\ a_{m-k} \\ a_{m-k+1} \\ \vdots \\ \vdots \\ a_{m-1} \end{bmatrix} = W_k\,\overline{A}$$

The elements of the resultant W_{MK} follows the relation, $W_{MK}(i+1,j+1)=W_{MK}(i,j)$, $0 \le i , j \le m-2$, which slightly differs from the eqn. (9). Therefore, W_{MK} is a Toeplitz matrix (TM).

Example 1: A(x) = $\sum_{i=0}^{8} ai\, x^i$ and B(x) = $\sum_{i=0}^{8} bi\, x^i$ are two polynomials over GF(2). Suppose, G(x) =1+x^{-4}+x^9 is an irreducible trinomial over GF(2). $T_1(x)+T_3(x)x^4$ can be computed as:

$$\begin{bmatrix} p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & b_8 & b_7 & b_6 & b_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_8 & b_7 & b_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_8 & b_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_1 & b_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_2 & b_1 & b_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_3 & b_2 & b_1 & b_0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix} = W_4.\overline{A}$$

Now, A(x) B(x) x^{-4} mod(1+x^4+x^9) can be expressed as:

$$\begin{bmatrix} p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} b_8 & b_7 & b_6 & b_5 & b_4 & b_3+b_8 & b_2+b_7 & b_1+b_6 & b_0+b_5 \\ b_0 & b_8 & b_7 & b_6 & b_5 & b_4 & b_3+b_8 & b_2+b_7 & b_1+b_6 \\ b_1 & b_0 & b_8 & b_7 & b_6 & b_5 & b_4 & b_3+b_8 & b_2+b_7 \\ b_2 & b_1 & b_0 & b_8 & b_7 & b_6 & b_5 & b_4 & b_3+b_8 \\ b_3 & b_2 & b_1 & b_0 & b_8 & b_7 & b_6 & b_5 & b_4 \\ b_4+b_0 & b_3 & b_2 & b_1 & b_0 & b_8 & b_7 & b_6 & b_5 \\ b_5+b_1 & b_4+b_0 & b_3 & b_2 & b_1 & b_0 & b_8 & b_7 & b_6 \\ b_6+b_2 & b_5+b_1 & b_4+b_0 & b_3 & b_2 & b_1 & b_0 & b_8 & b_7 \\ b_7+b_3 & b_6+b_2 & b_5+b_1 & b_4+b_0 & b_3 & b_2 & b_1 & b_0 & b_8 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix}$$

2.2. Unified Model AOP and Trinomials

Montgomery Multiplications in polynomial ring generated by degree-*m* binomial, and in $GF(2^m)$, generated by any irreducible trinomial of degree *m*, can be represented by multiplication of a TM, formed by arranging [eqn's (7),(14)] the coefficients of multiplicand, say *B(x)*, with a column matrix representing the multiplier, say *A(x)*. Since, a TM is fully specified by the elements of the first column and the first row, and if the elements of the first row and the first column of the TM are pre-computed, a single TM multiplier module may be used to implement eqn's (5) and (13). In [23-24], similar approach has been used to produce MM in $GF(2^m)$ generated by trinomials. Their pre-computation stage, called Henkel-Matrix-Addition block, is generic but not in-circuit configurable for any value of *k*.

In general, the pre-computation stage should calculate, for any value of *k*, modulo-2 addition of one input, which corresponds to the first column and the first row of *W*[eqn. (7)], and another input, which corresponds to the first column and the first row of W_k[eqn. (14)]. For MM in $GF(2)[x]/(x^m+1)$, the second input should be forced to a sequence of zeroes. This stage can be implemented in parallel, in serial, or in digit-serial form, depending on the input requirement of TM multiplier. Algorithm 2 gives the operation of the pre-computation stage. W_{row} and W_{col} are the row and the column of the matrix *W* , respectively. $W_{k,row}$ and $W_{k,col}$ are the row and column of the matrix W_k , respectively. V_0, V_1, H_0, and H_1 are *L*-bit digit-serial input words to the pre-computation block, where *L* is the digit size. For serial and parallel implementation of TM multiplier, the value of *L* is 1 and *m*, respectively. The index *i* in b_i's (coefficient of x^i in *B(x)*) in the inputs is computed modulo-*m*. In case of binomial, $W_{k,row}=W_{k,col}=[0,..,0]$. The '⊕' in Algorithm 2 stands for bit-wise modulo-2 addition (EXOR) of two words.

Algorithm 2: PRECOMPUTE /*Generic pre-computation of inputs of TM. */
Inputs: W_{row} [0:m-1] = [b_{2k} , b_{2k-1}, . . . , b_{m-1}, b_0, . . . , b_{2k+1}] , // first row of W matrix//
W_{col} [0:m-1] = [b_{2k} , b_{2k+1}, . . . , b_{m-1}, b_0, . . . , b_{2k-1}] , // first column of W matrix// 13
$W_{k,row}$ [0:k] = [0 , . . . , 0] , $W_{k,row}$ [k+1:m-1] = [b_{m-1} , b_{m-2} , . . . , b_{k+1}] , // first row of W_k matrix//
$W_{k,col}$ [0:m-k-1] = [0 , . . . , 0] , $W_{k,row}$ [m-k:m-1] = [b_0 , b_1 , . . . , b_{k-1}] , // first row of W_k matrix//
Outputs: $W_{Mk,row}$ [0:m-1] = $W_{row} \oplus W_{k,row}$
$W_{Mk,col}$ [0:m-1] = $W_{col} \oplus W_{k,col}$
Parameters: Binomial: (x^m + 1) , and Trinomial: (x^m+x^k+1) , and Digit Size (L) = m or 1 or L , when TM is respectively either parallel or serial or digit-serial , and N=⌈m/L⌉.

