International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

An Enhanced Residue Modular Multiplier for Cryptography

Vundela Sarada

M.Tech ,Department of ECE, JNTUA College of engineering, Anantapur, A.P.

Abstract: ***This paper presents an implementation of VLSI architecture for Dual Field Residue Arithmetic modular multiplier with less delay based on finite field arithmetic to support all public key cryptographic applications. A new method for incorporating Residue Number System (RNS) and Polynomial Residue Number system (PRNS) in modular multiplication is derived and then implemented VLSI Architecture for dual field residue arithmetic modular multiplier with less delay . This architecture supports the conversions, modular multiplication for polynomials and integers and modular exponentiation in same hardware. This architecture has a carry save adders (CSAs) and parallel prefix adders in MAC units to speed up the large integer arithmetic operations over GF(P) and GF(2^n),hence this reduces the delay up to 10 percent.***

Keywords: Finite field arithmetic, Residue number and Polynomial Residue number systems,modular arithmetic, parallel arithmetic and logic structures, and montgomery multiplication.

1. Introduction

Now a days, many of applications including cryptography, error correction coding, computer algebra, DSP, etc., depen - ds 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 is a prime. Special case of multiplications are formed by Cryptographic applications, since, for security reasons, they require large integer operands. Almost all public key cryptography, such as Elliptic Curves Cryptography (ECC) and RSA cryptography, employ modul-ar multiplication with very large numbers, so faster modul-ar multiplication has become an important cryptography issue.

For achieving satisfactory cryptosystem performance, Efficient field multiplication with large operands is crucial 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. The perfect approach to achieve this would be through parallelization of their operations.

The RNS/PRNS is a non-weighted number system which speeds up arithmetic operations by dividing them into smaller parallel operations, and they provide interesting low power architecture. Since the RNS/PRNS system is not a positional number system where each digit corresponds to a certain weight, it is hard to implement the operations of comparison and division. RNS/PRNS is one of the most popular techniques for reducing the power dissipation and the computation load in VLSI systems design. On the other hand, for RNS/PRNS implementations, the extra cost of in-put converters to translate numbers from a standard binary format into residues and output converters to translate from RNS/PRNS to binary representations are needed .A new methodology for embedding residue arithmetic in a dual field Montgomery modular multiplication algorithm for integers in and for polynomials in is presented in this paper. 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 in the same hardware.

2. Previous Work

GF(2^n) implementations has been progressed a lot in these days. PRNS incorporation in field multiplication based on a straightforward implementation of the Chinese Remainder Theorem (CRT) for polynomials is implemented in [1], requires large storage resources and many pre-computations. The multipliers perform multiplication in PRNS are proposed in [2], [4] but the result is then converted back to polynomial representation.

This limitation makes these methods inappropriate for cryptographic algorithms, since it require consecutive mul - tiplications. Finally, algorithm which employs trinomials for the modulus set and performs PRNS Montgomery multiplication has been proposed [3].But [3] has no reference to conversion methods and the trinomials requirement 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 world length, while multiplier is handled bit-by-bit [5]–[6]. These designs are optimized for certain word lengths and do not scale 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 [7] and [8].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 architecture were to be used in ECC, An architecture configured at bit-level overcomes this problem.

3. Residue Arithmetic

A. Residue number system :

There is a set of L pair-wise relative prime integers $A = (M_1, M_2, \dots\dots, M_L)$ in RNS and the range of the RNS is calculated as $A = \prod_{i=1}^{L} m_i$.Any integer $Z \in [0, A-1]$ has a unique representation that is $Z_A = (Z_1, Z_2, \dots\dots\dots, Z_L) = (\langle Z \rangle m1, \langle Z \rangle m2, \dots\dots\dots\dots\dots, \langle Z \rangle mL)$, where $< z >_{m_i} = z \bmod m_i$. If two integers a and b are in residue format then one can perform any operations in parallel

$a_A * b_B$

$$= \begin{pmatrix} < a_1 * b_1 >_{m_1}, < a_2 * b_2 >_{m_2}, \dots\dots\dots\dots\dots, \\ < a_L * b_L >_{m_L} \end{pmatrix} \quad (1)$$

Two techniques may be employed [10] for reconstruction or the integers from residues. They are,

1. Chinese remainder theorem (CRT), according to

$$z = \sum_{i=1}^{L} < z_i . A_i^{-1} >_{m_i} . A_i - \gamma A \quad (2)$$

Where $A_i = A/m_i$,A_i^{-1} is the inverse of $A_i \bmod m_i$, γ is an integer correction factor.

