ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.10, No.23, 1–12

A scalable RNS Montgomery multiplier over $\mathbb{F}_{2^m}$

Jingwei Hu[1a)], **Wei Guo**[1,2b)], **Jizeng Wei**[1c)], **and Ray C.C. Cheung**[3d)]

[1] *School of Computer Science and Technology, Tianjin University*

[2] *Tianjin Key Laboratory of Cognitive Computing and Application*

[3] *Department of Electronic Engineering, City University of Hong Kong*

a) *jingweihu@tju.edu.cn*

b) *weiguo@tju.edu.cn*

c) *weijizeng@tju.edu.cn*

d) *r.cheung@cityu.edu.hk*

Abstract: This paper presents a fully parallelized and scalable RNS Montgomery multiplier over binary field. By generalizing the RNS Montgomery Multiplication (RNS MM) and elaborating a highly efficient RNS base selection, we are able to obtain a considerably high speed in our FPGA implementation experiments with acceptable circuit area and modest critical path delay. Furthermore, this design can be easily scalable by adjusting a variety of field sizes and field polynomials.

Keywords: RNS MM, binary field, FPGA, scalable

Classification: Integrated circuits

References

[1] J.-C. Bajard, L.-S. Didier and P. Kornerup: IEEE Trans. Comput. **47** [7] (1998) 766.

[2] R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede and G. X. Yao: Cryptographic Hardware and Embedded Systems–CHES 2011 **6917** (2011) 421.

[3] C. K. Koc and T. Acar: Designs, Codes and Cryptography **14** [1] (1998) 57.

[4] P. Kitsos, G. Theodoridis and O. Koufopavlou: Microelectronics Journal **34** [10] (2003) 975.

[5] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew and S. Hsu: Proc. 17th IEEE Symp. Computer Arithmetic (ARITH) (2005) 172.

[6] D. Hankerson, A. J. Menezes and S. Vanstone: *Guide to elliptic curve cryptography* (Springer, 2004) 101.

[7] C. Grabbe, M. Bednara, J. Teich, J. von zur Gathen and J. Shokrollahi: Proc. 2003 International Symposium on Circuits and Systems **2** (2003) II–268.

[8] J.-P. Deschamps: *Hardware implementation of finite-field arithmetic* (McGraw-Hill, Inc., 2009).

[9] A. P. Fournaris and O. Koufopavlou: Integration, the VLSI Journal **41** [3] (2008) 371.

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

1 Introduction

Finite field arithmetic is used in many different digital communication systems and theoretical researches. Two well-known examples are error-coding theory and cryptography. Finite field is usually divided into prime field ($\mathbb{F}_p$) and binary extension field ($\mathbb{F}_{2^m}$). $\mathbb{F}_{2^m}$ is becoming rapidly attractive due to its "carry-free" logic which is well suitable for the hardware implementation.

In most cases, field multiplication (modular multiplication) defines the system overall performance, the efficient implementation of ($\mathbb{F}_{2^m}$) multiplier is now gaining an extensive attention. Among them, well-known MSbit-first (MSB), LSBit-first (LSB) and Karatsuba multipliers [6] have been proposed and expanded by many researchers. However, one big disadvantage of these multipliers is that they usually work in a specific, fixed field $\mathbb{F}_{2^m}$, cannot be easily extended to any other fields. What's more, to achieve a higher speed and low space complexity these multipliers adopt special polynomials as moduli, like AOL, trinomials or pentanomials [8], making their work not so scalable when defined over arbitrary field polynomial. Among alternative modular multiplication algorithms found in the literature, Montgomery's method has been extensively analyzed, since it replaces divisions with additions, multiplications and shifts [3]. Also this method can be easily tuned into different field sizes and field polynomials, putting the concept of scalable design methodology into practice [4, 9]. In recent years, RNS has enjoyed renewed scientific interest due to its ability to perform fast and parallel modular arithmetic. This method splits large-scale numbers into smaller ones (RNS channel reduction) by exploiting different small co-prime modulis (RNS bases). As a result, integrating the two methods above together, a series of RNS Montgomery multiplier (RNS MM) architectures over prime field have been proposed [1, 2].

In this paper, RNS MM algorithm over $\mathbb{F}_{2^m}$ is elaborately studied and optimized. After a careful mapping into the FPGA platform, an efficient scalable RNS MM architecture is proposed. The main contributions of this paper include proposing a binary field version of RNS MM algorithm, concluding a modified base extension algorithm without parameter approximations and adopting pseudo-Mersenne-like modulis reducing the time and area complexity in the architecture. Additionally, our design is scalable for different field sizes and field polynomials.

The paper is organized as follows: A brief introduction to RNS and Montgomery algorithm is given in section 2. The proposed RNS MM algorithm is detailed in section 3. Section 4 depicts the hardware implementation of our design on FPGA platform. Relevant analyses, results and comparisons are concluded in section 5.

2 Preliminaries

2.1 Extended binary field $\mathbb{F}_{2^m}$

Let $\beta \in \mathbb{F}_{2^m}$ and be a root of the irreducible polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ over $\mathbb{F}_2$. Then, the set of $\{1, \beta, \ldots, \beta^{m-1}\}$ con-

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

stitutes the *polynomial basis* in $\mathbb{F}_{2^m}$. With polynomials basis, the elements in $\mathbb{F}_2$ can be represented as polynomials of degree at m-1 in the form $\mathbb{F}_{2^m} = \{a(\beta)|a(\beta) = a_{m-1}\beta^{m-1} + \ldots + a_1\beta + a_0\}$, where the coefficients a_i are the polynomial basis coordinates in $\mathbb{F}_2$. Polynomial basis can also be represented as the set $\{1, x, \ldots, x^{m-1}\}$ and, therefore, $\mathbb{F}_{2^m} = \{a(x)|a(x) = a_{m-1}x^{m-1} + \ldots + a_1x + a_0\}$.

