www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 7 July 2015, Page No. 13403-13410

An Efficient FPGA Implementation Of Multifunction Residue Architectures

Chembeti silpa, G.Mukesh,, M.Tech
Department of Electronics and communication Engineering
Audisankara College of Engineering & Technology, gudur
(Autonomous)

Abstract*—**A design methodology for incorporating Residue Number System (RNS) and Polynomial Residue Number System (PRNS) in Montgomery modular multiplication in ***GF(p) or ***GF(2^n)*** respectively, as well as a VLSI architecture of a dual-field residue arithmetic Montgomery multiplier are presented in this paper. An analysis of input/output conversions to/from residue representation, along with the proposed residue Montgomery multiplication algorithm, reveals common multiply-accumulate data paths both between the converters and between the two residue representations. A versatile architecture is derived that supports all operations of Montgomery multiplication in ***GF(p)*** and ***GF(2^n)***, input/output conversions, Mixed Radix Conversion (MRC) for integers and polynomials, dual-field modular exponentiation and inversion in the same hardware. Detailed comparisons with state-of-the-art implementations prove the potential of residue arithmetic exploitation in dual-field modular multiplication.

***Index Terms*—Computations in finite fields, computer arithmetic, Montgomery multiplication, parallel arithmetic and logic structures**

I. INTRODUCTION

A significant number of applications including cryptography, error correction coding, computer algebra, DSP, etc., rely on the efficient realization of arithmetic over finite fields of the form ***GF(2^n)***, where n € Z and n ≥ 1 or the form ***GF(p)*** where p a prime. Cryptographic applications form a special case, since, for security reasons, they require large integer operands [1]–[5].

Efficient field multiplication with large operands is crucial for achieving a satisfying cryptosystem performance, since multiplication is the most time- and area-consuming operation. Therefore, there is a need for increasing the speed of cryptosystems employing modular arithmetic with the least possible area penalty. An obvious approach to achieve this would be through Parallelization of their operations. In recent years, RNS and PRNS have enjoyed renewed scientific interest due to their ability to perform fast and parallel modular arithmetic [6]–[13]. Using RNS/PRNS, a given path serving a large data range is replaced by parallel paths of smaller dynamic ranges, with no need for exchanging information between paths. As a result, the use of residue systems can offer reduced complexity and power consumption of arithmetic units with large word lengths [14].On the other hand, RNS/PRNS implementations bear the extra cost of input converters to translate numbers from a standard binary format into residues and output converters to translate from RNS/PRNS to binary representations [14].

A new methodology for embedding residue arithmetic in a dual-field Montgomery modular multiplication algorithm for integers in ***GF(p)*** and for polynomials in is presented in ***GF(2^n)*** this paper. The mathematical conditions that need to be satisfied for a valid RNS/PRNS incorporation are examined. The derived architecture is highly parallelizable and versatile, as it supports binary-to-RNS/PRNS and RNS/PRNS-to-binary conversions, Mixed Radix Conversion (MRC) for integers and polynomials, dual-field Montgomery multiplication, and dual-field modular exponentiation and inversion in the same hardware.

The rest of the paper is organized as follows. A brief overview of related previous work is offered in Section II, while the main concepts of RNS and PRNS are summarized in Section III. In Section IV, basic finite-field arithmetic concepts are provided and the operation of field multiplication is defined. Following ***GF(2^n)*** and ***GF(p)*** Montgomery multiplication algorithms are presented. In Section VI, the proposed RNS/PRNS Montgomery multiplication algorithm is analyzed. The mathematical conditions that allow a valid incorporation of residue arithmetic in the Montgomery algorithm are also presented. In section VI, a detailed overview of the proposed dual-field Montgomery multiplier architecture is provided. Section VII provides time and area complexity measurements, as well as comparisons with other state-of-the-art implementations. Conclusions are offered in Section VIII.

II. PREVIOUS WORK

Important progress has been reported lately regarding ***GF(2^n)*** implementations. The Massey-Omura algorithm the introduction of optimal normal bases and their software and hardware implementations the Montgomery algorithm for multiplication in ***GF(2^n)*** as well as PRNS application in ***GF(2^n)*** multiplication, are, among others, important advances [9], [10], [12], PRNS incorporation in field multiplication, as proposed in [9], is based on a straightforward implementation of the Chinese Remainder Theorem (CRT) for polynomials which requires large storage resources and many pre-computations. The multipliers proposed in [10], [20], perform multiplication in PRNS, but the result is converted back to

polynomial representation. This limitation makes them inappropriate for cryptographic algorithms which require consecutive multiplications. Finally, an algorithm which employs trinomials for the modulus set and performs PRNS Montgomery multiplication has been proposed [12]. However, there is no reference to conversion methods and the use of trinomials may issue limitations in the PRNS data range.

GF(p) implementations have also withstood great analysis, with the Montgomery algorithm being used in the majority of them. Montgomery multiplication designs fall into two categories. The first includes fixed-precision input operand implementations, in which the multiplicand and modulus are processed in full word length, while the multiplier is handled bit-by-bit. These designs are optimized for certain word lengths and do not scales efficiently for departures from these word lengths. Their performance has been improved by high-radix algorithms and architectures.

The second category includes scalable architectures for variable word-length operands, based on algorithms, in which the multiplicand and modulus are processed word by word, while the multiplier is consumed bit by bit.

Montgomery's algorithm has also become a predicate for dual-field implementations. The Montgomery architectures perform well for RSA key word lengths, by processing word-size data, since RSA key sizes (512, 1024, 2048, etc.) are always multiples of word size. However, in ECC, key sizes are not integer multiples of word size, meaning that, if these architectures were to be used in ECC, they would require more clock cycles for their execution and thus more power consumption. An architecture configured at bit-level overcomes this problem

Finally, methods for embedding RNS in Montgomery multiplication have also been proposed [6], [7], [13]. Detailed analysis and comparisons with such works are offered in Section VII.