2. Mixed Radix conversion (MRC),the MRC of an integer Z with an RNS representation is given by

$$z = U_1 + w_2 U_2 + \cdots\dots\dots\dots + w_L U_L \quad (3)$$

Where $W_i = \prod_{j=1}^{i-1} m_j$ and U_is are computed according to

$U_1 = z_1$

$U_2 = < z_2 - z_1 >_{m_2}$

$U_3 = < z_3 - z_1 - W_2 U_2 >_{m_3}$

.

..

$U_L = < z_L - z_1 - W_2 U_2 - W_3 U_3 - \cdots - W_{L-1} U_{L-1} >_{m_L}$.

(4) Among above two methods the proposed architecture uses MRC ,as it avoids the problem of evaluating the correction factor γ of (2).

B. Polynomial residue number system:

Similar to RNS, a PRNS is defined through a set of , pair-wise relatively prime polynomials

$A = (m_1(x), m_2(x), \dots\dots\dots\dots\dots, m_L(x))$. We denote by $A(x) = \prod_{i=1}^{L} m_i(x)$ the dynamic range of the PRNS. In PRNS, every polynomial has a unique PRNS representation:

$z_A = (z_1, z_2, \dots\dots\dots, z_L)$ such as $z_i = z(i) \bmod m_i(x)$,

$i \in [1, L]$, denoted as $< z >_{m_i}$.In the rest of the paper, the notation "(x) " to denote polynomials shall be omitted, for simplicity. The notation z will be used interchangeably to denote either an integer or a polynomial, according to context according to the context.

In the PRNS representation all operations can be performed in parallel. Conversion from PRNS to weighted polynomial representation is identical to the MRC for integers. The only difference is that, the subtractions in () are substituted by polynomial additions.

4. Montgomery Multiplication

A. GF(P) arithmetic

Field elements GF(P) in are integers in[0 to P-1] and arithmetic is performed modulo P . 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:

In GF(2^n) arithmetic, field elements are polynomials are represented as vectors with dimension n ,relative to a given polynomial basis $(1, \alpha, \alpha^2, \dots\dots, \alpha^{n-1})$,where α is a root of an irreducible polynomial p of degree n over GF(2).

The addition of two polynomials a and b in GF(2^n) is performed by adding the their coefficients i.e., modulo 2.The multiplication of two polynomials is

$$c = a \,.\, b \bmod p$$

5. Conversions

Let us consider $A = (p_1, p_2, p_3 \dots\dots\dots\dots\dots, p_L)$ as base, this shall be used to analyze the Conversions to/from residue representations.

1. **Binary-to-Residue Conversion:** A radix- representa -tion of an integer z as an L- tuple $(z^{(L-1)}, \dots\dots\dots, z^{(0)})$ satisfies

$$z = \sum_{i=0}^{L-1} z^{(i)}\, 2^{ri} \quad (5)$$

where,$0 \le z^{(i)} \le 2^r - 1$. To compute z_A a method is devised ,the Multiply and Accumulate structure in DRAMM is implemented for this method. By applying the modulo p_j operation in (14) ,we obtain

$$< z >_{P_j} = \langle \sum_{i=0}^{L-1} z^{(i)} \; < 2^{ri} >_{P_j} \rangle_{P_j} \;, \forall\, j$$

$$\in [1, L] \quad (6)$$

If constants $\langle 2^{ri} \rangle_{P_j}$ are precomputed, 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 $lz(x) \in GF(2^n)$ can be written as

$$z = \sum_{i=0}^{L-1} z^{(i)}\, x^{ri} \quad (7)$$

Applying the modulo p_j operation in the above equation

$$< z >_{P_j} = \langle \sum_{i=0}^{L-1} z^{(i)} \; < x^{ri} >_{P_j} \rangle_{P_j} \;, \forall\, j \in [1, L] \quad (8)$$

which is a similar operation to operation for integers, if $\langle x^{ri} \rangle_{P_j}$ are pre-computed.

From the above analysis conversions in both fields can be unified into a common conversion method, if dual-field circuitry is employed.

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

can be rewritten as matrix notation. The inner products of a row 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^rdigit $z^{(i)}$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.

6. Hardware Implementation

1. Dual Field Addition and subtraction:

A.Dual field adder: Dual field adder is a full adder cell equipped with a field select signal (f_{sel}).The f_{sel} signal controls the operation mode, the same circuit would operate on both polynomials and integers as well.