1.for i in 0 to N-1 do
2. V_0 = W_{row} [iL : (i+1)L-1]
3. V_1 = $W_{k,row}$ [iL : (i+1)L-1]
4. H_0 = W_{col} [iL : (i+1)L-1]
5. H_1 = $W_{k,col}$ [iL : (i+1)L-1]
6. $W_{Mk,row}$ [iL : (i+1)L-1]= $V_0 \oplus V_1$

7. $W_{Mk,col}$ [iL : (i+1)L-1]= $H_0 \oplus H_1$
8. return ($W_{Mk,row}$, $W_{Mk,col}$) /* Serial outputs to TM */9.end for

3. PROPOSED ARCHITECTURE

Formulation for Proposed Architecture

The product matrix P in (13) can be written in terms of smaller $L \times L$ TMs as in (15), where $W_{T,L}(i,j)= W_{T,m}[iL:(i+1)L-1][jL:(j+1)L-1]_{LxL}$, and $W_{T,m}$ equals to $W_{M,k}$.

$$P=W_{T,m}\,A_m=\begin{bmatrix} W(0,0) & W(0,1) & \dots & W(0,N-1) \\ W(1,0) & W(1,1) & \dots & W(1,N-1) \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ W(N-1,0) & W(N-1,1) & \dots & W(N-1,N-1) \end{bmatrix}\begin{bmatrix} A(0) \\ A(1) \\ . \\ . \\ . \\ A(N-1) \end{bmatrix}$$

Where A(i)=A[iL:iL+L-1]. The product matrix P=[P(i)]$_{N*1}$ is expressed as a column matrix, i-th element (digit) of which consists of the array P(i)=[$p_{k+iL}, p_{k+iL+1}, .., p_{k+iL+L-}$].

$W_{T,L}(i,j)= W_{T,L}(i+1,j+1)$ (16a),

$W_{T,L}(i+1,j+1)[0][0:L-1]$ (the 1st row of $W_{T,L}(i+1,j+1)$)=$W_{T,L}(i,j)[0][0:L-1]$ (the 1st row of $W_{T,L}(i,j)$) (16b),

$W_{T,L}(i+1,j+1)[0:L-1][0]$ (the 1st column of $W_{T,L}(i+1,j+1)$)=$W_{T,L}(i,j)[0:L-1][0]$ (the 1st column) (16c),
$W_{T,L}(i,j+1)[L-1:1][0] = W_{T,L}(i,j)[0][1:L-1]$ (ie., bit-reversed relation) (16d), and $W_{T,L}(i+1,j)[0][L-1:1]$ $= W_{T,L}(i,j)[1:L-1][0]$ (ie., bit-reversed relation) (16e).

From eqn. (15), i-th product digit can be calculated as the sum of series of L-dimensional TM multiplications, as shown in (17).

$$P_L(i) = \sum_{j=0}^{N-1} W_{T,L}(i,j) \times A_L(j) \quad (17)$$

For a given number of PEs (N_{PE}) in an ASIC, N_{PE} may be greater than, equal to, or less than N in (17). In such cases, (17) is broken down to series of smaller sums as in (18), where $N_c = \left\lceil N/N_{PE} \right\rceil$.

$$P_L(i,c) = \sum_{j=0}^{N_{PE}-1} W_{T,L}(i, j+cN_{PE}) \times A_L(j+cN_{PE}) \quad (18a)$$

$$P_L(i) = \sum_{j=0}^{N_C-1} P_L(i,c) \quad (18b)$$

The eqn (18b) can be computed using Algorithm 3.

Algorithm 3:

Inputs: $W_{T,m}[0][0:m-1]$, $W_{T,m}[0:m-1][0]$ and A for $0 \le i,j<L$. Outputs: $P=[P_L(i) \mid 0 \le i < N]_{N_{PE}N_C \times 1}$ of (18b).

Parameters: N, N_{PE} and N_C

Initialize: $W_{T,L}(0,v)= (v<N)?\ W_{T,m}[0][jL:jL+L-1] : [0]_{L \times L}$ for $0 \le v< N_C N_{PE}$, and $W_{T,L}(u,0)=$ $W_{T,m}[uL:uL+L-1][0]$ for $0 \le u<N$, and

$[A_V(v)]_{N_C N_{PE} \times 1} = (v<N)?\ A(v) : 0$ for $0 \le v< N_C N_{PE}$.

1. for c in 0 to N_C-1
1.1. for i in 0 to N-1
1.1.1. if $c==0$ then
1.1.1.1. $P_L(i,0)=[0..0]$
1.1.1.2. end if
1.1.2. $P_L(i,c,0)=[0..0]$
1.1.3. for j in 0 to N_{PE}-1
1.1.3.1. $P_L(i,c,j+1)=P_L(i,c,j)+W_{T,L}(i,j+cN_{PE}) \times A_V(j+cN_{PE})$
1.1.3.2. derive $W_{T,L}(i,j+1)$ using (16d) and (16e)
1.1.3.3. end for
1.1.4. $P_L(i,c+1)= P_L(i,c)+ P_L(i,c,N_{PE})$
1.1.5. derive $W_{T,L}(i+1,0)$ using (16)
1.1.6. end for
1.2. end for
2. return $P=[P_L(i) \mid 0 \le i<N_C N_{PE}]_{N_C N_{PE} \times 1}=[P_L(i,N_{PE}) \mid 0 \le i< N_C N_{PE}]$