Arithmetic operations in $\mathbb{F}_{2^m}$ are performed modulo an irreducible polynomial $f(x)$ over $\mathbb{F}_2$. *Addition* of polynomials is carried out under modulo 2 arithmetic. Therefore, the addition of two polynomials becomes the bitwise exclusive-or (XOR) of their binary representations. *Subtraction* is exactly the same as addition in modulo 2 arithmetic, so $1 - x$ equals $1 + x$.

Among the $\mathbb{F}_{2^m}$ arithmetic operations, *multiplication* is usually considered the most important, complex, and time-consuming operation. With reference to a degree m irreducible polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ over $\mathbb{F}_2$, let $a(x)$ and $b(x)$ be two elements in the field and $c(x)$ be their product. Then *field multiplication* can be defined as:

$$c(x) = a(x)b(x) \bmod f(x) \tag{1}$$

2.2 Montgomery multiplication on $\mathbb{F}_{2^m}$

In order to reduce the computational complexity of *field multiplication* over $\mathbb{F}_{2^m}$, Koc [3] proposed Montgomery multiplication on $\mathbb{F}_{2^m}$. The basic idea of this method is to achieve field multiplication efficiently without trial divisions. **Algorithm 1** [3] depicts the word-level algorithm for the Montgomery multiplication on $\mathbb{F}_{2^m}$. Operand X is partitioned into $\lceil m/w \rceil$ words of length w where $X = \sum_{i=0}^{\lceil \frac{m}{w} \rceil} X_i x^{iw}$. Let S_0 and P_0 be the least significant words of S and P, respectively. This algorithm is similar to the algorithm given for the Montgomery multiplication of integers. The only difference is that the final subtraction step required in the integer case is not necessary in the polynomial case.

Algorithm 1: Montgomery Multiplication on $\mathbb{F}_{2^m}$

Input: $X = \sum_{i=0}^{\lceil \frac{m}{w} \rceil} X_i x^{iw}$, $Y = \sum_{i=0}^{m-1} y_i x^i$, $P = \sum_{i=0}^{\lceil \frac{m}{w} \rceil} P_i x^{iw}$, $P'_0 = P_0^{-1} \bmod x^w$
Output: $S = XYx^{-w} \bmod P$

```
S ← 0
for i ← 1 to ⌈m/w⌉ do
    S ← S + X_i Y
    M ← S_0 P'_0 ( mod x^w)
    S ← S + MP
    S ← S/x^w
return S
```

2.3 Residue Number System (RNS)

RNS is defined by pairwise co-prime integer constants: $\mathfrak{B} = \{b_1, b_2, \ldots, b_n\}$ and $M_{\mathfrak{B}} = \Pi_{i=1}^{n} b_i, b_i \in \mathfrak{B}$. Any integer X, $0 \leq X < M_{\mathcal{B}}$, is uniquely represented by $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$, where $x_i = X \bmod b_i = |X|_{b_i}$, $1 \leq i \leq n$. $b_i, i \in [1, n]$ is called *RNS base* (*RNS channel*), $\{X\}_{\mathfrak{B}}$ is called *RNS*

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

number, $x_i, i \in [1, n]$ is called *RNS element* and the process of $X \rightarrow |X|_{b_i}$ is called *channel reduction*.

The arithmetic operations of RNS are similar to ordinary integers with a great improvement in their parallelism. Let $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$, $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_n\}$, $\{R\}_{\mathfrak{B}} = \{r_1, r_2, \ldots, r_n\}$ be the RNS representation of X, Y, R respectively, the arithmetic operations of *RNS* are defined as follows:

- $R = |X \pm Y|_{M_{\mathcal{B}}} \Leftrightarrow \{R\}_{\mathcal{B}} = \{X\}_{\mathcal{B}} \pm \{Y\}_{\mathcal{B}}$, where $r_i = |x_i \pm y_i|_{b_i}$
- $R = |X \cdot Y|_{M_{\mathcal{B}}} \Leftrightarrow \{R\}_{\mathcal{B}} = \{X\}_{\mathcal{B}} \cdot \{Y\}_{\mathcal{B}}$, where $r_i = |x_i y_i|_{b_i}$
- $R = |X/Y|_{M_{\mathcal{B}}} \Leftrightarrow \{R\}_{\mathcal{B}} = \{X\}_{\mathcal{B}} \cdot \{Y^{-1}\}_{\mathcal{B}}$, where $r_i = |x_i y_i^{-1}|_{b_i}$

According to the operations above, field arithmetics become easier due to smaller operands in length and data independency in each RNS channel. Therefore, they are extremely suitable for parallelized hardware architectures to implement.

Likewise, it is possible to apply RNS into binary field because Chinese Remainder Theorem (CRT) still holds in binary field. That means, for a very large element in $\mathbb{F}_{2^m}$, RNS is capable of dividing it into several much smaller elements as a whole for the sake of efficient computation.

