POLYNOMIAL RESIDUE NUMBER SYSTEM GF(2^M) MULTIPLIER USING TRINOMIALS

Junfeng Chu, Mohammed Benaissa

Department of Electronic and Electrical Engineering, the University of Sheffield

Email: j.chu@sheffield.ac.uk, m.benaissa@sheffield.ac.uk

ABSTRACT

This paper introduces a new approach for implementing GF(2^m) multiplication using Polynomial Residue Number Systems (PRNS). Irreducible trinomials are selected as the generating polynomials for the PRNS channels to enable conversion to-and-from PRNS to be implemented using simple XOR networks. A novel approach for modular reduction over GF(2^m) is also presented for the PRNS architecture to achieve better performance.

1. INTRODUCTION

Many researchers have been encouraged by the escalating use of Galois fields GF(2^m) arithmetic in digital signal processing, cryptography, coding theory and computer algebra to investigate different architectures and novel algorithms to advance Galois field circuits.

In the Polynomial Residue Number System (PRNS), each channel is generated by a polynomial instead of a prime number as in the typical RNS. The Chinese Remainder Theorem (CRT), which is valid in RNS, can also be applied to PRNS [3]. Two main advantages of using RNS architectures are it uses less time and power consumption relative to traditional systems due to smaller operands and simpler circuits [19]. PRNS has also been used to implement fault tolerance in DSP and communication systems [12].

Due to the nature of independence between RNS channels and scope for randomisation, RNS architectures have also been advocated for improving side-channel resistance in cryptosystems [4]. For example, the same data can be represented and processed differently by using different PRNS sets, which can be used as an attractive randomization generating method to against Differential Power Analysis (DPA). Also, thanks to its parallelism, distributed architecture and data independency between channels, it is quite useful to against Electromagnetic Attacks. In addition, PRNS is also capable of tolerance faults by using extended bases, so that it can also be applied to countermeasure Fault Attacks.

PRNS shares most of the attractive properties with RNS; it has been adapted to build fast FIR filters in [1] [2] and proposed for computing large polynomial products in [13]. In [24], PRNS has been chosen to simplify GF(p^m) computations and in [7], PRNS has been proposed to achieve parallel multiplication over binary fields. In a recent paper [23], we reported the first hardware implementation of a PRNS multiplier over binary fields and its application in cryptography.

In this paper, the work in [23] is further improved by adopting trinomial irreducible polynomials for the PRNS channels to simplify the modular reduction and conversion operations thereby resulting in significant improvement in speed and area. An FPGA implementation is presented which shows a 47 times' speed*area product improvement over [23] for a Galois Field GF(2^{163}) multiplier.

The following paper is structured as: In Section 2, background knowledge is given in terms of Galois fields arithmetic, PRNS representation of GF(2^m) elements, the polynomial residue arithmetic (PRA), according to which the multiplier is built and the proposed PRNS modular reduction method is introduced. In Section 3, the proposed PRNS GF(2^m) multiplier is introduced, including the selection of channel generating polynomials, simplified PRNS conversion and reduction. In Section 4, FPGA implementations synthesis results are given and comparisons with previous work are made. The paper is concluded in Section 5.

2. BACKGROUND

2.1 Galois field arithmetic

In this paper, polynomial basis representation is used (other representations may be found in [5]). A field element is written as:

$a(x)=a_{m-1}x^{m-1}+a_{m-2}x^{m-2}\ldots a_2x^2+a_1x+a_0$, where $a_i=1$ or 0

or , in binary vector format as: $(a_{m-1}a_{m-2}\ldots a_2a_1a_0)$.

The modular 2 additions and subtractions can be implemented using bitwise XOR, so the channel modular adder and subtracter circuits can be much easier implemented compared with normal RNS [6].

The multiplication is demonstrated as follows:

Let A, B $\in$ GF(2^m), then their product C=AB mod f(x), where f(x) is the field generating polynomial. f(x) is used to perform degree reduction to ensure that C is also in GF(2^m) and the multiplication is closed [19].

The Galois field arithmetic is also adopted in each channel of the proposed PRNS architectures.