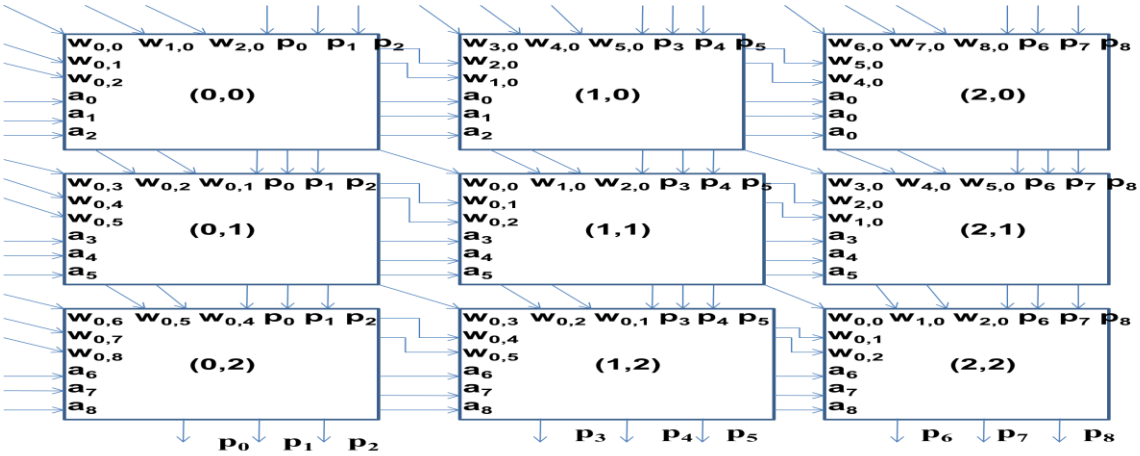


Fig. 1: SFG of the loop starting from line 1.1 with m=9, L=3

4. MEMORY SYSTEM

Proposed Architecture

The memory systems for M_A and M_H are identical and are represented in Fig. 4(a). It consists of a single port $S\times L$-bit RAM and an internal address generator elaborated in Fig. 4(b). For data loading, the address generator is initiated by the "reload" =1 signal to "value"=0h. Before execution, M_V and M_H require to initialize address to 0h. Additionally, M_H requires to reset the address to 0h (reload=1, value=0) when $i=cN_{PE}$ for each c [Algorithm 3]. Never during the execution, it is required to read simultaneously either from M_V or from M_A. Therefore, each of them has a single port to read and write with L-bits of data buses. The words are stored in little-endian mode starting from address location zero onwards.

For $N_C>c\geq 1$ in Algorithm 1, the memory system of M_V are required to be read simultaneously. Two bursts of reads, start at neighbouring locations, ie. cN_{PE} and $cN_{PE}-1$, respectively. Also, the burst lengths are different depending on value of N_{PE} and N_C and iteration value of c. Also, the read directions are opposite of each other. This requires that, M_V consists of two $S_P\times L$-bit ($S_P=S/2$) *RAMs* which are interleaved with respect to the LSB of the address buses, and two ports of which one dedicated to read only. The first port supports both write and read operations. The ($d-1$)-bit addresses to RAMs are generated in the internal address generator depicted in fig.5(b) . The "ld"(=1) signal of the port1 enables external data to load into the memory. Before data loading, the addresses are initialized to 0h using "cnt_rst"(=1) signal. Also, before start of execution, the up-counter, that generates the address "rdwr_addr1" associated with primary read-write port, is initialized to 0h. Its data outputs through "do1" bus. Every time $i=cN_{PE}-1$ for a value of c, the down counter, that generates the address "rd_addr2" for 2nd read port, is reloaded with the current value of up-counter. However, the read by the 2nd port for ($c+1$)-th iteration is strobed by "rd2"=1. The read for the primary port is strobed by "rd1"=1. The output data bus for the 2nd port is "do2".

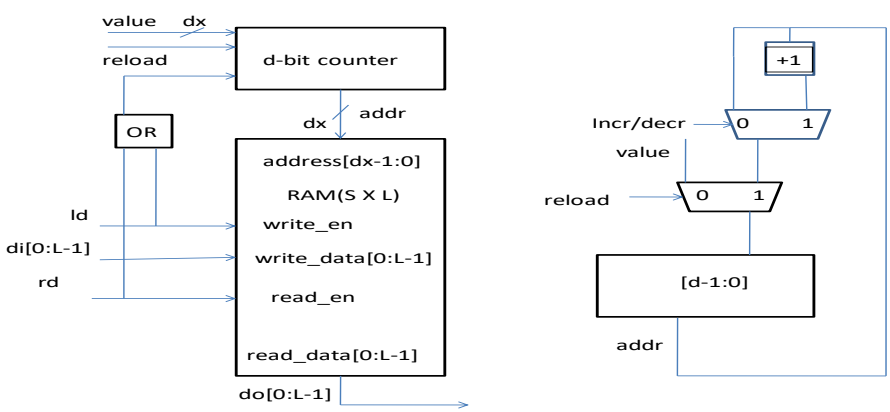


Fig. 4: (a) Memory system for M_H & M_A(S×L bits, max(N)=S), (b) address generator (d=$\lceil \log_2 S \rceil$)

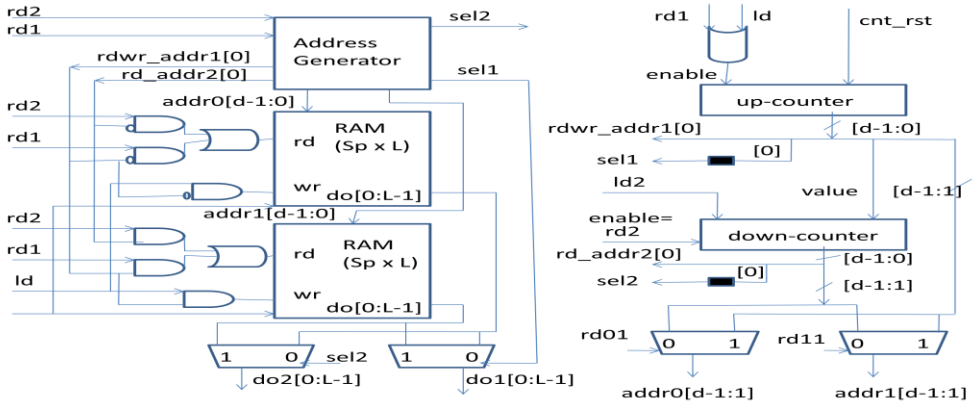


Fig. 5: (a) Memory system for M_V(S_P×L bits, S_P=S), (b) address generator (d=$\lceil \log_2 S \rceil$)

