A METHOD FOR FLEXIBLE REDUCTION OVER BINARY FIELDS USING A FIELD MULTIPLIER

Saptarsi Das[1], Keshavan Varadarajan[1], Ganesh Garga[2], Rajdeep Mondal[1], Ranjani Narayan[2] and S. K. Nandy[1]

[1]*CAD Lab, Indian Institute of Science, Bangalore, India*

[2]*Morphing Machines Pvt. Ltd., Bangalore, India*

Keywords: Elliptic Curve Cryptography, Binary Fields, Flexible Reduction, Polynomial Multiplication.

Abstract: Flexibility in implementation of the underlying field algebra kernels often dictates the life-span of an Elliptic Curve Cryptography solution. The systems/methods designed to realize binary field arithmetic operations can be tuned either for performance or for flexibility. Usually flexibility of these solutions adversely affects their performance. For solutions to reduction operation this adverse effect is particularly prominent. Therefore it is a non-trivial task to design a flexible reduction method/system without compromising performance. In this paper we present a method for flexible reduction. The proposed reduction technique is based on the well-known repeated multiplication technique and Barrett reduction. This technique is particularly appealing in the context of coarse-grain programmable architectures where performance of any kernel is heavily influenced by granularity of operations. In this context we propose a design of a polynomial multiplier based on the well-known Interleaved Galois Field multiplier to accelerate the underlying multi-word polynomial multiplications. We show that this modified IGF multiplier offers a significant improvement in throughput over a purely software realization or a hybrid software-hardware implementation using a conventional polynomial multiplier.

1 INTRODUCTION

Proliferation of various kinds of threats has lead to an increased interest in cryptographic solutions for communication equipments. Thus strong cryptography has emerged as an indispensable part of different communication protocols. One of the strongest deterrents of such threats is the class of Elliptic curve cryptography (ECC) algorithms. Due to ever increasing threat level, the key-length applied to these algorithms keeps on increasing with time. In order to cope with such growing need for stronger security the ideal approach would be to design ”future-proof” solutions. Quantitatively, the life-span of such a solution can be evaluated by measuring its flexibility to support various key lengths. The ECC algorithms are designed based on algebraic properties of finite fields. The nature of arithmetic involved in these algorithms makes it difficult to build arbitrarily flexible solutions without compromising on performance. The two fundamental operations involved in finite field arithmetic are addition and multiplication. Binary fields (finite fields of the form $GF(2^m)$) are especially popular due to the ease of implementation of addition and subtraction (which are equivalent to one another) over them. However, multiplication is a relatively expensive operation. Unlike addition or subtraction, multiplication of two polynomials from a finite field may produce a polynomial whose degree exceeds the order of the finite field. In order to translate such a result to an equivalent canonical form within the order of the finite field, a reduction operation is performed. Flexibility in polynomial multiplication can be easily achieved. However, supporting flexible reduction efficiently over arbitrarily large binary fields and for any irreducible polynomial requires special attention.

In this paper we investigate the case of flexible reduction and analyze different possible solutions. In section 2 we discuss the nature of the reduction operation and show that a software-hardware hybrid solution is best suited for flexible reduction over *any* binary field using *any* irreducible polynomial. We identify the possibility of using a hardware assist in the form of a field multiplier for improving the performance of such a hybrid technique. In this context we present the design of a Modified Interleaved Galois Field (MIGF) multiplier as an accelerator for the well-known Repeated Multiplication Method of reduction. In section 3 we present the improvement in performance achieved through the use of MIGF multiplier. We compute the increase in hardware complexity of the said multiplier which is offset by the improvement in performance of the reduction operation. In section 4 we present the synthesis results of a 32-bit MIGF multiplier and evaluate the improvement in performance of reduction operation over five NIST recommended irreducible polynomials. Finally we conclude the paper with a short summary.