On top of that, the computation of Montgomery multiplication on $\mathbb{F}_{2^m}$ can become beneficiaries from RNS as well. This is primarily because operations in each step in **Algorithm 1** can be transformed into RNS form and thus, it is likely to make further efforts to improve the efficiency of this algorithm.

In the next section, RNS Montgomery multiplication is presented in accordance with this initiative.

3 Proposed RNS Montgomery multiplication on $\mathbb{F}_{2^m}$

3.1 RNS Montgomery reduction

Let $X, Y \in \mathbb{F}_{2^m}$, $\mathfrak{B} = \{b_1, b_2, \ldots, b_n\}$, $\mathfrak{C} = \{c_1, c_2, \ldots, c_n\}, n < m$ be two discrepant RNS bases and $M_{\mathfrak{B}} = \Pi_{i=1}^{n} b_i, b_i \in \mathfrak{B}$, $M_{\mathfrak{C}} = \Pi_{i=1}^{n} c_i, c_i \in \mathfrak{B}$. RNS Montgomery multiplication aims at computing $S = |XYM_{\mathfrak{B}}^{-1}|_p$ in $\mathfrak{B}, \mathfrak{C}$ (namely, $\{S\}_{\mathfrak{B}}$ and $\{S\}_{\mathfrak{C}}$), which is illustrated in **Table I** in comparison with the original Montgomery method. *Base Extension* shown in this table is used to transform representation in RNS base $\mathfrak{B}$ to that in RNS base $\mathfrak{C}$, or the opposite.

Table I. Derivation of RNS Montgomery Multiplication on $\mathbb{F}_{2^m}$

	in base $\mathfrak{B}$		in base $\mathfrak{C}$	Montgomery [3]
1	$\{T\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} \cdot \{Y\}_{\mathfrak{B}}$		$\{T\}_{\mathfrak{C}} \leftarrow \{X\}_{\mathfrak{C}} \cdot \{Y\}_{\mathfrak{C}}$	$T \leftarrow XY$
2	$\{Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \cdot \{p^{-1}\}_{\mathfrak{B}}$			$Q \leftarrow Tp^{-1} \bmod M_{\mathfrak{B}}$
3	$\{Q\}_{\mathfrak{B}}$	$\xrightarrow{BaseExtension}$	$\{Q\}_{\mathfrak{C}}$	
4			$\{S\}_{\mathfrak{C}} \leftarrow (\{T\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \cdot \{p\}_{\mathfrak{C}}) \cdot \{\lvert M_{\mathfrak{B}}{}^{-1}\rvert_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$	$S \leftarrow (T + Qp) \cdot M_{\mathfrak{B}}^{-1}$
5	$\{S\}_{\mathfrak{B}}$	$\xleftarrow{BaseExtension}$	$\{S\}_{\mathfrak{C}}$	

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

Notice that two RNS bases ($\mathfrak{B}$ and $\mathfrak{C}$) are necessary for the correctness of this algorithm. If either of them is left out, the result in step 4 will always be zero. For instance, assume base $\mathcal{C}$ has been taken away, that means *base extension* in step 3 is discarded. Accordingly, the computation has to be done in base $\mathcal{B}$ instead: $\{S\}_{\mathcal{B}} = (\{T\}_{\mathfrak{B}} + \{Q\}_{\mathfrak{B}} \cdot \{p\}_{\mathfrak{B}}) \cdot \{|{M_{\mathfrak{B}}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{B}} = (\{T\}_{\mathfrak{B}} + \{T\}_{\mathfrak{B}}) \cdot \{|{M_{\mathfrak{B}}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{B}} = 0$, which is base off the right track.

We have concluded the RNS Montgomery Multiplication on $\mathbb{F}_{2^m}$ in **Algorithm 2**. For a simplistic description of this algorithm, technical details of *base extension* inside will be addressed in the next subsection. This algorithm offers high parallelism with each RNS channel working independently. Meanwhile, scalability is obtained by increasing RNS channel numbers in the algorithm mentioned regardless of filed sizes or irreducible polynomials. Additionally, the stage of pre-computation in step 1 is irrelevant to operands X and Y. As a result, all the computations in this step can be computed beforehand and stored in memory components. Thus, this algorithm is further simplified.

Algorithm 2: RNS Montgomery Multiplication

Input: RNS bases $\mathfrak{B}$,$\mathfrak{C}$, multiplication operands $\{X\}_{\mathfrak{B}}$,$\{X\}_{\mathfrak{C}}$,$\{Y\}_{\mathfrak{B}}$,$\{Y\}_{\mathfrak{C}}$ being RNS representation of X and $Y (X, Y < x^{k+1})$ and moduli p

Output: $\{S\}_{\mathfrak{B}}$,$\{S\}_{\mathfrak{B}}$ such that $|S|_p = |XYM_{\mathfrak{B}}^{-1}|_p$

1 **Precompute** $\{|p^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$, $\{|{M_{\mathfrak{B}}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$ and $\{p\}_{\mathfrak{C}}$, where $M_{\mathfrak{B}} = \Pi_{i=1}^{n} b_i, b_i \in \mathfrak{B}, M_{\mathfrak{C}} = \Pi_{i=1}^{n} c_i, c_i \in \mathfrak{C}$

2 $\{T\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} \cdot \{Y\}_{\mathfrak{B}}$

3 $\{Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \cdot \{|p^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$