The memory system M_P, depicted in Fig.6(a), for the intermediate product words $P_L(i, cN_{PE})$ is an accumulator, where i^{th} product word $P_L(i)$ is assigned to i^{th} memory location. During execution, computation associated to the i-th location is $P_L(i)=P_L(i)+P_L(i, cN_{PE})$, where $P_L(i, cN_{PE})$ is the output of the core in $(i+cN_{PE})$ cycle. So, i-th location is to be read out in $(i+cN_{PE})$ cycle, and is to be written back in $(i+1+cN_{PE})$ cycle. But, $(i+1)$-th location, also simultaneously, is to be read out for computation of $P_L(i+1)=P_L(i+1)+P_L(i+1,cN_{PE})$. Consequently, M_P consists of two (S_P×L)-bit RAMs, interleaved by the *LSBs* of the addresses of its two ports. The $P_L(i)$'s are read out and written to by using 1st port and the 2nd port, respectively. Also, the 2nd port is used to load external data before execution. This requires a bus-multiplexer to be put before the input data bus, as done in the top level architecture in fig. 8. The internal address generator needs to reset to 0h (by "cnt_rst"=1) before the start of the data loading and before the start of the execution. The address associated with the 1st port and the 2nd port are "*rd_addr*" and "*wr_port*", respectively. During loading of external data, i.e. *"ld"*=1, generates the write enable "wr" for the 2nd port. But, during execution, a read operation, determined by "rd"=1 associated with 1st port, automatically precedes a write operation. In that case, the "wr"(=1) is just delayed version of "rd".

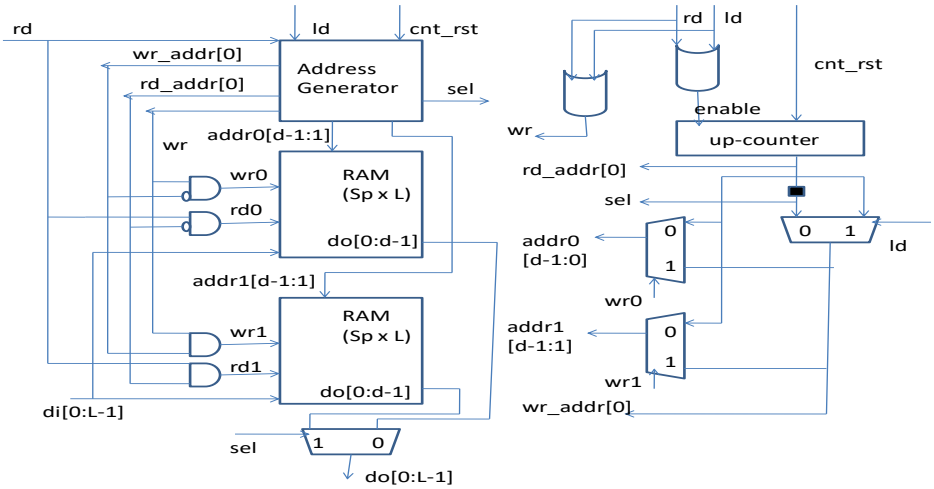


Fig. 6: (a) Memory system for M_P(S_P×L bits,S_P=S), (b) address generator (d=$\lceil \log_2 S \rceil$)

Top Level Architecture

The top level architecture that integrates all the memory systems and the digit-serial core is depicted in Fig.8. The components of the architecture are described in section IV.A and IV.B. To operate the architecture, eleven control signals are required. There are two phases, namely data loading and execution, which are involved in the operation of the architecture. But in this project I compute execution and loading in same time. For m-bit operands with L-bit word memory systems, the loading phase takes N cycles, and the execution phase takes {N(N_{PE}-1)+2N_C}. In execution phase, the control signals are described by the algorithm 4.

Before the start of the loading phase, all the internal address counters are to be reset to zero. Therefore, c7 for M_H, and *c8* for M_V, M_A and M_P need to be pulsed to 1 for a single cycle. After that, for each cycle of data loading, c4 is held to 1 for M_H, M_V, M_A and M_P. The accumulator memory can be loaded with initialization data, which is useful firstly for zeroing out the product values, and secondly for initializing the product value to calculate *MM(A, B)+C*. Therefore, there exists a multiplexer in the input bus to M_P, selected by c61, which also acts as read enable to M_P. During external data loading, c61 are to be held to 0. Example 2 describes an instance of configuration of the top level architecture and related execution cycles.

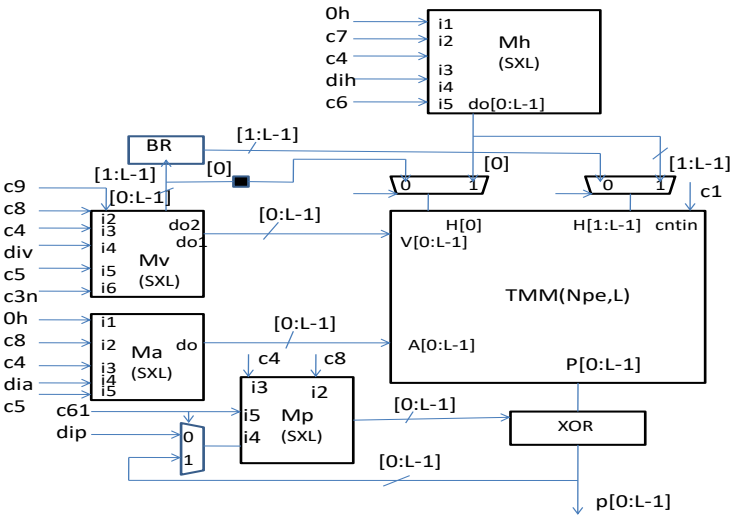


Fig. 8: Integration of memory systems and digit-serial core

Algorithm 4: Behaviour of the control signals during execution phase.

Parameters: m, L, N_{PE}, $N = \lceil m/L \rceil, N_C = \lceil N/N_{PE} \rceil$

Assumption: 1) The cycle when PE_0 executions $W_{T,L}(0,0)$ is denoted as 0-th cycle. Therefore, the execution starts in '-2'-th cycle.

2) The cycle is represented by the variable c_T .