2.2 PRNS representation of GF(2^m) elements

A list of irreducible polynomials over binary field is selected as the generating polynomials for PRNS channels. They are written as: $m_1, m_2\ldots m_N$, where N is the number of channels.

The degree of each m_i is d. To cover the whole dynamic range of a GF(2^m) multiplication, the degree *D* of the product polynomial $M(x)=\prod_1^N m_i(x)$ should be no smaller than *2m* ($d \times N \geq 2m$).

A polynomial basis field element $p(x)$ can be represented in PRNS format using the remainders, when the $d \times N \geq 2m$ equation is satisfied:

$$\vec{p} = (p_1, p_2 \dots p_N)$$

Where $\vec{p} = p(x)\ mod\ m_i(x)$ for i = 1, 2, …, N. [7]

2.3 Polynomial Residue Arithmetic (PRA)

PRA is very similar to RNS arithmetic. In furthermore, because addition and subtraction operations are performed by bitwise XOR in GF(2), it does not encounter overflow problems. In another word, modular reduction is not necessary for addition and subtraction operations. However, the modular reduction is still needed for multiplications to ensure all operations are closed. The main arithmetic operations in PRA can be performed as following:

$$A \pm B = (\langle a_1\ XOR\ b_1\rangle_{m1}, \dots, \langle a_N\ XOR\ b_N\rangle_{m_N})$$

$$A \times B = (\langle a_1 \times\ b_1\rangle_{m1}, \dots, \langle a_N\ \times\ b_N\rangle_{m_N})$$

The reason that the magnitude determination cannot be performed directly from RNS [8], and in order to prevent overflow, a conversion back to polynomial representation is necessary before performing the original GF(2^m) modular reduction.

In this work, the conversion from PRNS format to weighted polynomial representation is based on the extension of the CRT to polynomials [3] [20]. Single Radix Conversion (SRC) algorithm is used to perform the conversion. It is described as [7]:

$$p(x) = \sum_{i=1}^{N}(p_i \cdot I_i\ mod\ m_i) \cdot M_i \qquad (1)$$

Where $M_i(x) = \frac{M(x)}{m_i(x)} = m_1 \dots \cdot m_{i-1} \cdot m_{i+1} \dots \cdot m_N$

$$I_i = M_i^{-1}(mod\ m_i)$$

2.4 Modular Reduction over GF(2^m) using PRNS Architecture

Galois Field arithmetic required the multiplication to be closed, to prevent overflow, modular reduction is performed following the calculation of the intermediate product using the field generating polynomial *f(x)*, whose degree is equal to *m*.

Focusing on polynomial basis multiplications over *GF(2^m)*, several approaches have been reported for the modular reduction, such as "shift-and-add" algorithms [14][15][16], look-up-table (LUT) based algorithms [17][18], Itoh-Tsujii algorithm based reduction method [10], etc.

In our proposed multiplier architecture, a novel modular reduction method for PRNS is derived as follows:

Over *GF(2^m)*, which is generated by *f(x)*, the multiplication can be expressed as:

$$pdt(x) = a(x) \cdot b(x) \bmod f(x) \qquad (2)$$

The intermediate product, $a(x) \cdot b(x)$ can be expressed in polynomial forms as

$c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m+ c_{m-1}x^{m-1}+\dots+ c_1x+c_0$

So, equation (2) can be written as

$(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m+ c_{m-1}x^{m-1}+\dots+ c_1x+c_0)$ mod *f(x)*=$(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ mod *f(x)* + $(c_{m-1}x^{m-1}+$ $\dots+ c_1x+c_0)$ mod *f(x)* (3)

Since *f(x)* is a polynomial with degree *m,*

(3)= $(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ mod *f(x)* + $(c_{m-1}x^{m-1}+\dots+ c_1x+c_0)$ (4)

In *GF(2)*, 1+1=0, so $(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ can be added to (4) twice without changing the equation's value.