4 $\{Q\}_{\mathfrak{B}} \xrightarrow{BaseExtension} \{Q\}_{\mathfrak{C}}$

5 $\{T\}_{\mathfrak{C}} \leftarrow \{X\}_{\mathfrak{C}} \cdot \{Y\}_{\mathfrak{C}}$

6 $\{S\}_{\mathfrak{C}} \leftarrow (T_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \cdot \{p\}_{\mathfrak{C}}) \cdot \{|{M_{\mathfrak{B}}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$

7 $\{S\}_{\mathfrak{B}} \xleftarrow{BaseExtension} \{S\}_{\mathfrak{C}}$

8 **return** $\{S\}_{\mathfrak{B}}$

Nevertheless, the boundary values of inputs in **Algorithm 2** are supposed to be restricted so as to ensure the validity of this method. If the two base-extension steps (step 3 and step 5 in **Algorithm 2**) are error-free, we can specify the condition that $M_{\mathfrak{B}}$ and $M_{\mathfrak{C}}$ should satisfy for a given p. Condition that $gcd(M_{\mathfrak{B}}, p) = 1$ and $gcd(M_{\mathfrak{B}}, M_{\mathfrak{C}}) = 1$ is sufficient for the existence of $|p^{-1}|_{M_{\mathfrak{B}}}$ and $|{M_{\mathfrak{B}}}^{-1}|_{M_{\mathfrak{C}}}$ respectively. $M_{\mathfrak{B}} \geq x^{k+1}$ is also sufficient for $S < x^{k+1}$ when $x, y < x^{k+1}$. Actually,

$$\begin{aligned} S &= \frac{xy + |xy \times p^{-1}|_{M_{\mathfrak{B}}} p}{M_{\mathfrak{B}}} < \frac{xy + M_{\mathfrak{B}} p}{M_{\mathfrak{B}}} \\ &= \frac{xy}{M_{\mathfrak{B}}} + p < max\{x^{k+1}, p\} = x^{k+1} \end{aligned} \tag{2}$$

This equation also shows $M_{\mathfrak{C}} \geq x^{k+1}$ is sufficient for $S < M_{\mathfrak{C}}$. In summary, the following four conditions are sufficient for the correctness of this algorithm:

$$gcd(M_{\mathfrak{C}}, p) = 1, gcd(M_{\mathfrak{B}}, M_{\mathfrak{C}}) = 1, M_{\mathfrak{B}} \geq x^{k+1}, M_{\mathfrak{C}} \geq x^{k+1} \tag{3}$$

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

3.2 Base extension

The operation to transform the representation in one RNS base to another base is called Base Extension (BE). The reason why we have to do this is that one can not obtain the correct value S in step 4 of **Algorithm 2** unless Base Extension is done in step 3 (actually one can infer $S = 0$). The final base extension in step 5 of **Algorithm 2** helps to convert the value S back into the RNS form in $\mathfrak{B}$. To compute $\{T\}_{\mathfrak{C}} = \{t'_1, t'_2, \ldots, t'_n\}$ from $\{T\}_{\mathfrak{B}} = \{t_1, t_2, \ldots, t_n\}$, we exploit Chinese Remainder Theorem (CRT) to obtain the following equations,

$$T = \left| \sum_{i=1}^{n} |t_i B_i^{-1}|_{b_i} B_i \right|_{M_{\mathfrak{B}}} = \left| \sum_{i=1}^{n} \xi_i B_i \right|_{M_{\mathfrak{B}}} = \sum_{i=1}^{n} \xi_i B_i - \lambda M_{\mathfrak{B}} = \sum_{i=1}^{n} \xi_i B_i \quad (4)$$

where $|\xi_i|_{b_i} = |t_i B_i^{-1}|_{b_i}$, $1 \le i \le n$, and $B_i = M_B/b_i$. C_i is defined similarly for Base $\mathfrak{C}$. Notice that approximation parameter $\lambda = 0$ makes the Base Extension Transformation (**Algorithm 3**) quite easier in the binary field because no additional logic is required to evaluate the approximation, with multiplication and addition on $\mathbb{F}_{2^m}$ only to perform the base extension procedure. Then $\{T\}_{\mathfrak{C}} = \{t'_1, t'_2, \ldots, t'_n\}$ can be computed as follows:

$$t'_j = \left| \sum_{i=1}^{n} \xi_i B_i \right|_{c_j} = \left| \sum_{i=1}^{n} \xi_i |B_i|_{c_j} \right|_{c_j} \quad (5)$$

$|B_i|_{c_j}$ $(1 \le i, j \le n)$ can be precomputed once $\mathfrak{B}$ and $\mathfrak{C}$ are fixed.

Algorithm 3: Base extension algorithm for k-th element of $\{T\}_{\mathfrak{C}}$

Input: $|T|_{b_i}$, for $i \in \{1, \ldots, n\}$
Output: $|T|_{c_k}$
1 **Precompute** $|{B_i}^{-1}|_{b_i}, |B_i|_{c_k}$
2 $z \leftarrow 0$
3 **for** $i \leftarrow 1$ ***to*** n **do**
4 $\quad \xi_i \leftarrow |t_i {B_i}^{-1}|_{b_i}$
5 $\quad z \leftarrow z + \xi_i |B_i|_{c_k}$
6 **return** $|z|_{c_k}$