1. for c_T in -2 to $NN_C+N_{PE}-1$

1.1. $c = \lfloor c_T/N \rfloor, i = c_T \bmod N$

1.2. $c = \lfloor (c_T+1)/N \rfloor, i = (c_T+1) \bmod N$

1.3. $c = \lfloor (c_T+2)/N \rfloor, i = (c_T+2) \bmod N$

/* signal : c1 */

1.4. if(c_T<0 or c_T>NN_C-1)then

1.4.1. c1=0

1.4.2. eleif i==0

1.4.3. c1=0

1.4.4. else

1.4.5. c1=1

1.4.6. end if

/* signal : c2 */

1.5. c2=c3

/* signsl : c3 */

1.6. c3=one cycle delayed value of c6

/* signal : c4 */

1.7. c4=0

/* signal : c5 */

1.8. if(c_T<-1 or c_T>NN_C-2)then

1.8.1. c5=dc

1.8.2. else if($0 \le i_{m1} < N_{PE}$)then

1.8.3. c5=1

1.8.4. else

1.8.5. c5=0

1.8.6. end if

/* signal : c9 */

1.9. if(c_T<-1 or c_T>NN_C-2)then

1.9.1. c9=dc
1.9.2. else if(i_{m1}==N_{PE})then
1.9.3. c9=1
1.9.4. else
1.9.5. c9=0
1.9.6. end if
/* signal : c3n */
1.10. c3n=not(c6)
/* signal : c6 */
1.11. if(c_T<-1 or c_T>NN_C-2)then
1.11.1. c6=dc
1.11.2. if(i_{m1}>$c_{m1}N_{PE}$)then
1.11.3. c6=1
1.11.4. else
1.11.5. c6=0
1.11.6. end if
/* signal : c8 */
1.12. if(c_T==-2)then
1.12.1. c8=1
1.12.2. else
1.12.3. c8=0
1.12.4. end if
/* signal : c7 */
1.13. if(c_T<-2 or c_T>NN_C-3)then
1.13.1. c7=dc
1.13.2. if(i_{m2}==0)then
1.13.3. c7=1
1.13.4. else
1.13.5. c7=0
1.13.6. end if
/* signal : c61 */
1.14. if(c_T<N_{PE}-1 or c_T>NN_C+N_{PE}-2)then
1.14.1. c61=0
1.14.2. else
1.14.3. c61=1
1.14.4. end if
/* signal : c81 */
1.15. if(c_T<0 or c_T>NN_C+N_{PE}-2)then
1.15.1. c81=0
1.15.2. elseif(i_{NPEm2}=0)then
1.15.3. c81=1
1.15.4. else
1.15.5. c81=0
1.15.6. end if

Example2: Consider the example 1 for implementation with N_{PE}=2. In this case $N=m/L$=3 and $N_C=N/N_{PE}$=2. The solid thick line in the Fig. 9 shows the order of execution of PEs of the core. The detailed cycle diagram is given in table 1.

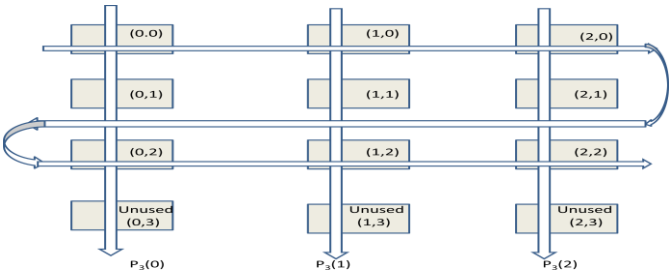
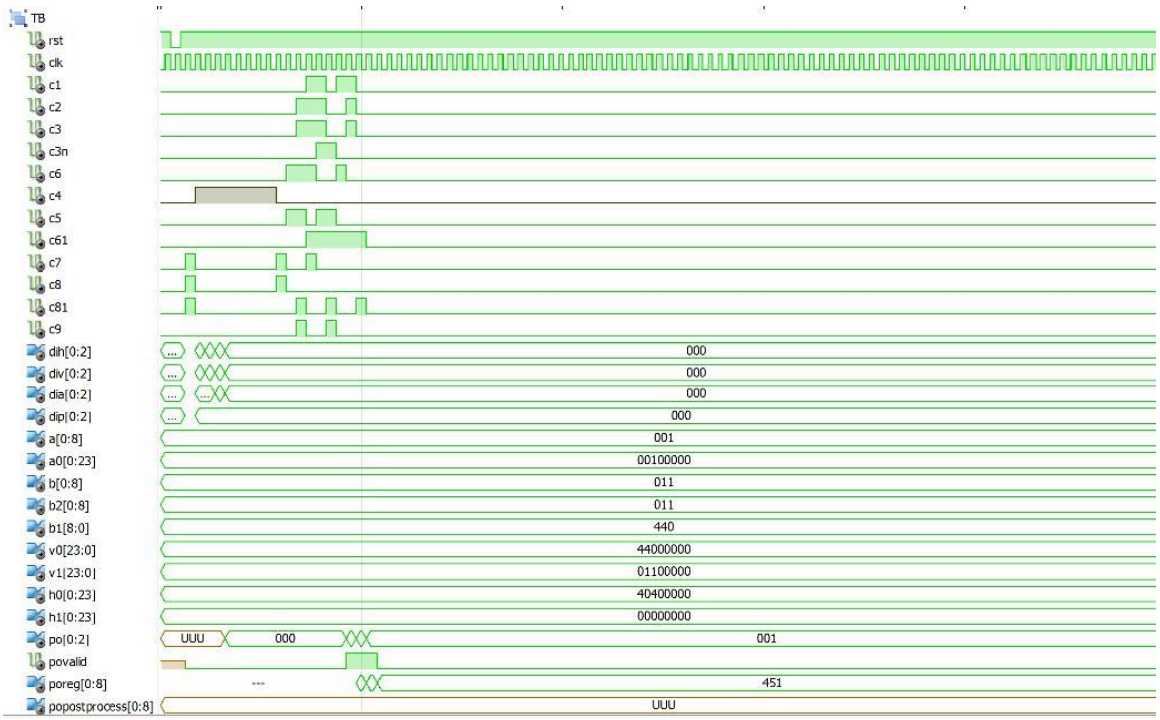


Fig. 9: Flow of the Processing Element

Table 1: Cycle Diagram (m=9,L=3,N_{PE}=2,N=3,N_C=2)