III. RESIDUE ARITHMETIC

A. *Residue Number System*

RNS consists of a set of L, pair-wise relatively prime integers $A= (m_1,m_2,..,.m_L)$ (called the *base*) and the range of the RNS is computed as $A=\pi_{i=1}^{L} m_i$. Any integer $z \in [0, A-1]$ has a unique RNS representation z_A given by $z_A= (z_1,z_2,..z_L) = ((z)_{m1} ,(z)_{m2} ,\ldots (z)_{mL})$, where $(z)_{mi}$ denotes the operation z mod m_i. Assuming two integers a,b in RNS format, i.e., $a_A= (a_1,a_2,\ldots,a_L)$ and $b_A = (b_1, b_2,\ldots b_L)$ then one can perform the operations $\in (+, -, *)$ in parallel by $a_A \; b_A= ((a_1 \; b_1)_{m1} , (a_2 \; b_2)_{m1} , ,(a_L \; b_L)_{mL} ,$

To reconstruct the integer from its residues, two methods may be employed [14]. The first is through the CRT according to $z=\sum_{k=0}^{n} (z_i \; A_i^{-1})_{mi} \otimes A_i - \gamma A \otimes$ where $A_i=A/m_i$, A_i^{-1} is the inverse of A_i modulo m_i and γ is an integer correction factor.

The second method is through the MRC. The MRC of an integer with an RNS representation $z_A= (z_1,z_2,..z_L)$ is given by $z = U_1 + W_2 U_2 + \ldots. + W_L U_L$ where $W_i = \pi_{j=1}^{i-1} m_j$, for all $i \in [2,L]$ and the U_i s are computed according to

$$U_1 = z_1$$
$$U_2 = (z_2 - z_1)\, m_2$$
$$U_3 = (z_3 - z_1 - W_2 U_2)\, m_3$$
$$\ldots.$$
$$\ldots..$$

$U_L= (z_L - z_1 - W_2 U_2 - W_3 U_3 - \ldots - W_{L-1}U_{L-1})m_L$ providing that the predetermined factors $V_1 \equiv 1$ and $V_i = ((\pi_{j=1}^{i-1} m_j$, for all $i \in [2,L]$ are all unity . Of the two methods, the proposed architecture uses the MRC, as it avoids the problem of evaluating the correction factor of (2).

B. *Polynomial Residue Number System*

Similar to RNS, a PRNS is defined through a set of *L*, pair-wise relatively prime polynomials $A=(m_1(x), m_2(x), \ldots m_L(x))$. We denote by $A(x) = \pi_{i=1}^{L} m_i(x)$ the dynamic range of the PRNS. In PRNS, every polynomial $z(x) \in$ ***GF(2^n)*** **,** with deg{ z(x) } < deg { A(x) }, has a unique PRNS representation: $z_A = (z_1,z_2,\ldots z_L)$ such as $z_i = z(x) \bmod m_i(x)$, $i \in [1,L]$, denoted as $(z)m_i$. In the rest of the paper, the notation "(x)" to denote polynomials shall be omitted, for simplicity. The notation will be used interchangeably to denote either an integer or a polynomial z(x), according to context.

Assuming the PRNS representation $a_A= (a_1,a_2,\ldots a_L)$ and $b_A= (b_1,b_2,\ldots b_L)$ of two polynomials $a,b \in$ ***GF(2^n),*** then all operations $\in (+, - , *)$, can be performed in parallel, as $a_A \; b_A= ((a_1 \; b_1)_{m1} , (a_2 \; b_2)_{m1} , \ldots,(a_L \; b_L)_{mL} ,$ Conversion from PRNS to weighted polynomial representation is identical to the MRC for integers. The only difference is that, the subtractions in (4) are substituted by polynomial additions.

IV. MONTGOMERY MULTIPLICATION

A. *GF(p) Arithmetic*

Field elements in ***GF(p)*** are integers in [0,p-1] and arithmetic is performed modulo. Montgomery's algorithm for modular multiplication without division [43] is presented below, as Algorithm 1, in five steps, where R is the Montgomery radix, gcd(R,p) = 1,and p<R. R must be chosen so that steps 2 and 5 are efficiently computed.

Algorithm 1 Montgomery modular multiplication

Input : a,b,p,R,R^{-1} /* $a,b<p$

Output : $c \equiv abR^{-1} \bmod p$, /* $c< 2p$

1. $s \leftarrow a.b$
2. $t \leftarrow s.(-p^{-1}) \bmod R$
3. $u \leftarrow t.p$
4. $v \leftarrow s+u$
5. $c \leftarrow v/R$

It is usually chosen to be a power of 2, when radix-2 representation is employed. Since Montgomery's method was originally devised to avoid divisions, it is well-suited to RNS implementations, considering that RNS division is inefficient to perform.

B. *GF(2^n) Arithmetic*

Field elements in ***GF(2^n)*** are polynomials represented as binary vectors of dimension , relative to a given polynomial basis $(1, \alpha , \alpha^2 , \ldots \alpha^{n-1})$, where α is a root of an irreducible polynomial of degree over ***GF(2).*** The field is then realized as ***GF(2)***[x]/(p) and the arithmetic is that of polynomials of degree at most n-1, modulo p[1].