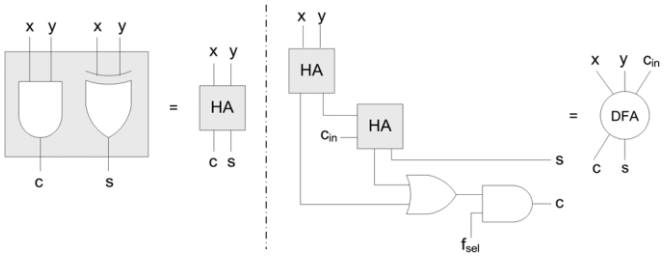

Figure 1: Dual-field full-adder cell (DFA).

B.Dual field Parallel Prefix Adder :
This is implemented by 3-level parallel prefix adder with four bit carry generator groups. The GAP modules generate the signals $p_i = x_i$ XOR y_i ,
$g_i = x_i$ AND y_i and $\alpha_i = x_i$ OR y_i .The AND gate along with the f_{sel} eliminates the carry for polynomials and stores the carry for integer arithmetic.

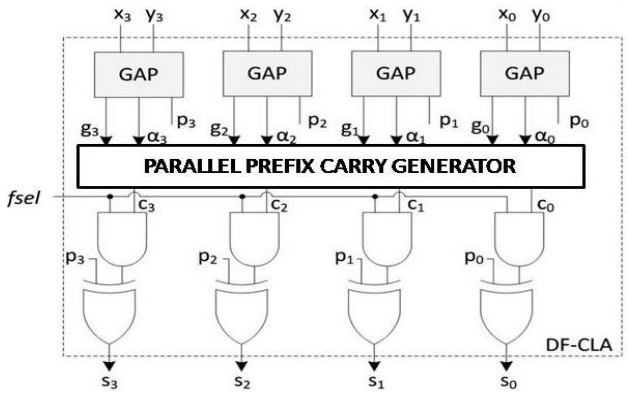

Figure 2: Dual-field PPA.

The parallel prefix generator generates all carries in parallel by using input carry only, hence Dual Field PPA gives fast operations on large number of bits.

C. **Dual-Field Modular/Normal Addition/Subtraction**
After some modifications of algorithms for modular addition/subtraction in GF(P) a dual-field modular adder/subtracter (DMAS) shown in Fig. 3 is implemented using Parallel prefix adder (PPA) adders.

When $f_{sel} = 0$, the circuit is operates in $GF(2^n)$ mode and the output is computed directly from the top adder which performs a $GF(2^n)$addition. When$f_{sel} = 1$, the circuit may operate either as a normal –bit adder /subtracter (conv-mode=0) or as a modular adder/ subtracter (conv-mode=1) .

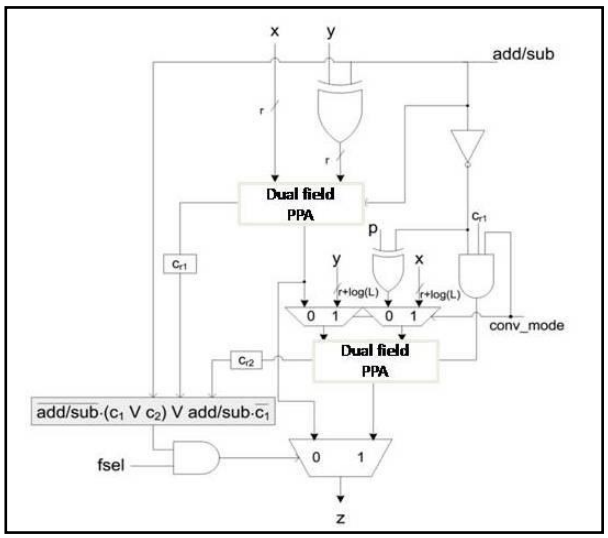

Figure 3: Dual-field modular/normal adder/subtracter (DMAS).

2. Dual-Field Multiplication:

A traditional parallel tree multiplier, which is suitable for high-speed arithmetic with little modification to support both fields, is implemented in this architecture. For both 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 a carry save adder tree(CSA tree) that sums the partial products must support both formats. In this multiplier the DFA cells eliminates the carry for $GF(2^n)$ mode and only XOR operations performed among partial products. This multiplier act as a traditional multiplier for GF(p) mode

.