Cycle	-2	-1	0	1	2	3	4	5	6	7	8	9
PE0	Dc	Dc	$W_{T,L}(0,0)$	$W_{T,L}(1,0)$	$W_{T,L}(2,0)$	$W_{T,L}(0,2)$	$W_{T,L}(1,2)$	$W_{T,L}(2,2)$	dc	Dc	dc	Dc
PE1	Dc	Dc	Dc	$W_{T,L}(0,1)$	$W_{T,L}(0,2)$	$W_{T,L}(2,1)$	unused	unused	Dc	Dc	dc	Dc
P[0:2]	Dc	Dc	Dc	Dc	Dc	Dc	Dc	$P_4P_5P_6$	$P_7P_8P_0$	$P_1P_2P_3$	dc	Dc
V[0:2]	Dc	Dc	0,0:2	0,3:5	Dc	0,6:8	zero	Dc	Dc	Dc	dc	Dc
H[1:2]	Dc	Dc	1:2,0	4:5,0	7:8,0	0,5:4	0,2:1	1:2,0	Dc	Dc	dc	Dc
H[0]	Dc	Dc	0:0,0	3:3,0	6:6,0	Dc	0,3:3	0:0,0	Dc	Dc	dc	Dc
A[0:2]	Dc	Dc	0:2	3:5	Dc	6:8	Dc	Dc	Dc	Dc	dc	Dc
C1	0	0	0	1	1	0	1	1	0	0	0	0
C2	Dc	Dc	1	1	1	0	0	1	Dc	Dc	dc	Dc
C3	Dc	Dc	1	1	1	0	0	1	Dc	Dc	dc	Dc
C3n	Dc	0	0	0	1	1	0	Dc	Dc	Dc	dc	Dc
C6	Dc	1	1	1	0	0	1	Dc	Dc	Dc	dc	Dc
C5	Dc	1	1	0	1	1	0	Dc	Dc	Dc	dc	Dc
C9	Dc	0	0	1	0	0	1	Dc	Dc	Dc	dc	Dc
C7	1	0	0	1	0	0	Dc	Dc	Dc	Dc	dc	Dc
C8	1	0	0	0	0	0	0	0	0	0	0	0
C81	0	0	1	0	0	1	0	0	1	0	0	0

5. SIMULATION RESULT -(Propose Architecture)

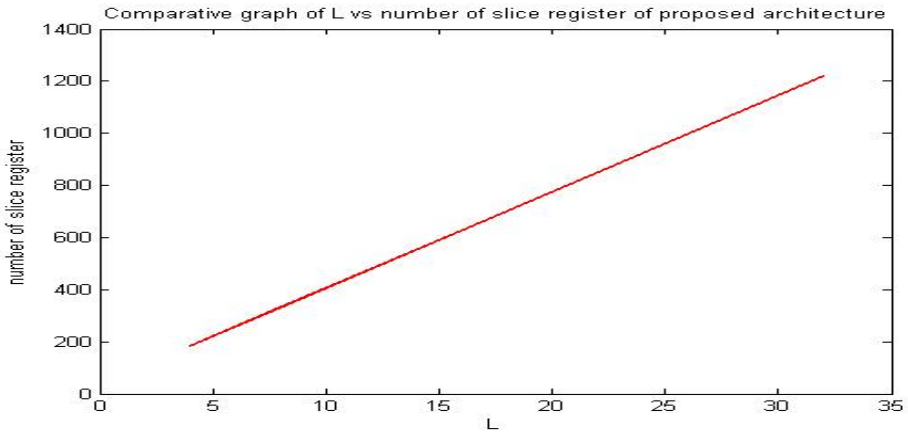


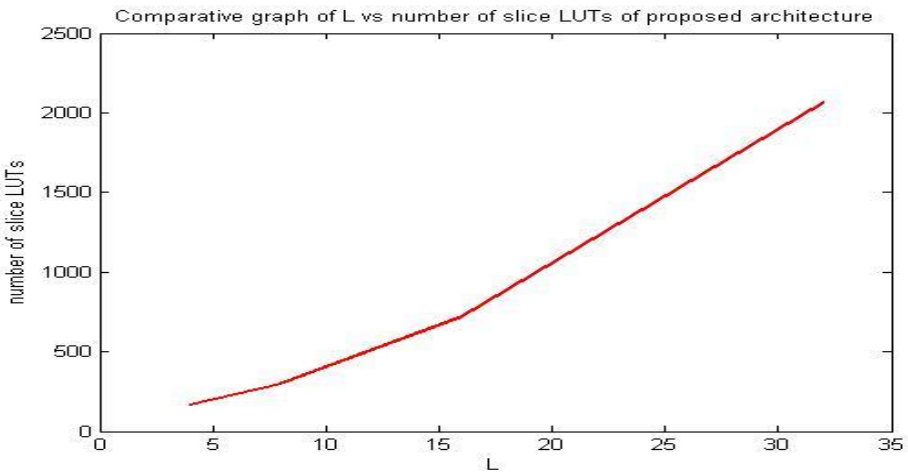
6. SYNTHESIS RESULT AND DISCUSSION

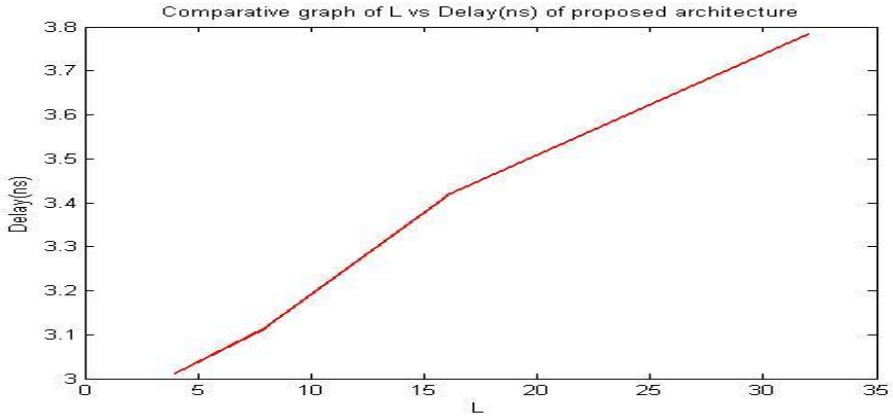
A comparative analysis done on the proposed two architecture for different parameters as shown in the tables given below with their ghraph.
Selected Device: xc7a100t-3csg324
Architecture-1
For M=159