The addition of two polynomials a and b in ***GF (2^n)*** is performed by adding the polynomials, with their coefficients added in ***GF(2)*** i.e., modulo 2. This is equivalent to a bit-wise XOR operation on the vectors. The product of two elements a

and b in GF(2^n) is obtained by computing c = a . b mod p where is a polynomial of degree atmost n-1 and c € ***GF (2^n).*** A Montgomery multiplication algorithm suitable for polynomials in ***GF (2^n)*** has been proposed [19]. Instead of computing the product c = a . b mod p , the algorithm computes c = a.b.R^{-1}mod p, with deg{ c(x) < n} and R is a special fixed element in ***GF (2^n).*** The selection of R(x) = x^n is the most appropriate, since modular reduction and division by x^n are simple shifts [3]. The algorithm is identical to Algorithm 1, except from the constant $-p^{-1}$ in step 2, which is p^{-1} in ***GF (2^n).***

V. A NEW METHODOLOGY FOR EMBEDDING RESIDUE ARITHMETIC IN MONTGOMERY MULTIPLICATION

A. *Embedding RNS in GF(p) Montgomery Multiplication*

An MRC-based algorithm [44] that avoids the evaluation of the γ factor of (2) forms the basis of the proposed RNS-based Montgomery multiplication algorithm. The derived algorithm is briefly presented here as Algorithm 2, and extended for the case of ***GF (2^n) .***

Two RNS bases are introduced, namely A= ($p_1, p_2, \ldots, p_L$) and B= ($q_1,q_2,..q_L$), such that gcd(p_i,q_j) = 1, for all i, j € [1,L]. The 5 steps of the Montgomery algorithm are translated to RNS computations in both bases, denoted from now on as Ί = A U B.

Initially, the inputs are expressed in RNS representation in both bases as a_I and b_I. Steps 1, 3, and 4 of Algorithm 1, involve addition and multiplication operations, thus their transformation to RNS is straightforward. For steps 2 and 5, the Montgomery radix is replaced by Q = $\pi_{i=1}^{L} q_i$, which is the range of B .We also denote as P = $\pi_{i=1}^{L} p_i$ the range of base A. Then, in the second step, in RNS format is computed in base B by $t_B = s_B.B^{-1}$. Nevertheless, the computations in base cannot be continued for steps 3, 4, and 5 of Algorithm 1, since in step 5 we would need to compute a quantity of the form Q^{-1} mod q_i, which does not exist since q_i s are factors of Q. Thus, a base conversion (BC) step, from base to base , is inserted, to compute t_A.t_A is then used to execute the old steps 3, 4, and 5 in base .The result at the end of this algorithm is a quantity c_I in RNS format that equals c ≡ abQ^{-1} mod p, since BC is error-free.

Algorithm 2 RNS Montgomery multiplication (RMM)

Input : a_I,b_I,$(-p^{-1})$, Q_A^{-1}, p_A a,b < 2p

Output : c_I , /* c< 2p and c ≡ abQ^{-1} mod p

1. $s_I \leftarrow a_I . b_I$
2. $t_B \leftarrow s_B . (-p^{-1})_B$
3. $t_A \leftarrow t_B$ /* base conversion step
4. $u_A \leftarrow t_A . p_A$
5. $v_A \leftarrow s_A + u_A$
6. $c_A \leftarrow v_A Q_A^{-1}$
7. $c_B \leftarrow c_A$ /* base conversion step

In Algorithm 2, inputs and output are all less than 2_p, so that they are compatible with each other. This allows the construction of a modular exponentiation algorithm by repetition of the RNS Montgomery multiplication. Base conversion in step 7 is utilized for the same reason. Algorithm 3 depicts the proposed base conversion process that converts an integer ζ expressed in RNS base B as ζ_B to the RNS representation of another base A. In contrast to other RNS Montgomery multiplication algorithms which also employ MRC the proposed one implements a simplified version of the MRC in (3) and (4) which, as will be shown in next sections, not only reduces the total complexity of the algorithm but also offers better opportunities for parallelization of operations. Dual-field circuitry is also not offered by the works in. Algorithm 3 implements (4) in steps 1–8 to obtain the mixed-radix digits U_i of ζ. In steps 9, (3) is realized, while the whole summation is computed modulo each modulus p_i of the new base A. The two base conversions in the RMM algorithm are error-free, contrasted to other algorithms that employ CRT and utilize approximation methods to compute the correction factor γ of (2) [6], [7] . Conditions gcd(Q,p)=1 and gcd(P,Q)=1 are sufficient for the existence of $(-p_B^{-1})$ and Q_A^{-1}, respectively. As it holds that

c = $\frac{v}{Q} = \frac{ab+tp}{Q} < \frac{(2p)^2}{Q} + \frac{Q_p}{Q} = (\frac{4p}{Q} + 1)$ p ≤ 2p it follows that 4p ≤ Q is sufficient for c < 2p to hold when a, b < 2p. Finally (8) also shows that 2p ≤ P is sufficient for c < Q and v < PQ. Since v is the maximum intermediate value, all values are less than PQ[6].

B. *Embedding PRNS in GF(2^n) Montgomery Multiplication*

A modification of the Montgomery algorithm for multiplication in *GF(2^n)* that encompasses PRNS is proposed next. The proposed algorithm employs general polynomials of any degree, and is an extension of an algorithm [12], which employs trinomials for the PRNS modulus set. Additionally, the proposed algorithm addresses the problem of converting data to/from PRNS representation. In contrast to a similar algorithm in which employed CRT for polynomials for the BC algorithm, the proposed architecture employs MRC. This allows for dual-field RNS/PRNS implementation, which is not supported in [45], and a new methodology to implement RNS-to-binary conversion as will be shown in Section V-D.

The proposed algorithm for *GF(2^n)* PRNS Montgomery multiplication (PRMM) is presented below as Algorithm 3. The only difference is that integer additions/subtractions and multiplications are replaced by polynomial ones. Again, the degree of input and output polynomials are both less than n, which allows the construction of a modular exponentiation algorithm by repetition of the PRMM. Base conversion is employed for the same reason.