2 REDUCTION OVER BINARY FIELDS: THE BASIC OPERATIONS INVOLVED AND THEIR REALIZATION

The reduction operation is a modulo operation of a polynomial with an irreducible polynomial that generates the finite field under consideration. Section 2.1 presents a brief mathematical background of reduction operation, various ways of implementing it and the associated implications. In section 2.2 we compare two algorithms for reduction operation and identify polynomial multiplications as the core computations in them. In section 2.3 we analyze the multiplication operations involved in reduction. In section 2.4 we present the design of a MIGF multiplier that can be used for efficient implementation of the aforementioned polynomial multiplications.

2.1 Mathematical Background of Reduction Operation

Elements of a binary field are usually represented as polynomials over the base field $GF(2)$ i.e. the degree of the polynomials is determined by the order of the field and the coefficients belong to $GF(2)$. Multiplication of such elements is governed by the addition and multiplication rules over $GF(2)$. For instance, let us consider two elements $A(x)$ and $B(x)$ belonging to the binary field $GF(2^m)$. These polynomials can be represented as a string of m symbols, where each symbol is 0 or 1. Therefore they are equivalent to two m-bit long binary strings. Equation 1 shows the two polynomials and their product $C(x)$.

$$\begin{aligned} A(x) &= \Sigma_{i=0}^{m-1} a_i x^i \\ B(x) &= \Sigma_{i=0}^{m-1} b_i x^i \\ C(x) &= A(x) \times B(x) \\ &= \Sigma_{i=0}^{2m-2} c_i x^i; where\, c_k = \Sigma_{i+j=k} a_i b_j \end{aligned} \tag{1}$$

As is apparent from equation 1, the result $C(x)$ is almost twice as long as the input polynomials. $C(x)$ has a unique equivalent canonical representation among the set of polynomials of degree $m-1$. Though, mathematically both the representations are equivalent, efficient utilization of computation resources necessitates conversion from the $2m-1$-bit representation to the m-bit representation. This conversion often referred to as reduction operation is based on an irreducible polynomial that generates the binary field of interest. The reduction operation is based on the fact that a polynomial $C(x)$ belonging to a finite field is equivalent to the polynomial modulo an irreducible polynomial $P(x)$ that generates the finite field.

$$C(x) \equiv C(x)\, mod\, P(x) \tag{2}$$

From equation 2 it is clear that the reduced polynomial can be computed by traditional long division technique for polynomials. But this method is iterative in nature and requires up to $m-1$ iterations.

At this point let us digress a little and consider the aspect of flexibility regarding reduction operation over finite fields. There are two major factors that govern flexibility of a reduction method: the order of the finite field and the irreducible polynomial that generates the finite field. A flexible reduction method/system should be capable of operating over finite fields of arbitrarily large order. Such a solution should also be versatile enough to handle *all* possible irreducible polynomial for *any* given field order. A purely hardware approach (Peter et al., 2007; Saqib et al., 2004) to support arbitrarily flexible reduction cannot be employed since a hardware solution cannot be used for finite field beyond a certain range. Moreover supporting all possible irreducible polynomials even upto a specified field order will immensely increase the complexity of the hardware. A purely software implementation is capable of delivering the desired flexibility, but poor performance of such an implementation may make it highly inefficient over very large fields. In order to cope with this, it is necessary to develop hybrid solutions. In a hybrid solution the data-path of the core computations are realized as fast hardware kernels and the control-path to invoke and cascade the hardware kernels is realized using a thin layer of software. Such coexistence of hardware and software necessitates some kind of a protocol to govern the communication between the two domains. One of the most important aspects of such a protocol is the data-granularity of the hardware kernels. Data-granularity determines the amount of data that can be processed by the individual hardware kernels at any time. In architecture terminology, this granularity translates to word-length. Transport latency of data and metadata in such hybrid systems is strongly dependent on data-granularity. In order to minimize (or even hide) the transport latency, it is preferable to deploy coarse-grain hardware kernels in hybrid systems. However, it should be noted that, higher granularity implies increased complexity of the hardware kernels. Therefore it is essential to find a balance in hardware complexity versus data-granularity to design optimized hybrid systems.