3.3 Base selection

Before starting to perform RNS Montgomery multiplication (**Algorithm 2**), it is required to turn primitive operands in representation of binary field into RNS elements. This initial operation of conversion is called *channel reduction* [2]. In order to expedite the processing of channel reduction, it is essential to find an appropriate RNS base selection. Here we propose the pseudo-Mersenne-like numbers $b_i = x^w + \xi(i)(w < m, \xi(i) < x^{w/2})$ as RNS bases in this paper. Consequently, for an operand $X \in \mathbb{F}(2^m)$ and RNS base $\mathfrak{B} = \{b_1, b_2, \ldots, b_l\}$, the conversion of this operand can be written as $(X \to \{X\}_{\mathfrak{B}})$:

$$\begin{aligned} x_i = |X|_{b_i} &= (X_H x^w + X_L) \bmod x^w + \xi(i) \\ &= X_H(x^w + \xi(i)) + X_H\xi(i) + X_L \bmod x^w + \xi(i) \\ &= X_H\xi(i) + X_L \bmod x^w + \xi(i), i \in \{1, \ldots, l\}, x_i \in \{X\}_{\mathfrak{B}} \end{aligned} \quad (6)$$

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

where X_L denotes the least significant w bits of X, X_H denotes the most significant $m-w$ bits of X and $x_i \in \{X\}_{\mathfrak{B}}$. It is noticeable that the conversion can be remarkably efficient when $\xi(i) < x^{w/2}$. As a matter of fact, the degree m of X is reduced to degree $m - w/2$ of $X_H\xi(i) + X_L$ via **Equation (6)**. After $\lceil \frac{2m}{w} \rceil - 2$ iterations of **Equation (6)**, RNS representation of X can be obtained eventually. By using this base selection, readers can also find the efficiency of the modular multiplication on each RNS channel presented in the next subsection.

3.4 Fast field multiplication on $\mathbb{F}(2^w)$

In this subsection, we tackle the matter of efficient implementation of $\{R\}_{\mathcal{B}} = \{X\}_{\mathcal{B}} \cdot \{Y\}_{\mathcal{B}}$, where $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$, $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_n\}$, $\{R\}_{\mathfrak{B}} = \{r_1, r_2, \ldots, r_n\}$, $r_i = |x_i y_i|_{b_i}$ $(1 \le i \le n)$. which is the paramount fundamental operation in the proposed RNS Montgomery multiplication algorithm (**Algorithm 2**).

Notice the operation of modular multiplication (namely, $r_i = |x_i y_i|_{b_i}$) on $\mathbb{F}(2^w)$ is critical becuase each operation is performed on $\mathbb{F}(2^w)$ in the context of RNS. The proposed base selection method in this paper is able to significantly reduce the computational intensiveness of this operation by exploiting the deliberately selected RNS bases.

Modular multiplication on $\mathbb{F}(2^w)$ involves two steps: polynomial multiplication and reduction modulo field polynomial. For the stage of polynomial multiplication, the multiplication of two operands in polynomial base consists of shift-operation and addition in $\mathbb{F}(2^w)$ and it can be easily implemented with utilization of XOR logic: Suppose $y_i = \sum_{j=0}^{w-1} (y_i)_j x^j$, then

$$r_i = x_i \cdot y_i = \sum_{j=0}^{w-1} x_i (y_i)_j x^j = \sum_{j=0}^{w-1} (x_i \cdot (y_i)_j) \gg j \tag{7}$$

As far as the stage of reduction is concerned, with reference to the pseudo-Mersenne-like numbers used as RNS modulis in section 3.3, the final reduction is greatly simplified, and we choose the pentanomial on the best performance (that is, $b_i = x^w + \xi(i) = x^w + x^l + x^m + x^n + 1$, we abandon the simplest trinomial because there are not enough co-prime trinomials for us to use). Nevertheless, these RNS bases in pentanomial form, unlike other literature [6, 8] mentioned in our introduction, do not exert negative influence on the scalability of our work. This is simply because all kinds of field polynomials (trinomials, pentanomials or whatever they are) can be adapted into RNS modulis via Channel Reduction. In other words, although RNS modulis are constructed in the form of pentanomials, our work is still capable of handling different type of filed polynomials, resulting in intactness of the hardware scalability. **Algorithm 4** depicts the simplicity of the field multiplication on $\mathbb{F}(2^w)$. This algorithm can be easily deduced from **Equation (6)** aforementioned $(\xi(i) = x^l + x^m + x^n + 1)$.

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

Algorithm 4: Fast Field Multiplication on $\mathbb{F}(2^w)$

Input: $x \in \mathbb{F}(2^w), y \in \mathbb{F}(2^w)$ and $b = x^w + x^l + x^m + x^n + 1, l < \lfloor w/2 \rfloor$
Output: $|xy|_b \in \mathbb{F}(2^w)$

$c \leftarrow xy$
for $i \leftarrow 0$ ***to*** 1 **do**
  $c_H \leftarrow c/x^w$
  $c_L \leftarrow c \mod x^w$
  $c \leftarrow x^l c_H + x^m c_H + x^n c_H + c_H + c_L$
return c

4 Hardware implementation

We have implemented the algorithm aforementioned on Xilinx Virtex-II platform. Figure 1 depicts the suitable architecture for the proposed RNS Montgomery multiplication algorithm (digit size $w = 33$, of $\mathbb{F}_{2^m}$). In our implementation, $\lceil \frac{m}{w} \rceil$ dual-mode multipliers (DMMs) are exploited to achieve