1) Proof of PRMM Algorithm's Validity:

Theorem 1: If (1) gcd {p,Q}=1, (2) gcd{Q,p}=1, (3) deg{P} > n, then Algorithm 4 outputs c_I, for which c = abQ^{-1}mod p and degree { c } < n.

Proof: Since gcd {p,Q}=1 and gcd {Q,P}=1,p is relatively prime to Q and Q is relatively prime to P. Thus, the quantities $(p^{-1})q_i$ and (Q^{-1})p_i exist for all € [1,L], and therefore $p_B^{-1}$and Q_A^{-1} exist.

Assume that the polynomial v is a multiple of Q , i.e., v mod Q = 0. Then, $s + t . p = 0$ mod Q, which means that $t = s . p^{-1}$ mod Q. This corresponds to step 2 of the PRMM algorithm,

which means that step 6 is error-free since base conversion in step 3 is error-free, therefore PRMM holds.

Furthermore, it must be proved that the resulting polynomial c of Algorithm 4 is a polynomial of degree less than n . It holds that

deg { c } ≤ deg { $\frac{v}{Q}$ } ⟹

$$\deg\{c\} \le \deg\{\frac{ab+tp}{Q}\} \Rightarrow$$
$$\deg\{c\} \le \deg\{ab+tp\} - \deg\{Q\} \Rightarrow$$
$$\deg\{c\} \le \max\{\deg\{ab\}, \deg\{tp\}\} - \deg\{Q\} \Rightarrow$$
$$\deg\{c\} \le \deg\{tp\} - \deg\{Q\} \Rightarrow$$
$$\deg\{c\} \le \deg\{Q\} - 1 + n - \deg\{Q\} \Rightarrow$$
$$\deg\{c\} \le n - 1 \quad (9)$$

Since v is the maximum intermediate value of Algorithm 4, its degree must be less than the degree of the polynomial PQ. Under this assumption, we get

$$\deg\{v\} < \deg\{PQ\} \Rightarrow$$
$$\deg\{cQ\} < \deg\{PQ\} \Rightarrow$$
$$\deg\{c\} + \deg\{Q\} < \deg\{P\} + \deg\{Q\} \Rightarrow$$
$$\deg\{c\} < \deg\{P\} \quad (10)$$

From (9) and (10) we have

$$\deg\{c\} < n$$
$$\deg\{c\} < \deg\{P\}, \text{ both} \Rightarrow \deg\{P\} > n \quad (11)$$

Finally, note that (10) is independent of deg $\{Q\}$, thus selecting $\deg\{Q\} > n$ is sufficient.

C. Conversions

In the following discussion, base A =(p_1, p_2, …..p_L) shall n be used as an example to analyze the conversions to/from residue representations, without loss of generality.

Algorithm 3.PRNS Montgomery Multiplication (PRMM) in *GF(*2^N) :

Input : $a_I, b_I, p_B^{-1}, Q_A^{-1}, p_A$, / * deg{ a} < n ,deg{ b} < n

Output : c_I, /*deg {c} < n, $c = abQ^{-1}$ mod p

1. $s_I \leftarrow a_I . b_I$
2. $t_B \leftarrow s_B . p_B^{-1}$
3. $t_A \leftarrow t_B$
4. $U_A \leftarrow t_A . p_A$
5. $v_A \leftarrow s_A + u_A$
6. $c_A \leftarrow v_A . Q_A^{-1}$
7. $c_B \leftarrow c_A$

1) Binary-to-Residue Conversion: A radix- 2^r representation of an integer z as an L-tuple ($Z^{(L-1)}$,….., $Z^{(0)}$) satisfies

$$\boldsymbol{z = \sum_{i=0}^{L-1} z^{(i)} 2^{ri} = \left(2^{r(L-1)}, \ldots, 2^r, 1\right) \begin{bmatrix} z^{(L-1)} \\ . \\ . \\ z^{(1)} \\ z^{(0)} \end{bmatrix}} \quad (14)$$

where,$0 \le z^{(i)} \le 2^r - 1$. A method to compute z_A must be devised, that matches the proposed DRAMM's multiply-accumulate structure. By applying the modulo p_j operation in (14) we obtain

$$\langle z \rangle_{pj} = \langle \sum_{i=0}^{L-1} z^{(i)} \langle 2^{ri} \rangle pj \rangle pj, \ \forall j \in [1, L] \quad (15)$$

If constants $\langle 2^{ri} \rangle_{pj}$ are pre-computed, this computation is well suited to the proposed MAC structure and can be computed in L steps, when executed by units in parallel. Similar to the integer case, a polynomial z(x) Є GF(2^n) can be written as

$$\boldsymbol{z = \sum_{i=0}^{L-1} z^{(i)} x^{ri} = \left(x^{r(L-1)}, \ldots, x^r, 1\right) \begin{bmatrix} z^{(L-1)} \\ . \\ . \\ z^{(1)} \\ z^{(0)} \end{bmatrix}} \quad (16)$$

Applying the modulo p_j operation in (16) it yields

$$\langle z \rangle_{pj} = \langle \sum_{i=0}^{L-1} z^{(i)} \langle x^{ri} \rangle pj \rangle pj, \ \forall j \in [1, L] \quad (17)$$

Which is a similar operation to (15), if $\langle x^{ri} \rangle_{pj}$ are pre-computed. From (15) and (17), it is deduced that conversions in both fields can be unified into a common conversion method, if dual field circuitry is employed, as already mentioned for the case of the DRAMM and DBC. In the rest of the paper, the radix vectors ($2^{r(L-1)}$,…., 2^r,1) and ($x^{r(L-1)}$,…, x^r,1) of both fields shall be denoted as a common radix vector V , without loss of generality.