With this background let us get back to the case of reduction operation. As mentioned before reduction involves a series of basic arithmetic and logical operations. The order of the finite field influences the implementation of these basic operations on a coarse-grain hybrid system. As mentioned before, an element of $GF(2^m)$ can be represented as an m-bit wide binary string. Let the word length of a certain coarse-grain system be w. If $m \leqslant w$, then the number can be represented within a single word. In such a situation, it is feasible to develop a $m \times m$ multiplier and reduction operation can be integrated with the multiplier itself. Such a multiplier requires three inputs to operate, two numbers to be multiplied and an irreducible polynomial for reduction of the product. Equation 3 describes multiplication of $A(x)$ and $B(x)$ belonging to $GF(2^m)$, which is generated by the irreducible polynomial $P(x)$.

$$\begin{aligned} C(x) &= (A(x) \times B(x))\, mod\, P(x) \\ &= (A(x) \times (\Sigma_{i=0}^{m-1} b_i x^i))\, mod\, P(x) \\ &= \Sigma_{i=0}^{m-1} b_i (A(x)x^i\, mod\, P(x)) \end{aligned} \tag{3}$$

Equation 3 describes the operation of a traditional shift-and-add multiplier. Note that, the shifted multiplicands of the form $A(x)x^i$ are reduced at each stage. So the $m-1$ iterations of the modulo operation are embedded in each stage of the multiplier.

On the other hand, if $m > w$, the element of the finite field can be represented using $\lceil \frac{m}{w} \rceil$ words. Therefore the direct multiplication-and-reduction approach cannot be applied. Under such circumstances, it becomes imperative to employ a software algorithm to break the m-bit operations into w-bit operations. Multiplication of two m-bit polynomials produces a $2m-1$-bit result which needs to be reduced separately. Algorithms such as the Karatsuba-Ofman (Karatsuba and Ofman, 1963) algorithm can be applied iteratively to perform the aforementioned multiplications. The $2m-1$-bit result can be reduced by the Repeated Multiplication Reduction (RMR) method (Eberle et al., 2003; Satoh and Takano, 2003). The RMR method and Barrett Reduction (Barrett, 1987) method are the most suitable techniques for flexible reduction. A brief description of the RMR method is reproduced here from (Eberle et al., 2003). Let $C_0(x)$ be the product of two polynomials of degree less than m. The degree of $C_0(x)$ is less than $2m-1$ and it can be split into two parts as shown in equation 4.

$$C_0(x) = C_{h,0}(x)x^m + C_{l,0}(x) \tag{4}$$

The RMR method is an iterative technique and the subsequent polynomials are computed by the equation 5.

$$\begin{aligned} C_{j+1}(x) &= C_{h,j}(x)(P(x) - x^m) + C_{l,j}(x) \\ until\, C_{h,j+1}(x) &= 0 \Leftrightarrow deg(C_{j+1}(x)) \leqslant m-1 \end{aligned} \tag{5}$$

The RMR method requires m iterations and each iteration involves multiplication of $C_{h,j}(x)$ with $P(x) - x^m$. This polynomial multiplication can be realized as a set of m left shift operations. However, the most commonly used irreducible polynomials are usually trinomials and pentanomials. This implies, each of the multiplication involves no more than five left shift operations. Moreover $deg(P(x) - x^m) < \frac{m}{2}$. For such classes of polynomials, the RMR method converges after only two iterations (Peter et al., 2007). In (Knezevic et al., 2008) the authors have presented an adaptation of the famous Barrett Reduction method for binary fields. In section 2.2 we analyze the RMR and Barrett reduction method and establish an equivalence between the two.