(4)= $(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ mod *f(x)* + $(c_{m-1}x^{m-1}+\dots+ c_1x+c_0)$ + $(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$+$(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ (5)

Rearranging (5):

$pdt(x)$=$(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ mod *f(x)* + $(c_{2m-2}x^{2m-2}+ c_{2m-3}x^{2m-3}+...+ c_mx^m)$ + $(c_{2m-2}x^{2m-2}+c_{2m-3}x^{2m-3}+...+ c_mx^m + c_{m-1}x^{m-1}+\dots+ c_1x+c_0)$

Rewrite it in binary vector format:

$pdt(x)$=$(c_{2m-2},c_{2m-3},\dots c_m,0,\dots 0)$ mod *f(x)* + $(c_{2m-2},c_{2m-3},$ $...c_m0,\dots 0) + (c_{2m-2}, c_{2m-3},...c_m,c_{m-1},\dots c_1,c_0)$ (6)

After mod mod *f(x)* operation, the result can be expressed as $(c'_{m-1},\dots c'_1,c'_0)$, so

$pdt(x)$ =$(c'_{m-1},\dots c'_1,c'_0)$ + $(c_{2m-2},c_{2m-3},...c_m0,\dots 0)$ + $(c_{2m-2},$ $c_{2m-3},...c_m,c_{m-1},\dots c_1,c_0)$

=$(c_{2m-2},c_{2m-3},...c_m,c'_{m-1},\dots c'_1,c'_0)$ + $(c_{2m-2},c_{2m-3},...c_m,$ $c_{m-1},\dots c_1,c_0)$ (7)

Both components in (7) can be expressed using PRNS representation. The modular reduction over *GF(2^m)* can be finally performed by PRNS addition.

To implement the modular reduction, it is required to partially convert the intermediate product's most significant m-1 bits from PRNS to normal polynomial representation to generate $(c_{2m-2},c_{2m-3},...c_m,0,\dots 0)$ component. And a modular reduction operation using *f(x)* to get $(c'_{m-1},\dots c'_1,c'_0)$ component. In addition, a to-PRNS converter is needed to convert the $(c_{2m-2},c_{2m-3},...c_m,c'_{m-1},\dots c'_1,c'_0)$ component to PRNS format to perform the final PRNS addition.

Due to the fact that partial conversion and a PRNS architecture are adopted, this design is effective in preventing leaking information while performing the conversion and modular reduction.

The detailed implementation information is described in part 3.

3. THE PROPOSED PRNS GF(2^M) MULTIPLIER

Consider the example application of the proposed multiplier in the context of public key cryptosystems, in particular elliptic curve cryptography (ECC) where the required large operands impose many design challenges. The curve K-163 presented in Fips-186 [11] is chosen as a standardized curve for ECC over the binary field. It uses the field generating polynomial $f(x) = x^{163}+x^7 + x^6 + x^3 + 1$ over $GF(2^{163})$. A $GF(2^{163})$ multiplier is constructed to demonstrate the proposed PRNS architecture.

To cover the whole dynamic range, four 84-degree irreducible trinomials are selected as the PRNS channels. This satisfies the dynamic range $d \times N \geq 2m$ equation, where d=84, N=4, m=163. There are two main reasons why trinomials are chosen. Firstly, in Galois field multiplications, using trinomials achieves the lowest hardware complexity in modular reduction, especially when the trinomials are of the form $x^m + x^k + 1$, where $k \leq \frac{m}{2}$ [21]. This property of trinomials is also attractive when building the PRNS converter. Secondly, in equation (1), it can be seen that M_i, which is a constant value in the determined PRNS, is the product of several channel generating polynomials. To make the multiplying by M_i operation simple, it is required that M_i should have a smaller number of '1's and this can be best achieved by using trinomials, because they are the irreducible polynomials with the fewest '1's over binary field.

However, trade-offs are required between the channel length and the channel number for an optimised design. To cover the same dynamic range, shorter channel lengths require more channels, which may lead to consuming more hardware resources and to more complex converter design. In addition, irreducible trinomials only appear in certain degrees and the number of trinomials, which satisfy $k \leq \frac{m}{2}$, is even smaller. That is the reason why four 84-degree irreducible trinomials are selected for this design.