2) Residue-to-Binary Conversion: As all operands in (4) are of word length r, they can be efficiently handled by an r-bit MAC unit. However, (3) employs multiplications with large values, namely the W_i s.

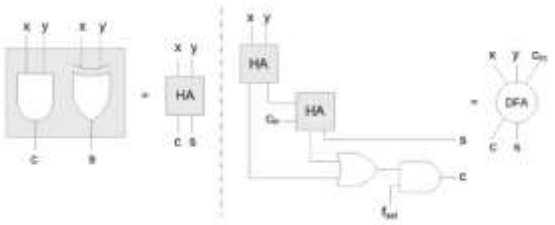

Fig.1. Dual-field full-adder cell (DFA).

To overcome this problem, (3) can be rewritten in matrix notation, as in (18), shown at the bottom of the page, which implies a fully parallel implementation of the conversion process. The inner products of a row i are calculated in parallel in each MAC unit. Each MAC then propagates its result to subsequent MACs, so that at the end the last MAC(L) outputs the radix-2^r digit $Z^{(i)}$ of the result. In parallel with this summation, inner products of the next row i+1 can be formulated, since the adder and multiplier of the proposed MAC architecture may operate in parallel.

VI. HARDWARE IMPLEMENTATION

A. Dual-Field Addition/Subtraction

A dual-field full-adder (DFA) cell (Fig. 1) is basically a full-adder (FA) cell, equipped with a field-select signal (f_{sel}) that controls the operation mode [33]. In the proposed implementation, 3-level, carry-look ahead adders (CLA) with 4-bit carry-look ahead generator groups (CLG) are employed [47]. An example of a 4-bit dual-field CLA adder is shown in Fig.2.

$$z = \left\{ U_1 \otimes \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix} \otimes U_2 \otimes \begin{bmatrix} 0 \\ \vdots \\ 0 \\ W_2^{(1)} \\ W_2^{(0)} \end{bmatrix} \otimes U_3 \otimes \begin{bmatrix} 0 \\ \vdots \\ W_3^{(2)} \\ W_3^{(1)} \\ W_3^{(0)} \end{bmatrix} \otimes \cdots \otimes U_L \otimes \begin{bmatrix} W_L^{(L-1)} \\ \vdots \\ W_L^{(2)} \\ W_L^{(1)} \\ W_L^{(0)} \end{bmatrix} \right\} \otimes V$$

$$= \begin{bmatrix} 0 & \otimes & \cdots & \otimes & 0 & \otimes & \cdots & \otimes & U_L \otimes W_L^{(L-1)} \\ \vdots & \otimes & \vdots & \otimes & \vdots & \otimes & \vdots & \otimes & \vdots \\ 0 & \otimes & 0 & \otimes & U_3 \otimes W_3^{(2)} & \otimes & \cdots & \otimes & U_3 \otimes W_3^{(2)} \\ 0 & \otimes & U_2 \otimes W_2^{(1)} & \otimes & U_3 \otimes W_3^{(1)} & \otimes & \cdots & \otimes & U_3 \otimes W_3^{(1)} \\ U_1 & \otimes & U_1 \otimes W_2^{(0)} & \otimes & U_1 \otimes W_3^{(0)} & \otimes & \cdots & \otimes & U_1 \otimes W_L^{(0)} \end{bmatrix} \otimes V = \begin{bmatrix} z^{(L-1)} \\ \vdots \\ z^{(2)} \\ z^{(1)} \\ z^{(0)} \end{bmatrix} \otimes V \qquad (18)$$

The GAP modules generate the signals pi= xi XOR yi, gi= xi AND yi, $\propto i = xi\ OR\ yi$ and AND gates along with a signal control whether to eliminate carries or not. The carry-look ahead generator is an AND-OR network based on (19) .

$$c_{i+1} = \overset{i}{\underset{j=0}{\mathrm{OR}}} \left(\overset{i}{\underset{k=j+1}{\mathrm{AND}}} \alpha_k \right) g_j \mathrm{OR} \left(\overset{i}{\underset{k=0}{\mathrm{AND}}} \alpha_k \right) c_0 \qquad (19)$$

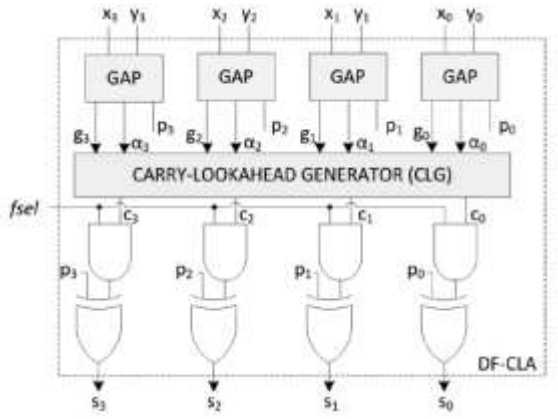

Fig. 2. Dual-field CLA.

1) Dual-Field Modular/Normal Addition/Subtraction*:*

With trivial modifications of algorithms for modular addition/subtraction in GF(p)[3], [4], a dual-field modular adder/subtractor (DMAS) shown in Fig. 3 can be mechanized using CLA adders. When , fsel=0, the circuit is in GF(2^n) mode and the output is derived directly from the top adder which performs a GF(2^n) addition. When fsel=1 , the circuit may operate either as a normal ($2r+\log_2 L$)-bit adder/subtracter (conv_mode = 0)or as a modular adder/subtracter (conv_mode = 1).