Parameter	L=4,N=40	L=8,N=20	L=16,N=10	L=32,N=5
Number of Slice Registers (out of 126800)	209(0%)[S=8,D=3,NPE=3] 349(0%)[S=16,D=4,NPE=3] 617(0%)[S=32,D=5,NPE=3] 311(0%)[S=8,D=3,NPE=9] 617(0%)[S=8,D=3,NPE=27]	381(0%)[S=8,D=3,NPE=3] 649(0%)[S=16,D=4,NPE=3] 1173(0%)[S=32,D=5,NPE=3] 579(0%)[S=8,D=3,NPE=9] 1173(0%)[S=8,D=3,NPE=27]	725(0%)[S=8,D=3,NPE=3] 1249(0%)[S=16,D=4,NPE=3] 2285(1%)[S=32,D=5,NPE=3] 1115(0%)[S=8,D=3,NPE=9] 2285(0%)[S=8,D=3,NPE=27]	1413(1%)[S=8,D=3,NPE=3] 2449(1%)[S=16,D=4,NPE=3] 4509(3%)[S=32,D=5,NPE=3] 2187(1%)[S=8,D=3,NPE=9] 4509(3%)[S=8,D=3,NPE=27]
Number of Slice LUTs (out of 63400)	176(0%)[S=8,D=3,NPE=3] 248(0%)[S=16,D=4,NPE=3] 403(0%)[S=32,D=5,NPE=3] 319(0%)[S=8,D=3,NPE=9] 753(1%)[S=8,D=3,NPE=27]	307(0%)[S=8,D=3,NPE=3] 454(0%)[S=16,D=4,NPE=3] 688(1%)[S=32,D=5,NPE=3] 649(1%)[S=8,D=3,NPE=9] 1681(2%)[S=8,D=3,NPE=27]	729(1%)[S=8,D=3,NPE=3] 963(1%)[S=16,D=4,NPE=3] 1357(2%)[S=32,D=5,NPE=3] 1761(2%)[S=8,D=3,NPE=9] 4875(7%)[S=8,D=3,NPE=27]	2089(3%)[S=8,D=3,NPE=3] 2377(3%)[S=16,D=4,NPE=3] 3106(4%)[S=32,D=5,NPE=3] 5526(8%)[S=8,D=3,NPE=9] 15918(25%)[S=8,D=3,NPE

				=27]
Number of bonded IOBs (out of 210)	34(16%)	54(25%)	94(44%)	174(82%)
Delay	3.011ns,332.105MHz[S=8, D=3,NPE=3] 3.115ns,320.986MHz[S=16 ,D=4,NPE=3] 3.597ns,277.994MHz[S=32 ,D=5,NPE=3] 2.991ns,334.359MHz[S=8, D=3,NPE=9] 3.023ns,330.847MHz[S=8, D=3,NPE=27]	3.115ns,321.007MHz[S=8, D=3,NPE=3] 3.341ns,299.294MHz[S=16 ,D=4,NPE=3] 3.764ns,265.640MHz[S=32 ,D=5,NPE=3] 3.115ns,321.007MHz[S=8, D=3,NPE=9] 3.125ns,319.980MHz[S=8, D=3,NPE=27]	3.415ns,292.809MHz[S=8, D=3,NPE=3] 3.645ns,274.333MHz[S=16 ,D=4,NPE=3] 4.068ns,245.791MHz[S=32 ,D=5,NPE=3] 3.415ns,292.809MHz[S=8, D=3,NPE=9] 3.429ns,291.613MHz[S=8, D=3,NPE=27]	3.784ns,264.278MHz[S=8, D=3,NPE=3] 4.015ns,249.091MHz[S=16 ,D=4,NPE=3] 4.423ns,226.116MHz[S=32 ,D=5,NPE=3] 3.769ns,265.308MHz[S=8, D=3,NPE=9] 3.784ns,264.278MHz[S=8, D=3,NPE=27]







7. CONCLUSIONS

This paper presents an VLSI architecture for in-circuit configurable Montgomery multiplier (ICCMM) for $GF(2^m)$, generated by irreducible trinomials and AOPs. The necessary mathematical formulation is also presented in systemic way. The field size can be selected, subject to memory limit of the ASIC for which it is suitable, without any physical modification of the circuit. All the necessary memory systems and their architectures are also presented. The digit sizes (L) of all the PE's connected in pipeline are selected by specification of the memory cores available in the ASIC library. The number of PE's (N_{PE}) directly influences the cycle latency and area, in opposite manners. The proposed ICCMM is shown to have scalable latency and low value of latency can be achieved with the latency-flip-flop and the latency-EXOR products, values of which are smaller than the corresponding values of few well known bit-parallel and digit-serial architectures.

REFERENCE

1.R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge Univ. 80 Press, 1994.

2.S. Lin and D. J. Costello Jr., Error Control Coding: Fundamentals and Applications, PH Inc., 2004.

3.D. Hankerson, A. Menezes and S. Vanstone. Guide to Elliptic Curve Cryptography. Springer-Verlag, 2004.

4.A. P. Fournaris and O. Koufopavlou, "A New RSA Encryption Architecture and Hardware Implementation based on Optimized Montgomery Multiplication," *IEEE ISCAS 2005*, vol.5, pp:4645 – 4648.

5.E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," *PhD thesis*, Linkoping University, Sweden, 1991.

6.Jorge Guajardo Merchan, "Arithmetic Architectures for Finite Fields $GF(p^m)$ with Cryptographic Applications," *PhD Thesis*, Ruhr-Universität Bochum, Bochum, 2004.

7.Selçuk Baktir, "Frequency Domain Finite Field Arithmetic for Elliptic Curve Cryptography," *PhD Thesis*, Worcester Polytechnic Institute, 2008.

8.Arjen K. Lenstra and Eric R. Verheul, "Selecting Cryptographic Key Sizes," *J. Cryptology* (2001) 14: 255–293.

9.Janusz Rajaski, and Jerzy Tyszer, "Primitive Polynomials Over GF(2) of Degree up to 660 with Uniformly Distributed Coefficient," J. of Electronics Testing: Theory and Applications, vol. 19, pp: 645-657, 2003.