3.1 Channel Multiplier Design

There are several approaches for implementing the PRNS channel multipliers, such as bit-serial architecture [14] [15] [16], bit-parallel architecture [21] [22] and digital serial/parallel architecture [9]. Since the selected field length for PRNS channels is quite large, which is degree of 84, the bit-serial architecture is adopted here in order to achieve the lowest hardware complexity. Figure 1 illustrates an MSB-first bit-serial multiplier over $GF(2^4)$ generated by trinomial x^4+x+1. In addition, such multiplier is not only suitable for performing channel multiplication, but also for calculating $(p_i \cdot I_i \ mod \ m_i)$ in equation (1).

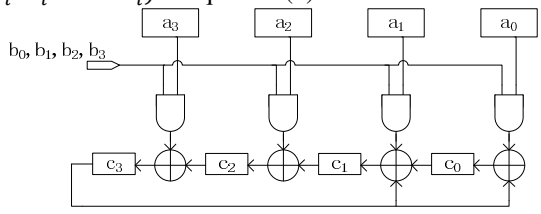

Figure 1 Bit-serial Multiplier over $GF(2^4)$.

3.2 Multiplying by M_i Operation

Consider the following example where M_1 is written in polynomial form as:

$$x^{252}+x^{181}+x^{179}+x^{177}+x^{168}+x^{108}+x^{106}+x^{104}+x^{84}+x^{33}+x^{24}+x^{22}+x^{20}+x^{13}+x^{11}+x^9+1 \quad (8)$$

(It is a product of all channels generating polynomials except the one for Channel 1. Those polynomials are $1+X^9+X^{84}$, $1+X^{11}+X^{84}$, $1+X^{13}+X^{84}$)

According to equation (6), since a partial conversion is performed to calculate the most significant 162 bits of the intermediate product, the component with the degree smaller than 84 can be ignored in (8), because they do not contribute to the final partial conversion result. So multiplying M_1 can be done by multiplying by the following polynomial instead:

$$x^{252}+x^{181}+x^{179}+x^{177}+x^{168}+x^{108}+x^{106}+x^{104}+x^{84}$$

It is assumed that the multiplicand is a(x) which is in a PB representation. The multiplication is as follows:

$$a(x) \bullet (x^{252}+x^{181}+x^{179}+x^{177}+x^{168}+x^{108}+x^{106}+x^{104}+x^{84})$$

$$= a(x) \bullet x^{252} + a(x) \bullet x^{181} + a(x) \bullet x^{179} + a(x) \bullet x^{177} + a(x) \bullet x^{168} + a(x) \bullet x^{108} + a(x) \bullet x^{106} + a(x) \bullet x^{104} + a(x) \bullet x^{84} \quad (9)$$

It is simple to implement (9) by using an XOR network and routing a(x) in the correct position as illustrated by Figure 2.

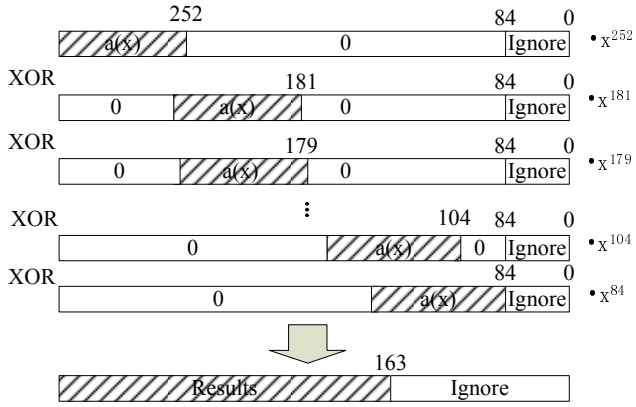

Figure 2 Multiplying by M_1

3.3 $GF(2^m)$ Modular Reduction and to_PRNS Converter

A bit-parallel modular reduction method is adopted to perform the field modular reduction to reduce (c_{2m-2},c_{2m-3},...c_m,0,…0) components to a degree smaller than m, the result is written in binary vector format as (c'_{m-1},…c'_1,c'_0), which implements the calculation of the first components in equation (7).

The to-PRNS converter is implemented by simple modular reduction using the selected trinomials. The detailed implementation approach can be found in [21].

Both modular reduction and conversion are implemented using a simple XOR network.