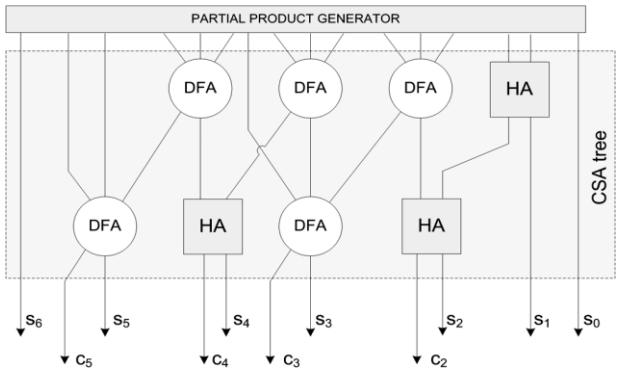

Figure 4: Dual-field multiplier (DM)

.

3. Dual-Field Modular Reduction:

After each multiplication, finally modular reduction by each RNS/PRNS modulus is required , for each multiplica-tion outcome, within each MAC unit. Several modular reduction strategies are employed for modular reduction ,this method is implemented 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.

Let us consider the 2r –bit product C that needs to be reduced modulo of an integer modulus P_i.By proper selection of the the P_i which is $2^r - \mu_i$,where the h-bit $\mu_i \ll 2^r$ modular reduction process can be simplified as

$$\langle C\rangle_{P_i} = \langle \overbrace{\sum_{i=0}^{r-1} c_i\, 2^i}^{E} + 2^r \overbrace{\sum_{i=0}^{r-1} c_{r+i}\, 2^i}^{F} \rangle_{P_i} = \overbrace{\langle E + 2^r F\rangle}^{P_i}$$
$$= \langle \textstyle\sum_{i=0}^{r-1} d_i\, 2^i + \mu_i \sum_{i=0}^{r-1} d_{r+i}\, 2^i \rangle_{P_i} \quad (9)$$

From (9), it is apparent that

$$\langle C\rangle_{P_i} = \begin{cases} \sum_{i=0}^{r-1} \beta_i\, 2^i, \beta < 2^r - \mu_i \\ \sum_{i=0}^{r} \beta_i\, 2^i + \mu_i, 2^r - \mu_i < \beta < 2^r \end{cases} \quad (10)$$

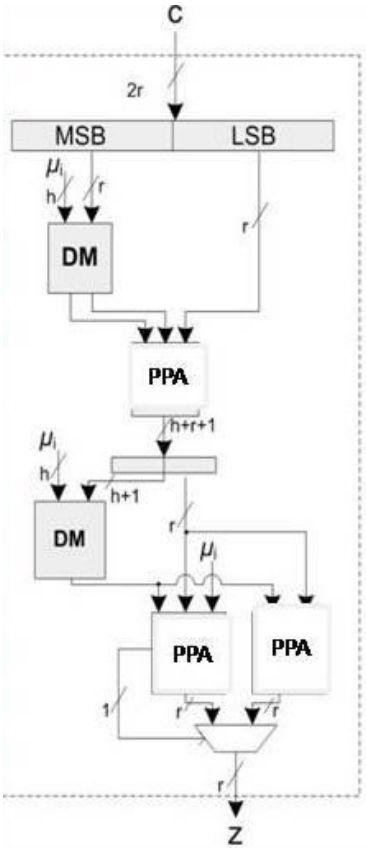

Figure 5: Dual-field modular reduction unit (DMR).

Similar to the integer case, a polynomialz(x) ∈ GF(2^n) canbe written as

$$z = \textstyle\sum_{i=0}^{L-1} z^{(i)}\, x^{ri} \quad (11)$$

Applying the modulo p_j operation in the above equation

$$< z >_{P_j} = \langle \textstyle\sum_{i=0}^{L-1} z^{(i)} \; < x^{ri} >_{P_j} \rangle_{P_j} \;, \forall\, j \in [1, L] \quad (12)$$

which is a similar operation to operation for integers, if $\langle x^{ri}\rangle_{P_j}$ are pre-computed.

From the above analysis conversions in both fields can be unified into a common conversion method, if dual-field circuitry is employed.

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

4. MAC UNIT

An implemented MAC unit in this paper is shown in fig 6. In this MAC unit operation is analyzed in three steps, corresponding to three phases of calculation it handles.