2.2 Barrett Reduction and the RMR Method

We reproduce the adaptation of Barrett reduction from (Knezevic et al., 2008) to compare with the RMR method for irreducible polynomials with the following property: $deg(P(x) - x^m) < \frac{m}{2}$. Let us consider the RMR method first. Let $C_0(x)$ be the product of two polynomials that needs to be reduced. $P(x) = x^m + x^k + \cdots + x^p + 1$ be the irreducible polynomial. Note that, for the RMR method to converge within two iterations, k should be less than $\frac{m}{2}$. Using equation 5 we reduce the polynomial $C_0(x)$ as shown in equation 6.

$$\begin{aligned} C_0(x) &= C_{h,0}(x)x^m + C_{l,0}(x) \\ C_1(x) &= C_{h,0}(x)(P(x) - x^m) + C_{l,0}(x) \\ &= C_{h,1}(x)x^m + C_{l,1}(x) \\ where\, C_{h,1}(x) &= C_{h,0}(x)(P(x) - x^m) div x^m \\ and\, C_{l,1}(x) &= C_{h,0}(x)(P(x) - x^m) mod x^m \\ &\quad + C_{l,0}(x) \\ C_2(x) &= C_{h,1}(x)(P(x) - x^m) + C_{l,1}(x) \end{aligned} \tag{6}$$

Clearly, $deg(C_{h,1}(x)) \leqslant k$ and therefore $deg(C_2(x)) < m$. Now, let us consider the Barrett Reduction method for the same irreducible polynomial. Barrett Reduction involves computation of three quotients $Q_1(x)$, $Q_2(x)$ and $Q_3(x)$ along with two remainders $R_1(x)$ and $R_2(x)$ as shown in equation 7. The final result is given by the remainder polynomial $R(x)$.

$$\begin{aligned}
Q_1(x) &= C_0(x)\ div\ x^m \\
&= C_{h,0}(x) \\
Q_2(x) &= Q_1(x)P(x) \\
Q_3(x) &= Q_2(x)\ div\ x^m \\
&= C_{h,0}(x)(x^m + x^k + \cdots + x^p + 1)\ div\ x^m \\
&= C_{h,0}(x) + C_{h,1}(x) \\
R_1(x) &= C_0(x)\ mod\ x^m \\
&= C_{l,0}(x) \\
R_2(x) &= Q_3(x)P(x)\ mod\ x^m \\
&= Q_3(x)(x^k + \cdots + x^p + 1)\ mod\ x^m \\
&= C_{h,0}(x)(x^k + \cdots + x^p + 1) mod x^m \\
&\quad + C_{h,1}(x)(x^k + \cdots + x^p + 1) mod x^m \\
R(x) &= R_1(x) + R_2(x) \\
&= C_{h,1}(x)(P(x) - x^m) + C_{l,1}(x) \qquad (7)
\end{aligned}$$

From equations 6 and 7 it is evident that both the methods are equivalent and both of them require multiplication of m-bit polynomials. Note that, the modulo and division operations in the two methods translate to partitioning of the polynomials into lower and higher half and therefore do not require any arithmetic operation. In section 2.3 we present a method for performing the aforementioned multiplications in order to achieve arbitrary flexibility in reduction.

2.3 Multiplication Operations in Reduction

From equations 6 and 7 we observe that multiplications of the form $C(x)(x^k + \cdots + x^p + 1)$ form the core of the computations. Therefore it is necessary to accelerate these multiplications in order to perform fast reduction. It should also be noted that the only other operations involved in reduction are addition over $GF(2^m)$. Since there is no carry involved in addition, addition of two m-bit polynomials which span more than one word in a w-bit architecture can be realized as $\lceil \frac{m}{w} \rceil$ w-bit XOR operations. Multiplication on the other hand requires multi-word shift and accumulation of results. Consider the two polynomials $C(x)$ and $P'(x)$ of degree m and k respectively. These polynomials can be represented in a w-bit architecture as a collection of m_c and m_p w-bit words respectively. Equation 8 shows the representation.