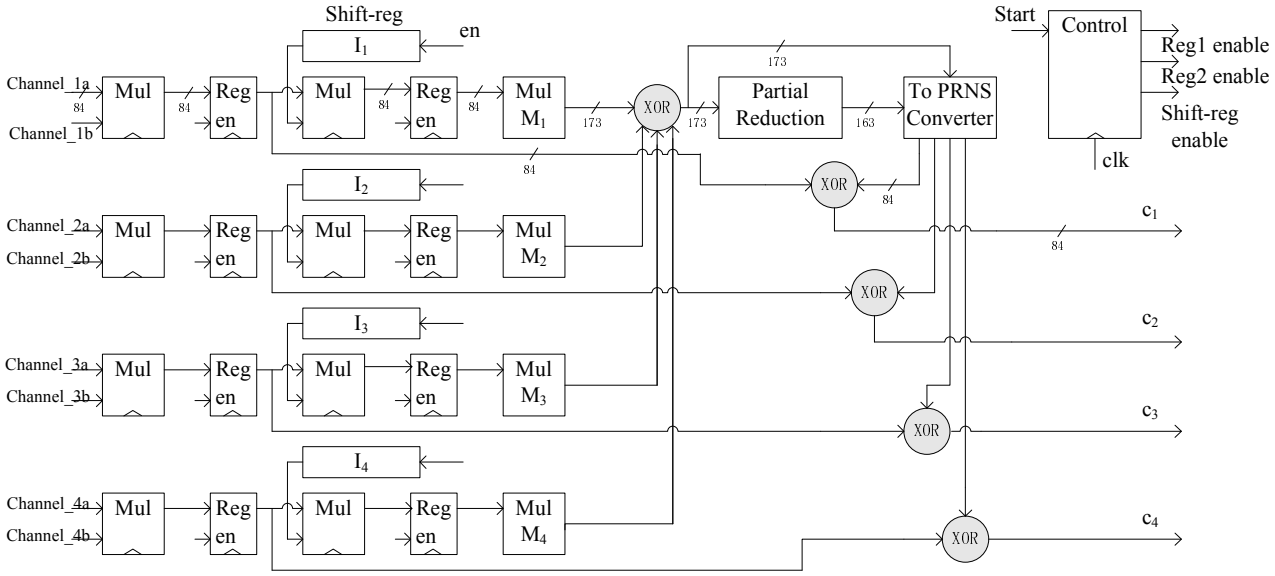

Figure 3 Architecture of PRNS Multiplier

3.4 Architecture of the Proposed PRNS Multiplier

Figure 3 shows the full architecture of the proposed PRNS multiplier over $GF(2^{163})$. It is assumed that the input and the output of the multiplier are all in PRNS representation.

The $GF(2^{84})$ multipliers on the left hand side perform PRNS channel modular multiplication, the one on the right performs part of SRC algorithm which is $(p_i \cdot I_i \ mod \ m_i)$ operation in equation (1). I_i are pre-calculated and stored into the shift registers. When there is a valid signal on Shift-reg-enable, the shift register starts to forward I_i bit by bit to the second $GF(2^{84})$ multiplier acting as a bit-serial input.

The module Mul M_i, Partial Reduction and To PRNS Converter are constructed using pure XORs. According to equation (7), the output of To PRNS Converter is the PRNS representation of $(c_{2m-2},c_{2m-3},...c_m,c'_{m-1},\ldots c'_1,c'_0)$, the output of the first registers is the PRNS representation of $(c_{2m-2},c_{2m-3},...c_m,c_{m-1},\ldots c_1,c_0)$. The final product after $GF(2^{163})$ modular reduction is generated by a PRNS addition operation, which is implemented as bitwise XORs.

It takes 168 clock cycles to finish a multiplication operation. This includes 83 clock cycles performing channel multiplication, another 83 clock cycles performing multiplication by I_i and 2 clock cycles on the data propagating through two registers.

As it can be seen from Figure 3, all channels are separate, similar and their operations are performed in parallel hence offering an inherent mechanism for masking, randomisation and fault tolerance which could help improve protection against any potential side channel leakage or analysis [4].

4. HARDWARE RESULTS AND COMPARISONS

Xilinx Spartan 3-3s1500lfg320-4 FPGA is used for synthesis and implementation to enable a fair comparison.