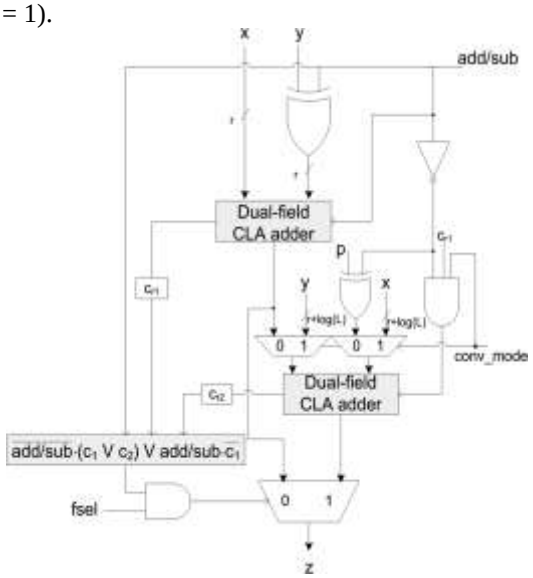

Fig.3.Dual-field modular/normal adder/subtracter (DMAS).

In the first case, the output is the concatenation of the outputs of the two adders. This is required during residue-to-binary conversion, since (18) dictates that L ,2r -bit quantities need to be added recursively via a normal adder.

B. Dual-Field Multiplication

A parallel tree multiplier, which is suitable for high-speed arithmetic, and requires little modification to support both fields, is considered in the proposed architecture. Regarding input operands, either integers or polynomials, partial product generation is common for both fields, i.e., an AND operation among all operand bits. Consequently, the addition tree that sums the partial products must support both formats. In GF(2^n) mode, if DFA cells are used, all carries are eliminated and only XOR operations are performed among partial products.

In GF(p) mode, the multiplier acts as a conventional tree multiplier. A 4 * 4-bit example of the proposed dual-field multiplier (DM) with output in carry-save format is depicted in Fig. 4.

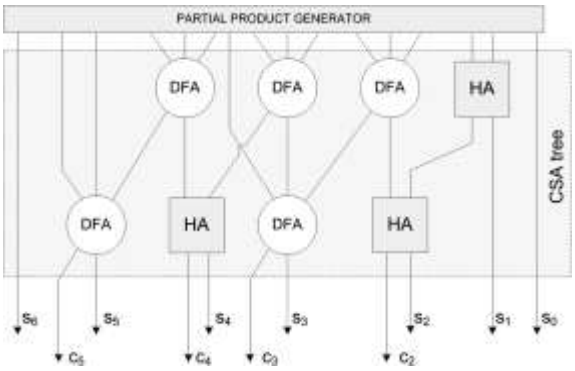

Fig. 4. Dual-field multiplier (DM)

C. Dual-Field Modular Reduction

A final modular reduction by each RNS/PRNS modulus is required, for each multiplication outcome, within each MAC unit. From several modular reduction strategies [3], a method based on careful modulus selection is utilized, since, not only it offers efficient implementations but also provides the best unification potential at a low area penalty. Assume a -bit product that needs to be reduced modulo an integer modulus . By selecting p_i of the form $2^r - \mu i$, where the h –bit $\mu i \ll 2r$, the modular reduction process can be simplified as

$$\langle c \rangle_{p_i} = \left\langle \overbrace{\sum_{i=0}^{r-1} c_i 2^i}^{E} + 2^r \overbrace{\sum_{i=0}^{r-1} c_{r+i} 2^i}^{F} \right\rangle_{p_i} = \overbrace{\langle E + 2^r F \rangle_{p_i}}^{d}$$

$$= \left\langle \overbrace{\sum_{i=0}^{r-1} d_i 2^i + \mu_i \sum_{i=0}^{h} d_{r+i} 2^i}^{\xi} \right\rangle_{p_i} \qquad (20)$$

From (20), it is apparent that

$$\langle c \rangle_{p_i} = \begin{cases} \sum_{i=0}^{r-1} \xi_i 2^i, \xi < 2^r - \mu \\ \sum_{i=0}^{r-1} \xi_i 2^i + \mu_i, 2^r - \mu < \xi < 2^r \end{cases} \qquad (21)$$

The same decomposition can be applied to polynomials and consequently, if dual-field adders and dual-field multipliers are employed, a dual-field modular reduction (DMR) unit can be mechanized as shown in Fig. 5. The word length of can be limited to a maximum of 10 bits for a base with 66 elements.

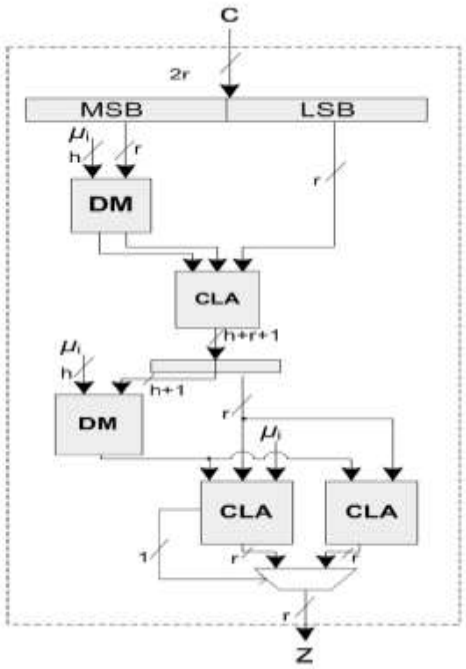

Fig. 5. Dual-field modular reduction unit (DMR).

D. MAC Unit

The circuit organization of the proposed MAC unit is shown in Fig. 6. Its operation is analyzed below in three steps, corresponding to the three phases of the calculations it handles, i.e., binary-to-residue conversion, RNS/PRNS Montgomery multiplication, and residue-to-binary conversion.