$$\begin{aligned}
C(x) &= \Sigma_{i=0}^{m_c-1} C_i(x)x^{iw}\ where\ m_c = \left\lceil \frac{m}{w} \right\rceil \\
P'(x) &= \Sigma_{j=0}^{m_p-1} P'_j(x)x^{jw}\ where\ m_p = \left\lceil \frac{k}{w} \right\rceil \qquad (8)
\end{aligned}$$

$C_i(x)x^{iw}$ and $P_j(x)x^{jw}$ denote the i-th and j-th words of the polynomials $C(x)$ and $P'(x)$ respectively. The product of these two polynomials can be computed as follows:

$$\begin{aligned}
C'(x) &= C(x)P'(x) \\
&= \Sigma_{j=0}^{m_p-1} C(x)P'_j(x)x^{jw} \\
&= \Sigma_{j=0}^{m_p-1}(\Sigma_{i=0}^{m_c-1} C_i(x)x^{iw}P'_j(x)x^{jw}) \qquad (9)
\end{aligned}$$

A closer look at equation 9 reveals that computation of $C_i(x)P'_j(x)$ involves computations of the form $C_i(x)x^r$. Each of the individual words like $C'_{i,j}(x)$ (refer to figure 1) in the product of the entire polynomial $C(x)$ and x^r can be computed as follows:

$$C'_{i,j} = (C_i \ll r \mid C_{i-1} \gg (w-r)) \qquad (10)$$

The individual words like $C'_{i,j}(x)$ in the product of $C(x)$ and $P'_j(x)$ can be expressed as given by equation 11.

$$C'_{i,j} = \oplus_{r=0}^{w-1}(C_i \ll r \mid C_{i-1} \gg (w-r))p'_{j,r} \qquad (11)$$

Note that, $p'_{j,r}x^r$ denotes the r-th term in the j-th word of the polynomial $P'(x)$ in equation 11. The operations of equation 11 can be repeated for each of the words in $P'(x)$ to compute the final result. Note that the product of $C(x)$ and each of the words in $P'(x)$ is $m_c + 1$ word wide. Henceforward we will refer to products of $C(x)$ with the individual words of $P'(x)$ as "partial products". It should be noted that these $m_c + 1$ word wide partial products need to be aligned to proper word boundaries before they can be added together to produce the final result. Figure 1 shows how the partial products are aligned.

2.4 A Modified Interleaved Galois Field Multiplier as a Hardware Assist for Reduction

The discussion in section 2.3 makes it clear that a reduction method is only as fast as the underlying multiplication operations. Therefore it is obvious that polynomial multiplication kernels are the candidates for acceleration in a crypto-system. The simplest way of accelerating a $w \times w$ polynomial multiplication is to introduce a w-bit polynomial multiplier that produces

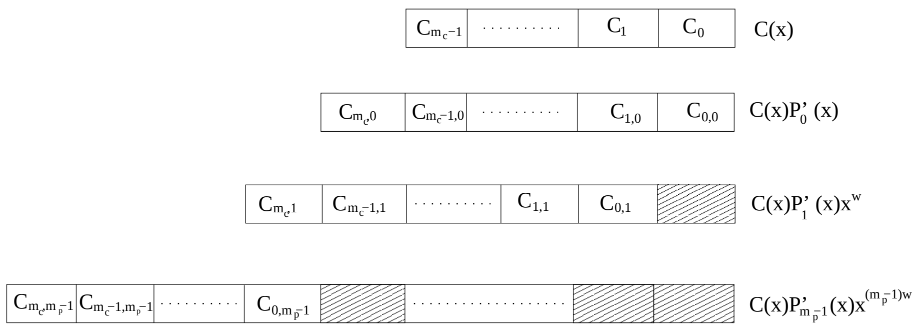

Figure 1: Arrangement of Partial Products.