Table 1 shows the synthesis results of the proposed multipliers compared with our previous work in [23] which to our knowledge is the only other reported implementation of a PRNS multiplier over binary fields.

From the results, the work in this paper shows significant improvements both in hardware consumption and speed compared with our previous work.

The figures indicate that this work consumes half and one third slices compared with the channel-serial and channel-parallel architecture respectively. The highest operating frequency is improved by over 30 times. The total delay is improved by 25 times over the channel-serial architecture and by 17 times over the channel-parallel architecture. This figures also show a 47 times' Time-Area Product improvement over the channel-serial architecture, a 57 times' improvement over the channel-parallel architecture.

	Channel-serial [23]	Channel-Parallel[23]	This work
FF	1010	1350	1691
LUT	5274	8675	2588
Slices	2752	4625	1429
Frequency (MHz)	5.179	5.119	164.015
Cycles	130	93	168
Delay (ms)	$25.1*10^{-3}$	$18.2*10^{-3}$	$1.024*10^{-3}$
Time-Area Product (Slices*second)	$69*10^{-3}$	$84*10^{-3}$	$1.463*10^{-3}$

Table 1 Synthesis Results

As mentioned in Section 3, using trinomials simplifies the modular reduction and To PRNS conversion operations. Furthermore, together with the partial conversion method, using trinomials breaks down the bottleneck in multiplying by M_i operation; hence it achieves higher speed and less resource.

Work implemented by [28]	Platform	Slices	Delay
Proposed by [25]	Virtex 2	5307	12.56µs
Proposed by [26]	Virtex 2	5409	13.37µs
Proposed by [27]	Virtex 2	5840	14.73µs
This work	Spartan 3	1429	1.024µs

Table 2 Comparisons to non-PRNS works

Table 2 shows the comparisons with some other 163 bits parallel GF(2^m) multipliers. The figures indicate that this work shows great improvements both in area and speed than the ones in the table. Though this multiplier is neither forcing on high speed nor on low area, it provides the potential to countermeasure side-channel-attacks as well as a feasible option to implement parallel architecture.

5. CONCLUSIONS

In this paper, irreducible trinomials are adopted in the design and implementation of a novel PRNS multiplication architecture over *GF(2^m)* . This architecture is shown to exhibit improved overall hardware performance in both speed and area as confirmed by FPGA results. Such multipliers are particularly useful in fault tolerant DSP application [12] and side-channel resistant cryptographic implementations [4].

REFERENCES

[1] S. Alexander, A.Jorge and G. Suhas. PRNS Approach To Fast FIR Filtering. IEEE Proceedings, Southeastcon, 1990.

[2] M. Abdallah,A. Skavantzos. The Multipolynomial Channel Polynomial Residue Arithmetic System. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, II: ANALOG AND DIGITAL SIGNAL PROCESSING, VOL. 46, NO. 2, FEBRUARY 1999.

[3] M.C. Yang and J.L.Wu. A New Interpretation of "Polynomial Residue Number System". IEEE Transactions on Signal Processing. Vol.42,No.8,August 1994.

[4] M. Ciet, M. Neve1, E. Peeters1, J. Quisquater., "Parallel FPGA implementation of RSA with Residue Number Systems: can side-channel threats be avoided?" IEEE Midwest Symposium on Circuits and Systems, Dec. 2003, Egypt.

[5] L. Batina, S.B. Őrs, B. Preneel, J. Vandewalle. "Hardware architectures for public key cryptography," Elsevier Integration, the VLSI Journal, special issue on Embedded Cryptographic Hardware 34(1-2), pp. 1-64, 2003.

[6] W.N.Chelton and M.Benaissa, Fast Elliptic Curve Cryptography on FPGA, IEEE Transactions on VLSI Systems, Vol.16,No.2, February 2008.

[7] A. Halbutoğullari, "Parallel multiplication in GF(2k) using polynomial residue arithmetic," Designs, Codes and Cryptography, Vol.20 No.2, pp. 155–173, June 2000.

[8] Riyaz Aziz Patel, "A study and implementation of parallel-prefix modular adder architechtures for the Residue Number System," Ph.D. dissertation, The University of Sheffield, Sheffield, The UK, 2006.