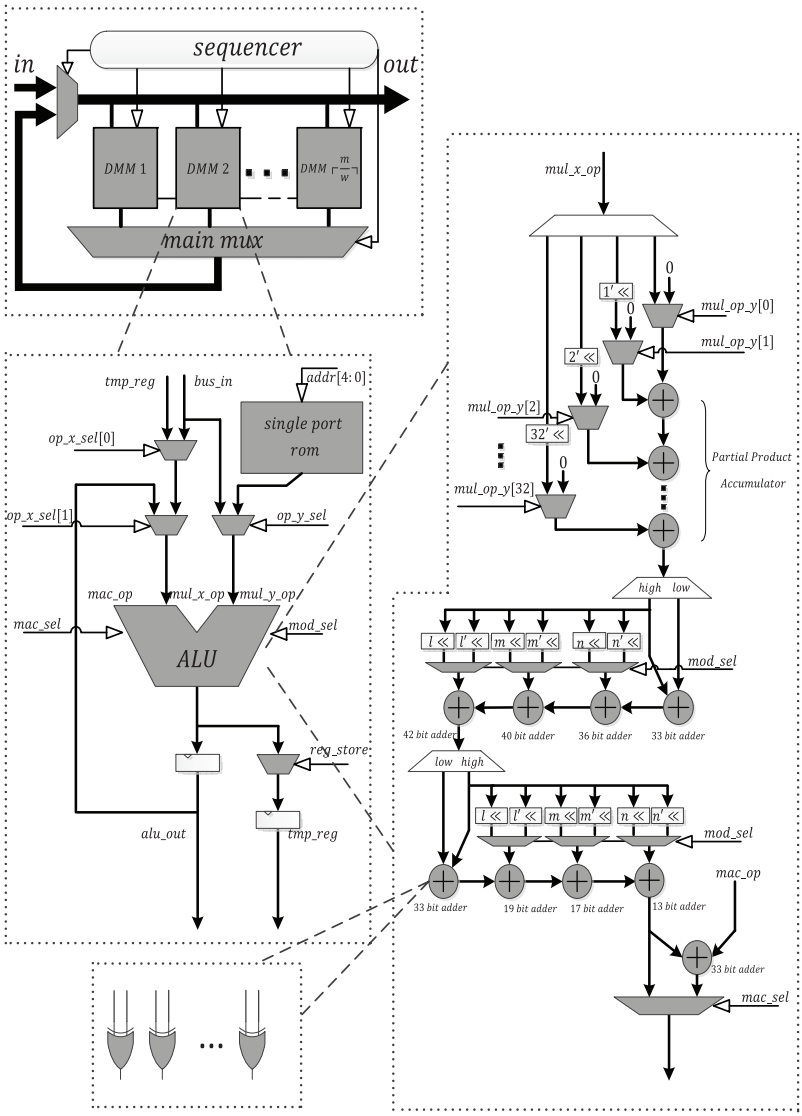

Fig. 1. Proposed $\mathbb{F}_{2^m}$ RNS Montgomery Multiplier Architecture

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

the highest parallelism. These multipliers controlled by a scheduling sequencer which performs the **Algorithm 2** in section 3. In fact, the sequencer is a finite state machine with 4 states (S_1, BT_A, BT_B, S_2). Step 1 and step 2 computed on RNS channel $\mathfrak{B}$ in **Algorithm 2** are executed in S_1, which takes two clock cycles each. On the other hand, step 5 and step 6, which are the computation on RNS channel $\mathfrak{C}$. Differently, S_2 takes three cycles to execute as step 5 takes one clock cycle while step 6 takes two (first perform multiply-accumulation then multiplication). BT_A and BT_B are about the Base Extension procedure in step 3 and step 4 respectively. Both of them take two cycles to achieve the task, which is detailed in **Algorithm 3**. The initial operands are conveyed into the multipliers through the system bus. The outputs of the multipliers are connected to the main MUX component which functions as the custodian for the bus entrance. There are interconnects between each of the multipliers because some intermediate results are shared during the Base Extension procedure (**Algorithm 3**).

Each DMM has two RNS channels (one in $\mathfrak{B}$ and the other in $\mathfrak{C}$) due to the two RNS bases employed within the algorithm. Our DMM is advantageous when one has to alter the modulis in the RNS channel to pursue a better time-area tradeoff, because one simply need to tune the $nbitshifter$ to the modulis they want (as long as the modulis are pseudo-Mersenne-like pentanomials, that is, $f = x^w + x^l + x^m + x^n + 1, l < \lfloor w/2 \rfloor$) without any other additional consumption. ROM is embedded into the multiplier with pre-computed $\{|p^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$, $\{|M_{\mathfrak{B}}{}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$, $\{p\}_{\mathfrak{C}}$, $|B_i|_{c_k}$, $|C_i{}^{-1}|_{c_i}$ and $|C_i|_{b_k}$ stored. **Table II** summarizes the memory requirements of the proposed architecture for $\mathbb{F}_{2^m}$ RNS Montgomery multiplication, note that there are exactly $5\lceil \frac{m}{w} \rceil w + 2\lceil \frac{m}{w} \rceil^2 w$ bits pre-computed data in total stored in the ROM. ALU serves as the arithmetic core component of the multiplier targeting the computation of multiplication and multiply-accumulation on $\mathbb{F}_{2^w}$ (namely, $mul\_x\_op \times mul\_y\_op$ and $mac\_op + mul\_x\_op \times mul\_y\_op$ respectively). The intermediate result generated in Base Extension procedure (**Algorithm 3**) is stored in tmp_reg, which is shared by all the other dual-mode multipliers.