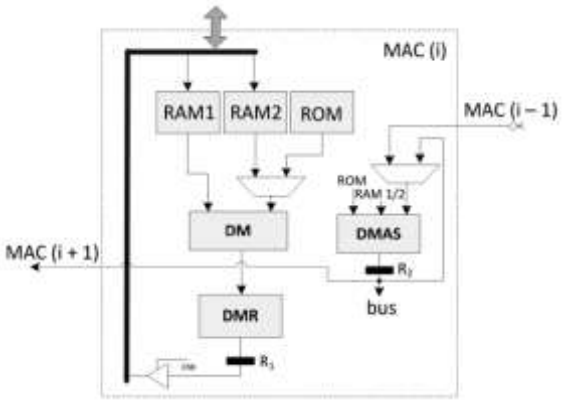

Fig.6. The proposed MAC unit.

1) Binary-to-Residue Conversion*:*

Initially, -bit words of the input operands, as implied by (15), are cascaded to each MAC unit and stored in RAM1 at the top of Fig. 6. These words serve as the first input to the multiplier, along with the quantities which are stored in a ROM. Their multiplication produces the inner products of (15) or (17) which are added recursively in the DMAS unit. The result is stored via the bus in RAM1. The process is repeated for the second operand and the result is stored in RAM2, so that when the conversion is finished, each MAC unit holds the residue digits of the two operands in the two RAMs. The conversion requires steps to be executed.

2) Montgomery Multiplication:

The first step of the proposed DRAMMis a modular multiplication of the residue digits of the operands. Since these digits are immediately available by the two RAMs, a modular multiplication is executed and the result in is stored in RAM1 for base and RAM2 for base . Step 2 of DRAMM is a multiplication of the previous result with a constant provided by the ROM. The results correspond to inputs of the DBC algorithm and are stored again in RAM1. All MAC units are updated through the bus with the corresponding RNS digits of all other MACs and a DBC process is initiated.

To illustrate the DBC process, a task distribution graph is presented in Fig. 7 for a DRAMM requiring moduli.

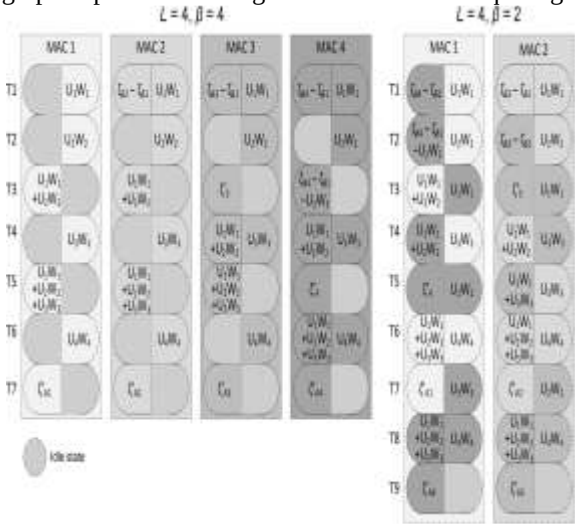

Fig.7. Task distribution in the proposed DRAMM.

Two cases are represented; the first corresponds to a fully parallel architecture with units and the second shows how the tasks can be overlapped when only MAC units are available. Each MAC unit has been assigned to a different color, thus in the overlapped case the color codes signify when a MAC unit performs operations for other units. In the example of Fig. 6, MAC(1) handles MAC(4) and MAC(2) handles MAC(3). In each cycle, modular additions and multiplications are performed in parallel in each MAC. To depict this, each cycle is split in two parts: the operations on the left correspond to modular additions and on the right to modular multiplications. The results obtained by each operation are depicted in each cycle, while idle states are denoted by dashed lines. An analysis on the number of clock cycles required, and how MAC units can be efficiently paired is presented in the next section.

Table I
Normalized Area and Delay Of The Proposed DRAMM Architecture

Module	Area	Critical path delay
DFA	$A_{FA}+A_{AND}$	$T_{FA}+T_{AND}$
DF-CLA[1]	$3rA_{DFA}$	$2\log_4(r)T_{DFA}$
DMAS[7]	$2A_{DF-CLA}+(3r+\log_2 L)A_{mux(2-1)}$	$2\log_4(r+\log_2(L)+1)T_{DFA}+T_{mux(2-1)}$
DM[8]	$(r-2)A_{DFA}+r^2A_{AND}+2A_{DF-CLA}$	$T_{DF-CLA}+(\log r)T_{DFA}+T_{AND}$
DMR	$A_{DM}{}^2+A_{DM}{}^3+A_{DF-CLA}{}^4+A_{DF-CLA}{}^5+$ $+A_{DF-CLA}{}^6+(h+r+1)A_{FF}+rA_{mux(2-1)}$	$(\log h+6\log_4(h+r+1))T_{DFA}$
MAC	$A_{DM}+A_{DMAS}+A_{DMR}+$ $+2rA_{mux(2-1)}+(2r)A_{FF}$	$T_{DM}+T_{DMR}$
DRAMM	LA_{MAC}	T_{MAC}

[1] r-bit DF-CLA. [2] $(h \times r)$-bit. [3] $(h \times (h+1))$-bit. [4] 3-input $(h+r+1)$-bit DF-CLA. [5] 3-input r-bit DF-CLA. [6] 2-input r-bit DF-CLA. [7] $(r+\log L)$-bit adders. [8] $(r \times r)$-bit.

3) Residue-to-Binary Conversion*:*

Residue-to-binary conversion is essentially a repetition of the DBC algorithm, except for steps 9–14, which is no longer modulo operations. To illustrate the conversion process, assume the generation of the inner products in row 1 of (18). Each product is calculated in parallel in each MAC unit and a "*carry-propagation*" from MAC(1) to is performed to add all inner products. When summation finishes the first digit of the

result is produced in . In parallel with this "*carry-propagation*", the inner products of line 2 are calculated. As soon as a MAC unit completes an addition of carry-propagated inner products for line 1, a new addition for line 2 is performed. The process continues for all lines of (18) and the result is available after steps. The complete DRAMM architecture is depicted in Fig. 8.