[9] M.Hütter, J.Großschädl and G.A.Kamendje, A Versatile and Scable Digit-Serial/Parallel Multiplier Architecture for Finite Field GF(2^m), The International Conference on Information Technology: Computers and Communications,2003.

[10] Bharathwaj, S.V., Narasimhan, K.L., "An alternate approach to modular multiplication for finite fields [GF (2m)] using Itoh Tsujii algorithm," IEEE-NEWCAS Conference, pp.103–105, June 2005.

[11] FIPS 186-2, Digital Signature Standard,2000.

[12] M.G.Parker and M.Benaissa, Fault-Tolerant Linear Convolution using Residue Number Systems. Proc of ISCAS '94, London, Vol 2, pp 441-445, May 1994.

[13] A.Skavantzos, N. Mitash.Computing Large Polynomial Products using Modular Arithmetic. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-11: ANALOG AND DIGITAL SIGNAL PROCESSING, VOL. 39, NO. 4, APRIL 1992.

[14] Y. Han, P.-C. Leong, P.-C. Tan, and J. Zhang. Fast algorithms for elliptic curve cryptosystems over binary finite field. In Advances in Cryptology — ASIACRYPT '99, LNCS 1716, pp. 75–85. Springer Verlag, 1999.

[15] M. A. Hasan. Look-up table-based large finite field multiplication in memory constrained cryptosystems. IEEE Transactions on Computers, 49(7):749–758, July 2000.

[16] J. C. L´opez Hern´andez and R. Dahab. High-speed software multiplication in IF2m. In Progress in Cryptology — INDOCRYPT 2000, LNCS 1977 pp. 203–212. Springer Verlag, 2000.

[17] C. K. Koc¸ and T. Acar. Montgomery multiplication in GF(2k). Designs, Codes and Cryptography, 14(1):57–69, Apr. 1998.

[18] N. P. Smart. A comparison of different finite fields for elliptic curve cryptosystems. Computers and Mathematics with Applications, 42(1-2):91–100, July 2001.

[19] J. Chu, "Public Key Cryptography using Residue Number Systems on FPGA" MSc dissertation, The University of Sheffield, Sheffield, UK, 2007.

[20] D. E. Knuth. The Art of Computer Programming, Volume 2, Seminumerical Algorithms. Reading,MA: Addison-Wesley, Third edition, 1998.

[21] H. Wu. Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis. IEEE Transactions on Computers, Vol.51, No.7, July 2002.

[22] Mastrovito Edoardo, "VLSI Architectures for Computations in Galois Fields," Ph.D. dissertation, Linkoeping University, Linkoeping, Sweden, 1991.

[23] J. Chu, M.Benaissa, GF(2m) Multiplier using Polynomial Residue Number System. IEEE APCCAS 2008, 30 Nov-4 Dec, Macao, China.

[24] M.G.Parker and M.Benaissa, GF(p^m) Multiplication Using Polynomial Residue Number Systems. IEEE Trans on Circuits and Systems II, Vol 42, No 11, pp 718-721, Nov 1995.

[25] N. S. Chang, C. H. Kim, Y. H. Park, and J. Lim. A Non-Redundant and Efficient Architecture for Karatsuba-Ofman Algorithm. In Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings, volume 3650 of Lecture Notes in Computer Science, pages 288–299. Springer, 2005

[26] S. Erdem and C¸. K. Koç. A Less Recursive Variant of Karatsuba-Ofman Algorithm for Multiplying Operands of Size a Power of Two. In 16th IEEE Symposium on Computer Arithmetic (Arith-16 2003), 15-18 June 2003, Santiago de Compostela, Spain, pages 28–35. IEEE Computer Society, 2003.

[27] F. Rodr´ıguez-Henr´ıquez and C.K. KoC¸. Parallel Multipliers Based on Special Irreducible Pentanomials. IEEE Trans. Computers, 52(12):1535–1542, 2003.

[28] V. Serrano-Hern´andez and F. Rodr´ıguez-Henr´ıquez. An FPGA Evaluation of Karatusba-Ofman Multiplier Variants (in spanish). Technical Report CINVESTAV_COMP 2006-2, 12 pages, Computer Science Department CINVESTAVIPN, Mexico, May 2006.