1. Binary to residue conversion.
2. Montgomery multiplication.
3. Residue to Binary conversion.

1) Binary-to-Residue Conversion:

Initially, r-bit words of the input operands, as shown by (9), are applied to each MAC unit and stored in RAM1 which is located at the top of Fig. 7. These words are the first input to the multiplier, along with the quantities $\langle 2^{ri}\rangle_{p_i,q_i}$, $\langle x^{ri}\rangle_{p_i,q_i}$.These quantities are stored in a ROM. Their multi-plication produces the inner products of (9) or (11) .These products are added recursively in the DMAS uni,result is stored via the bus in RAM1.For the second operand also the process is repeated and the result is stored in RAM2, so that at the completion of the conversion , each MAC unit holds the residue digits of the two operands in the two RAMs. It conversion requires L steps to be executed.

2) Montgomery Multiplication:

The first step of the DRAMM is a modular multiplication of the residue digits of the operands. After completion of the residue to binary conversion ,these residue digits are imme-diately available by the two RAMs, a modular multiplicati -on is executed and the result in R_1 is stored in RAM1 for base B and RAM2 for base A . Step2 of DRAMM is a multiplication of the previous result with a constant provided by the ROM. According to the requirement the, all MAC units are updated through the bus with the corresponding RNS digits of all other MACs and a DBC process is initiated. Here, the multiplication is done in parallel i.e., the operations in MAC are split two parts modular multiplication and addition with the result of previous MAC.

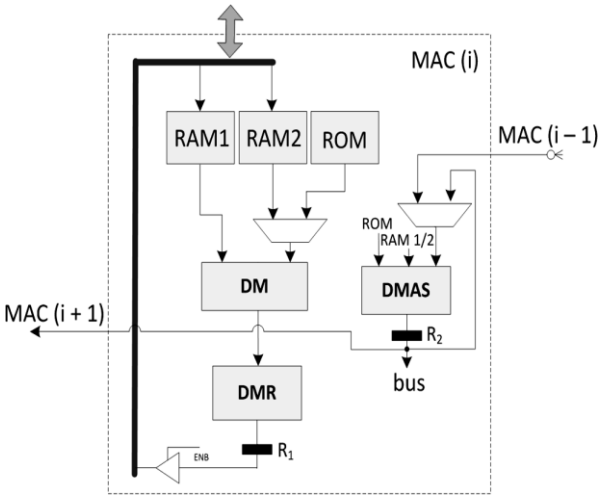

Figure 6: MAC unit

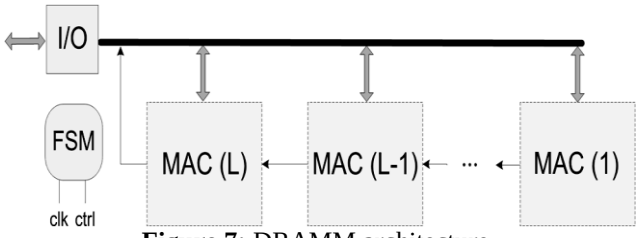

Figure 7: DRAMM architecture.

The remaining multiplications, additions, and the final DBC operation required by the DRAMM algorithm are computed in the same multiply-accumulate manner and the final residue Montgomery product can be either driven to the I/O interface, or it can be reused by the MAC units to convert the result to binary format.

3) Residue-to-Binary Conversion:

Residue-to-binary conversion is implemented based on the Mixed Radix Con-version method. The inner products are generated by the multiplier only. Each product is calculated in parallel in each MAC unit and a "carry-propagation" from MAC(1) to MAC(L) is performed to add all inner products. When summation finishes the first digit $z^{(0)}$ of the result is produced in MAC(L).. The inner products of line 2 are calculated in parallel with this "carry-propagation". A new addition for line 2 is performed immediately after an addition of carry-propagated inner products for line 1 is completed by the MAC unit. The process continues for all lines of (4) and the result is available after steps. The complete DRAMM architecture is depicted in Fig. 7

7. Performance and Comparisons

Dual field: The DRAMM architecture operates in GF(P) arithmetic when F_{sel} is 1 and for F_{sel} is 0 it is act as modular multiplier for $GF(2^n)$ arithmetic.

Memory requirement : The implanted architecture requires maximum (2L-1) r-bit RAM 1/2 per MAC UNIT, hence total a L(4L-2) r-bit RAM is required. For binary to residue conversion $4L^2$ r − bits , for DRAMM 6Lr − bit and for residue to binary L(L − 1) r-bit memory required.