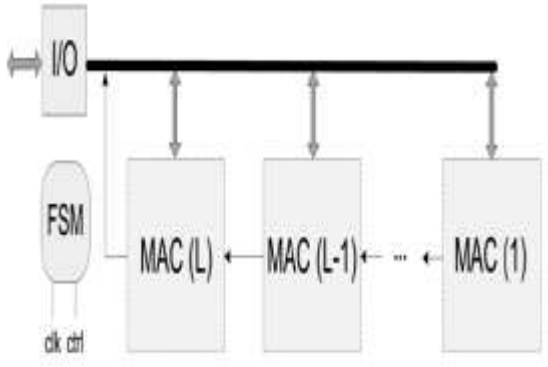

Fig. 8. The proposed DRAMM architecture.

VII. SIMULATION RESULTS

The simulations of the proposed design are carried out by using Verilog HDL in Xilinx tool. The RTL SCHEMATIC AND SIMULATION RESULTS OF FOUR MAC are shown in below figures.

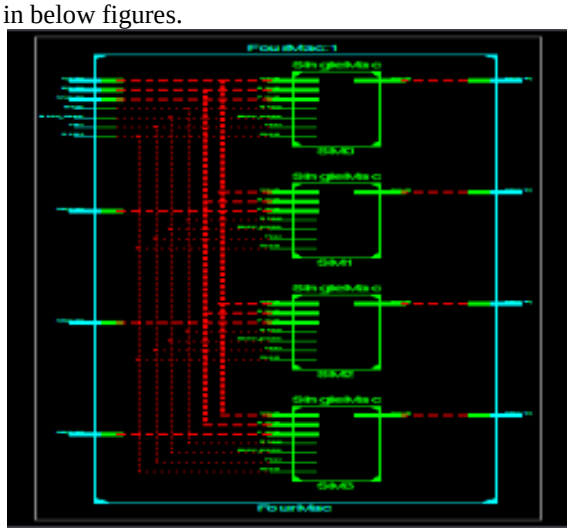

Fig.9. RTL schematic of proposed design

Fig.10. Simulation results of the proposed design

VIII. CONCLUSION

The mathematical framework and a flexible, dual-field, residue arithmetic architecture for Montgomery multiplication in GF(p) and GF(2^n) is developed and the necessary conditions for the system parameters (number of moduli channels, modulus word length) are derived. The proposed DRAMM architecture supports all operations of Montgomery multiplication in GF(p) and GF(2^n), residue-to-binary and binary-to-residue conversions, MRC for integers and polynomials, dual-field modular exponentiation and inversion, in the same hardware. Generic complexity and real performance comparisons with state-of-the-art works prove the potential of residue arithmetic exploitation in Montgomery multiplication.

REFERENCES

[1] I. Blake, G. Seroussi, and N.Smart, *Elliptic Curves in Cryptography*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[2] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curves Cryptography*. New York, NY, USA: Springer-Verlag & Hall/CRC, 2004.

[3] J.-P. Deschamps, *Hardware Implementation of Finite-Field Arithmetic*. New York, NY, USA: McGraw-Hill, 2009.

[4] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. New York, NY, USA: Cambridge Univ. Press, 1986.

[5] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, Feb. 1978.

[6] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower architecture for fast parallel Montgomery multiplication," in *EUROCRYPT' 00: Proc. 19th Int. Conf. Theory and Application of Cryptographic Techniques*, 2000, pp. 523–538.

[7] J.-C. Bajard and L. Imbert, "Brief contributions: A full RNS implementation of RSA," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 769–774, Jun. 2004.

[8] D. Schinianakis, A. Fournaris, H. Michail, A. Kakarountas, and T. Stouraitis, "An RNS implementation of an elliptic curve point multiplier,"
IEEE Trans. Circuits Syst. I, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[9] A. Halbutoğullari and Ç. K. Koç, "Parallel multiplication in using polynomial residue arithmetic," *Design, Codes and Cryptography*, vol. 20, no. 2, pp. 155–173, Jun. 2000.

[10] M. G. Parker and M. Benaissa, " multiplication using polynomial residue number systems," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 11, pp. 718–721, Nov. 1995.

[11] H. Nozaki, M. Motoyama, A. Shimbo, and S.-I. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery multiplication," in *Proc. 3rd Int.Workshop on Cryptographic Hardware and Embedded Systems (CHES '01)*, 2001.

[12] J.-C. Bajard, L. Imbert, and G. A. Jullien, "Parallel Montgomery multiplication in using Trinomial Residue Arithmetic," in *IEEE Symp. Computer Arithmetic*, 2005, vol. 0, pp. 164–171.

[13] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over ," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, 2010, pp. 48–64, Lecture Notes in Computer Science 6225.

[14] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Computer*, vol. 17, pp. 50–62, May 1988.

Author's Profile:

Chembeti Silpa received her B.Tech degree in Electronics and communication Engineering from Priyadarshini college of Engineering and Technology, Kanuparthipadu, Nellore District, affiliated to JNTU Anantapur. She is currently pursuing M.Tech VLSI in Audisankara college of Engineering and Technology, Gudur(Autonomous), SPSR Nellore (Dist), affiliated to JNTU Anantapur.

G..Mukesh received his M.Tech in VLSI from PBR VITS , Kavali, SPSR Nellore District. He has 3 years teaching experience. He is presently working as Assistant Professor in the department of ECE Audisankara College of Engineering and Technology,Gudur (Autonomous), Affiliated to JNTU, Anantpur.