The internal structure of ALU is primarily composed of XOR gates and switch MUXs. The Partial Product Accumulator (PPA) in the number of w $\mathbb{F}_2$ adders is used to generate the results of the multiplication with respect to (4). There are two reduction stages to obtain the final result by the method proposed in **Algorithm 4**. With 8 adders employed only, the reduction part

Table II. Memory Consumption of the Proposed RNS MM Architecutre over $\mathbb{F}_{2^m}$

Operation	Parameters stored	ROM Consumptions
RNS MM in Algorithm 2	$\{\|p^{-1}\|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$	$\lceil \frac{m}{w} \rceil w$ bits
	$\{\|M_{\mathfrak{B}}{}^{-1}\|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$	$\lceil \frac{m}{w} \rceil w$ bits
	$\{p\}_{\mathfrak{C}}$	$\lceil \frac{m}{w} \rceil w$ bits
Base Extension in Algorithm 3	$\|B_i{}^{-1}\|_{b_i}$ $(i\{\in 1, \ldots, \lceil \frac{m}{w} \rceil\})$	$\lceil \frac{m}{w} \rceil w$ bits
	$\|B_i\|_{c_k}$ $(i,k\{\in 1, \ldots, \lceil \frac{m}{w} \rceil\})$	$\lceil \frac{m}{w} \rceil^2 w$ bits
	$\|C_i{}^{-1}\|_{c_i}$ $(i\{\in 1, \ldots, \lceil \frac{m}{w} \rceil\})$	$\lceil \frac{m}{w} \rceil w$ bits
	$\|C_i\|_{b_k}$ $(i,k\{\in 1, \ldots, \lceil \frac{m}{w} \rceil\})$	$\lceil \frac{m}{w} \rceil^2 w$ bits

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

reveals a dramatic reduction in timing and area complexity. The final adder is functional when multiply-accumulation mode is enabled.

Another significant characteristic of our work is that the architecture proposed is easily adjusted into a variety of field sizes and field polynomials, making this design nicely scalable. One merely needs to trim the number of the multipliers to get adjusted to different field sizes without making any changes to interior of the multiplier. As for the field polynomials adopted for various applications, rewriting the data stored in ROM is sufficient enough to meet this requirement.

5 Performance and comparisons

In an effort to obtain the exact complexity of the proposed multiplier, we analyze and compare the time and area complexities of the presented RNS MM multiplier over $\mathbb{F}_{2^{233}}$ when it comes to different digit sizes in **Table III**. As the escalation of the digit size, it runs faster but with a considerable increase in circuit area. Thus it can be fairly desirable to restrict the digit size obtaining modest area consumption and an appropriate latency. The clock frequency of the proposed pipelined RNS Montgomery multiplier implementations remains almost constant as the chip covered area increases. It indicates that our design will maintain a modestly high clock frequency with a dynamic range of digit size.

Table III. Time and area complexity comparison of RNS Montgomery multiplier architecture over $\mathbb{F}_{2^{233}}$

Digit Size	Clock Cycles	Critical Path Delay	Time	Area
$w = 17$	35	$10T_{xor} + 6T_{mux}$ [a]	$350T_{xor}+210T_{mux}$	$\leq 5488S_{xor} + 432S_{mux}$ [b]
$w = 33$	23	$11T_{xor} + 6T_{mux}$	$253T_{xor}+138T_{mux}$	$\leq 9528S_{xor} + 344S_{mux}$
$w = 63$	15	$12T_{xor} + 6T_{mux}$	$180T_{xor} + 90T_{mux}$	$\leq 16644S_{xor} + 292S_{mux}$
$w = 127$	11	$13T_{xor} + 6T_{mux}$	$143T_{xor} + 66T_{mux}$	$\leq 33026S_{xor} + 274S_{mux}$

[a] T_{xor} = delay of xor gate, T_{mux} = delay of mux
[b] S_{xor} = area of xor gate, S_{mux} = area of mux

A casestudy on the performance for different field sizes (some of the NIST recommended binary fields and $\mathbb{F}_{2^{193}}$) is given in **Table IV**. We assume each RNS channel is fixed on $\mathbb{F}_{2^{33}}$ and maximal parallelism is reached ($\lceil \frac{m}{33} \rceil$ DMMs work simultaneously) for a more intuitive and clearer illustration. In this case, critical path delay is not affected by field size because DMM structure remains intact once datawidth of RNS channel is determined. To cater to different field sizes, one merely has to arrange MMs available, precomupte data in ROMs and slightly alter the implementation of sequencer in favor of constituting a robust scalable design.

The proposed RNS MM architecture is captured in Verilog HDL and implemented in hardware using Xilinx® Virtex™-II Family Series XC2V3000 device as the target FPGA (Virtex-4 and Virtex-5 implementations are also

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

Table IV. Scalability analysis of RNS Montgomery multiplier architecture over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{193}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$

Field Size	Clock Cycles[a]	Critical Path Delay	Time	Area
$m = 163$	17	$11T_{xor} + 6T_{mux}$	$187T_{xor}+102T_{mux}$	$\leq 5955S_{xor} + 215S_{mux}$
$m = 193$	19	$11T_{xor} + 6T_{mux}$	$209T_{xor}+114T_{mux}$	$\leq 7146S_{xor} + 258S_{mux}$
$m = 233$	23	$11T_{xor} + 6T_{mux}$	$253T_{xor}+138T_{mux}$	$\leq 9528S_{xor} + 344S_{mux}$
$m = 283$	25	$11T_{xor} + 6T_{mux}$	$275T_{xor}+150T_{mux}$	$\leq 10719S_{xor} + 387S_{mux}$

[a] Assume the highest parallelism is obtained and digit size $w = 33$.