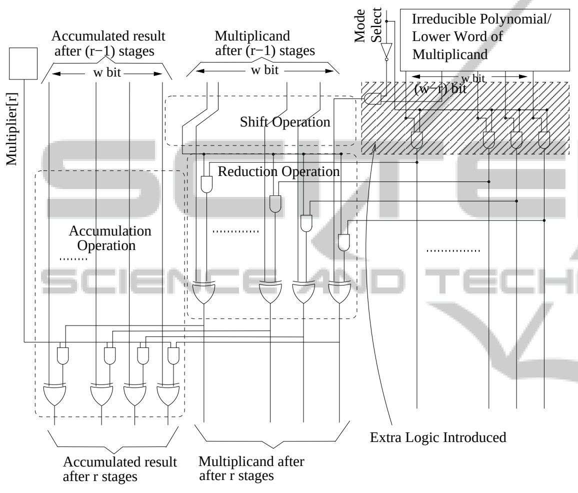
Accumulated result after (r−1) stages
Multiplicand after (r−1) stages
Mode Select
Irreducible Polynomial/ Lower Word of Multiplicand
w bit
(w−r) bit
Multiplier[r]
Shift Operation
Reduction Operation
Accumulation Operation
Extra Logic Introduced
Accumulated result after r stages
Multiplicand after after r stages

Figure 2: One stage of the Modified IGF Multiplier.

$2w$-bit results. Therefore each word in the input polynomial $C(x)$ produces a pair of words and these pairs need to be added (i.e. XORed) with proper alignment to compute a partial product.

In this section we propose a technique for combining the addition operations with the polynomial multiplications. Instead of considering one word of the polynomial $C(x)$ we focus on one word of the partial product (i.e. $C'_{i,j}(x)$). It is evident from equation 11 that to produce $C'_{i,j}(x)$ two words from the polynomial $C(x)$ and one word from $P'(x)$ are necessary. Thus the intended operation can be described as a $2w \times w$ polynomial multiplication that produces a w-bit result. In this section we show that an Interleaved Galois Field (IGF) Multiplier (Hinkelmann et al., 2009) can be modified to support this type of multiplications. In a shift-and-add IGF multiplier, the multiplicand operand is successively left shifted and the multiplier operand is used to selectively accumulate the results of the left shift operations. The IGF multiplier always produces a reduced result. Reduction over *large* fields however, requires support for multiplication of polynomials where the result is kept unreduced. This can be achieved by setting the irreducible polynomial to all zeros. This is achieved by masking the irreducible polynomial input to each stage of the multiplier with a one bit control signal (Mode Select signal in figure 2). In order to emulate the operations described in equation 11 the MIGF multiplier inserts the $(w-r)$-th bit from the second multiplicand operand to the LSB of the first multiplicand at the r-th stage of the multiplier. This is enabled by introducing a single AND gate that drives the LSB of the shifted polynomial. As can be seen from figure 2, we use the inverted control signal to mask the $(w-r)$-th bit from the second multiplicand operand. This added hardware (shown inside the shaded rectangle in figure 2) enables the multiplier to perform two-word shift operations successively which in turn alleviates the need for adding the individual products of the multiplier to form the partial product. In the section 3 we discuss the reduction in instruction-count of the reduction operation using this MIGF multiplier and the flexibility that this technique offers.

3 PERFORMANCE AND FLEXIBILITY OF THE PROPOSED REDUCTION TECHNIQUE

In this section we analyze the reduction method described in section 2.3 to evaluate the improvement in performance of the method with an MIGF multiplier as a hardware assist. We also show that this technique is arbitrarily flexible in terms of field order and choice of irreducible polynomial.

3.1 Performance Improvement due to use of Modified IGF Multiplier

In this section we analyze the benefits of using the MIGF multiplier for multiplication of $C(x)$ with

$P'(x)$. We proceed by first considering the case of multiplication of the entire C(x) polynomial with just one word of $P'(x)$ and continue the analysis to complete multiplication of $C(x)$ with entire $P'(x)$.