Comparisons

- The implemented architecture introduces the Dual field RNS montgomery Multiplier, which is not supported by existing RNS solutions [2],[12] and [10].
- It further reduces the number of Modular multiplications for the base conversion and the RNS to Binary conversion because it uses the simplified MRC instead of CRT [41].In this architecture simplified version of MRC requires L − 2 multiplications to implement (4),while [39] requires L(L − 1)/2 for same conversion.

Table 1: No. of Modular Multiplications

	Input conversions	Output conversions	Others
Present work	L^2	$\frac{L(L-1)}{2}$	5L
[3]	L^2	L(2L + 1)	5L
[11]	L^2	L(L + 1)	2L

- Parallel prefix adder has a less Fan out ,hence this is the fast adder when comparing with other adders. MAC unit uses a parallel prefix adder, which plays the important role in MAC unit. This leads to less path delay comparing with existing system. The results obtained from three works are shown below table 2.

Tabel 2: Comparsions Between Three Works

	Delay(ns)	Supported fields
Implemented method	18.214	Dual field (polynomials and integers)
[12]	20.732	Dual field (polynomials and integers)
[10]	22.137	Single (only integers)

The area of the architecture is over head for this work when comparing with other implementations. However , while considering non RNS implementations, the use full properties like Dual field design, fault detection and, more recently, immunity against hardware fault attacks should be taken into account [9].

.

8. Conclusion

An Efficient high speed RNS modular multiplier implement –ed in this paper ,that operates in both GF(p) and $GF(2^n)$ arithmetic fields and necessary conditions for the system parameters are mentioned. The DRAMM architecture supports all operations of montgomery multiplication, residue-to-binary conversion and binary-to-residue conver - sion , MRC for polynomials and integers and ,modular exponentiation in same hard ware.

The MAC units in DRAMM architecture reduces the delay, hence this is suited for high speed applications like all types of public key cryptography and DSP ete.,

References

[1] A. Halbutoğullari and Ç. K. Koç, "Parallel multiplication inusing polynomial residue arithmetic," Design, Codes and Cryptography,vol. 20, no. 2, pp. 155–173, Jun. 2000.
[2] 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.
[3] H. Nozaki, M. Motoyama, A. Shimbo, and S.-I. Kawamura, "Implementationof RSA algorithm based on RNS Montgomery multiplication,"in Proc. 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES '01), 2001.
[4] D. Schinianakis,A.Kakarountas, T. Stouraitis, and A. Skavantzos, "Ellipticcurve point multiplication in using Polynomial Residue Arithmetic," in Proc. IEEE Int. Electronics, Circuits, and Systems(ICECS 2009), 2009, pp. 980–983.
[5] C. McIvor, M. McLoone, and J. McCanny, "Modified Montgomery modular multiplication and RSA exponentialtion techniques," IEEProc.—Computers and Digit-al Techniques, vol. 151, no. 6, pp.402–408, Nov. 2004.

[6] M. Huang, K. Gaj, S. Kwon, and T. El-Ghazawi, "An ptimized hardware architecture for the Montgomery multiplication algorithm," inProc. Practice and Theory in Public Key Cryptography, PKC'08, 2008,pp. 214–228.
[7] W. Freking and K. Parhi, "Performance-scalable array architectures for modular multiplication," in Proc. IEEE Int. Conf. Application-SpecificSystems, Architectures, and Processors, 2000, pp. 149–160.
[8] M.-D. Shieh and W.-C. Lin, "Word-based Montgomery Modular multiplication algorithm for low-latency scalable architectures," IEEE Trans. Comput., vol. 59, no. 8, pp. 1145–1151, aug. 2010.
[9] D. Schinianakis and T. Stouraitis, "Harware-fault attack handling inRNS-based Montgomery multipliers," in Proc. IEEE Int. Symp. Circuits and Systems, 2013.
[10] Y. Tong-jie, D. Zi-bin, Y. Xiao-Hui, and Z. Qian-jin, "An improved RNS Montgomery modular multiplier," in Proc. 2010 Int. Conf. Computer Application and System Modeling (ICCASM), 2010, vol. 10, pp.V10-144–V10-147.
[11] F. Gandino, F. Lamberti, G. Paravati, J. Bajard, and P. Montuschi, "An algorithmic and architectural study on Montgomery exponentiation in RNS," IEEE Trans. Comput., vol. 61, no. 8, pp. 1071–1083, 2012.
[12] Dimitrios Schinianakis and Thanos Stouraitis, "Multifunction Residue Architectures for Cryptogra - phy", IEEE transactions on circuits and systems—i: regular papers, vol. 61, no. 4, april 2014