Table V. FPGA Implementations for the Proposed RNS Montgomery Multipliers on $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, and $\mathbb{F}_{2^{283}}$

Field Size	Platform	LUTs	Critical Path Delay(ns)	Clock Cycles
163	Virtex-II	6,216	10.02	17
	Virtex-4	5,481	7.16	17
	Virtex-5	3,863	6.19	17
233	Virtex-II	9,978	10.13	23
	Virtex-4	8,785	7.21	23
	Virtex-5	6,215	6.25	23
283	Virtex-II	11,325	10.16	25
	Virtex-4	9,893	7.22	25
	Virtex-5	6,997	6.31	25

Table VI. Performance Comparison of FPGA Implementations for $\mathbb{F}_{2^m}$ Multipliers

Reference	Field Size	Platform	LUTs	CPD[b] (ns)	Clock Cycles	Scal.[a]	Speed-Up
LSBbit [6]	163	-	163	2.97	163	No	2.08
Kitsos [4]	163	Virtex-E	869	13.00	163	Yes	9.09
HybridKara [7]	233	Virtex-II	11,746	11.07	9	No	0.43
HarrisMont [5]	256	Virtex-II	5,987	6.94	144	Yes	4.29
FournarisMont [9]	163	Virtex-II	2,114	4.01	328	Yes	5.65
Ours	233	Virtex-II	9,978	10.13	23	Yes	1
		Virtex-4	8,785	7.21	23	Yes	-
		Virtex-5	6,215	6.27	23	Yes	-

[a] scalability
[b] critical path delay

included for future comparison by other people). We choose this device as our evaluation platform mainly for a fair and square comparison with other previous works published in literature.

To demonstrate the scalability of our work, the synthesized results of the proposed RNS MM multiplier on $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$ are indicated in **Table V**. Unified 33-bit DMM is adopted in this experiment. 5 DMMs like this are instantiated to construct a $\mathbb{F}_{2^{163}}$ RNS Montgomery multiplier, while 8 DMMs and 9 DMMs are required for a $\mathbb{F}_{2^{233}}$ multiplier and a $\mathbb{F}_{2^{283}}$ multiplier seperately.

Table VI compares the synthesized results of the proposed RNS MM multiplier on $\mathbb{F}_{2^{233}}$ with other existing work ($\mathbb{F}_{2^{233}}$ is the recommended binary field by NIST for elliptic curve digital signature algorithm (ECDSA)).

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013

To clarify the timing performance of this work, we introduce the following indicator shown in the last column of Table IV,

$$speed\text{-}up = \frac{\mathbf{T}_{benchmark}}{\mathbf{T}_{ours}} = \frac{\mathbf{CYCLES}_{benchmark} \times \mathbf{CPD}_{benchmark}}{\mathbf{CYCLES}_{ours} \times \mathbf{CPD}_{ours}} \quad (8)$$

Our work performs best among all the scalable design in terms of total computing time elapsed, with 9.09, 4.29, 5.65 speed-up improvement when the benchmark comes to [4, 5, 9] respectively. One can also find that the unscalable design (HybridKara [7]) performs even better than ours (about 57% above ours from the perspective of speed-up). But it should be marked that this design is defined over fixed size field $\mathbb{F}_{2^{233}}$ and over fixed special form irreducible polynomials (AOL, trinomials, pentanomials), which implies that their work will be unscalable when defined over arbitrary fields. The large LUT consumption of this design compared with [4, 5, 9] is due to the intrinsic property of RNS parallelism, in which 8 DMMs are employed to obtain an ultimate computational capability. But for some area or power constrained applications, it is not so critically intended. To get adjusted to these applications, the numbers of DMMs used can be trimmed to be smaller (that is, 4 DMMs or 2 DMMs, so that LUTs can be cut down into a half or a quarter of the primitive design). Summarizing all the above, computation speed is very fast for the proposed RNS Montgomery multiplier implementation with the modestly acceptable circuit area and our work features in supporting scalable design methodology.

6 Conclusion

In this paper, the RNS Montgomery multiplication (RNS MM) has been generalized into binary extension field and an efficient base selection method has been examined. Then we have presented a scalable RNS Montgomery multiplier architecture and we also have evaluated the performance of the proposed RNS Montgomery multiplier over $\mathbb{F}_{233}$. It has been implemented in FPGA and the area and timing results have been presented. The experimental results have shown that we are able to achieve an impressive high speed capacity (for field $\mathbb{F}_{2^{233}}$, the proposed architecture attains the Montgomery multiplication in 0.233 μs, with at least 4.29 speed-up compared to other Montgomery scalable designs in literature) and support different field sizes and irreducible polynomials.

Acknowledgments

This work is supported by the Key Project Foundation of Fundamental and Frontier Technology Research of Tianjin under Grant No.11JCZDJC1580 and the Open Project Program of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences.

DOI: 10.1587/elex.10.20130704
Received September 11, 2013
Accepted November 06, 2013
Publicized November 21, 2013
Copyedited December 10, 2013