10. A. Reyhani-Masoleh, and M. A. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$", *IEEE TC*, vol.53, no.8, pp.945-959, 2004.

11.J.H. Guo and C. L. Wang, "Digit-serial systolic multiplier for finite field $GF(2^m)$," *IEE Computers and Digital Tech.*, vol. 145, no. 2, pp. 143-148, Mar. 1998.

12.C. H. Kim, C. P. Hong and S. Kwon, "A Digit–Serial Multiplier for Finite Field $GF(2^m)$", *IEEE Transactions on VLSI System*, vol. 13, No. 4, pp. 476-483, Apr. 2005.

13.C. H. Kim, S. D. Han and C. P. Hong, "An efficient digit-serial systolic multiplier for finite fields $GF(2^m)$," *14th Annual IEEE Inter. ASIC/SOC Conf. 2001*, pp.361–365, Sep. 2001.

14.M. Hütter, J. Großschädl and G.A. Kamendje, "A versatile and scalable digit-serial/parallel multiplier architecture for finite fields $GF(2^m)$," *4th International Conf. Info. Tech.: Coding and Computing (ITCC 2003)*, pp. 692–700.

15.Kee-Won Kim, Keon-Jik Lee and Kee-Young Yoo, "A new digit-serial systolic multiplier for finite fields $GF(2^m)$," *International Conf. on Info-tech and Info-net, 2001 (ICII 2001 - Beijing. 2001)*. vol. 5, pp.128 – 133, Nov. 2001.

16.S. Kumar, T. Wollinger and C. Paar, "Optimum digit serial $GF(2^m)$ multipliers for curve-based cryptography," *IEEE TC*, vol. 55, no. 10, oct. 2006.

17.P.L. Montgomery, "Modular Multiplication without Trial Division", *Math. Computation*, vol. 44, pp.519-521, 1985.

18.C.K. Koc and T. Acar, "Montgomery Multiplication in $GF(2^k)$," *3rd Annual Workshop on Selected Areas in Cryptography*, Queen's University, Kingston, Ontario, Canada, pp. 95-106, Aug. 1996.

19.C. K. Koc, T. Acar, and B. S. Kaliski Jr., "Analyzing and Comparing Montgomery Multiplication

Algorithms," *IEEE Micro*, vol.16, no. 3, pp. 26-33, Jun 1996.

20.A.F. Tenca and C.K. Koc, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm," *IEEE TC*, vol. 52, no. 9, Sep. 2003.

21.Germain Drolet, "A New Representation of Elements of Finite Fields $GF(2^m)$ Yielding Small Complexity Arithmetic Circuits," *IEEE TC*, vol 47, no. 9, pp. 938-946, Sep. 1998.

22.C. Y. Lee, J. S. Horng, I. C. Jou and E. H. Lu, "Low-complexity bit-parallel systolic Montgomery Multipliers for Special Classes of $GF(2^m)$", *IEEE TC*, vol. 54, no. 9, pp. 1061-1070, Sep. 2005.

23.C. Y. Lee, C. C. Chen and E. H. Lu, "Compact bit-parallel systolic Montgomery multiplication over $GF(2^m)$ generated by trinomials," *IEEE TENCON 2006*, pp. 1-4, Nov. 2006.

24.C.Y. Lee, C.W. Chiou, J.M. Lin, et al., "Scalable and systolic Montgomery multiplier over $GF(2^m)$ generated by trinomials", *IET circuits devices & systems*, vol. 1 (6), pp. 477-484, Dec. 2007.

25.C. C. Chen, C. Y. Lee and E. H. Lu, "Scalable and Systolic Montgomery Multipliers over $GF(2^m)$," *IEICE Trans. Fundamentals*, vol. E91-A, no. 7, pp. 1763-1771, Jul. 2008.

26.Somsubhra Talapatra, and Hafizur Rahaman, "Low Complexity Digit Serial Systolic Montgomery Multipliers for Special Class of $GF(2^m)$," *IEEE Trans. VLSI Syst.*, (accepted 16th Feb., 2009).

27.P.K. Meher, "Systolic and Super-Systolic Multipliers for Finite Field $GF(2^m)$ Based on Irreducible Trinomials," *IEEE Trans. Circuits & Syst.-I*, vol. 55, no. 4, May 2008.

28.J.P. David, and N. Tittley, "Hardware Complexity of Modular Multiplication and Exponentiation," *IEEE Trans. Comput.*, vol. 56, no. 10, pp. 1038-1319, Oct. 2007.

29.R.V. Kamala, M. Sudhakar and M.B. Srinivas, "An Efficient Reconfigurable Montgomery Multiplier Architecture for $GF(n)$," *9th EUROMICRO Conf. Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006*. pp:155 – 159, 2006.

30.M. Sudhakar, R.V. Kamala and M.B. Srinivas, "A Unified, Reconfigurable Architecture for Montgomery Multiplication in Finite Fields $GF(p)$ and $GF(2^n)$," *20th International Conf. VLSI Design, 2007*, pp:750-755, Jan. 2007.

31.M. Sudhakar, R.V. Kamala, M.B. Srinivas, "A bit-sliced, scalable and unified Montgomery multiplier architecture for RSA and ECC," IFIP Int. Conf. VLSI(VLSI - SoC 2007), pp:252 – 257, Oct. 2007.

32.Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolić, "Digital Integrated Circuits: A Design Perspective," Prentice Hall (India), 2nd ed., New Delhi, 2006.

33.Prasenjit Ray, Hafizur Rahaman, Somsubhra Talapatra, "Low Latency LSB First Bit-Parallel Systolic Multiplier over $GF(2^m)$," VLSI Design and Test 2008 (VDAT 2008), July 2008.

34.Wei. S.W.: "A systolic power-sum circuit for $GF(2^m)$," *IEEE TC*, vol.63(2), pp. 226-229, 1991.

35.Hafizur Rahaman, Prasenjit Roy, Debasis Mitra, and Amit. K. Dutta, "Area Efficient Bit-Serial Architecture for Polynomial Basis Multiplication over Galois Fields $GF(2^m)$," VLSI Design and Test 2008 (VDAT 2007), August 2007.