3.1.1 Multiplying $C(x)$ with One Word of $P'(x)$

From equation 10 it is evident that each m_c-word shift operation corresponding to a single term in the polynomial $P'(x)$ translates to $2m_c$ shift operations and $m_c - 1$ logical concatenations. Note that the concatenations can be conveniently expressed as either logical OR operations or logical XOR operations. Therefore the total number of logical operations necessary to produce the result of the multi-word multiplication as described in equation 11 is determined by the number of terms other than 1, present in the word of the irreducible polynomial $P'_j(x)$. Assuming there are p terms present in $P'_j(x)$, the number of logical shifts is $2m_cp$ and the number of concatenation operations is $(m_c - 1)p$. Note that these operations are required to produce the shifted polynomials of the form $C(x)x^r$ for different values of $r < w$. In order to accumulate these shifted polynomials of the form $C(x)x^r$ it is necessary to perform at most $p' - 1$ logical XOR operation for each word of the shifted polynomials where p' is total the number of terms present in $P'_j(x)$ (including a 1 if any). Since the shifted polynomials span across $(m_c + 1)$ words, the total number of XORs necessary is $(m_c+1)(p'-1)$ to accumulate the shifted polynomials. Thus the total number of basic logic operations necessary to compute $C(x)x^r$ is $2m_cp$ shift operations and $(m_c-1)p+(m_c+1)(p'-1)$ logical XORs. Using a polynomial multiplier to produce the partial product requires m_c multiplications and $m_c - 1$ XOR operations. Using the MIGF multiplier for this operation requires $m_c + 1$ two-word multiplications i.e one extra multiplication for $m_c - 1$ XOR operations. Thus we have reduced the total number of arithmetic and logical operations by approximately $4p$ times, when compared to a purely software realization.

3.1.2 Multiplying $C(x)$ with Entire $P'(x)$

So far we have considered multiplication of the polynomial $C(x)$ with one word of the irreducible polynomial $P'(x)$. With the analysis of the previous paragraph as the basis let us compute the number of operations involved in realizing the entire multiplication operation described in equation 9. The number of shift operations involved is determined by the number of terms with *distinct* indices present in the polynomial $P'(x)$. In a w-bit architecture $C(x)x^{tw+r}$ is computed by simply appending t words filled with zeros to the right of $C(x)x^r$. Therefore the terms of $P'(x)$ with indices $tw+r$ are equivalent to one another. Assuming that there are p terms with distinct indices present in the irreducible polynomial, the total number of shift operations necessary is given by $2m_cp$. The number of concatenation operations is $(m_c-1)p$. However, the total number of XOR operations required for accumulation of these shifted polynomials is determined by the number of terms (with distinct and equivalent indices) present in $P'(x)$. In order to calculate the total number XOR operations for accumulation it is necessary to examine the candidates for accumulation. Let us denote the number of terms present in $P'_j(x)$ by p_j. Therefore the accumulation of the shifted polynomials of the form $C(x)x^r$, produced by these p_j terms require $(p_j-1)(m_c+1)$ XOR operations. It should be noted that if $p_j = 0$ for any particular word $P'_j(x)$, no XOR operation is necessary. For simplicity let us assume $p_j > 0$ for all j. Total number of basic arithmetic-logic instructions to produce all the polynomials of the form $C(x)P'_j(x)$ is shown in equation 12

$$\begin{aligned} \#SHIFT &= 2m_cp \\ \#XOR &= (m_c-1)p+\Sigma_{j=0}^{m_p-1}(m_c+1)(p_j-1) \end{aligned} \tag{12}$$

It should be noted that, the number of XOR operations required to add results produced by different words of the polynomial $P'(x)$ remains unaltered irrespective of whether the MIGF multiplier is used or not. Thus we have intentionally not considered such XOR operations in counting the total number of XOR operations.

Using the MIGF multiplier reduces the number of operation required to perform the same set of operations. Note that, an intelligent sequencing of multiplication operations is necessary to minimize the number of multiplications. Sequencing of multiplication operation can be done by examining the irreducible polynomial. A set of m_c+1 multiplications are necessary to produce a term like $C(x)P'_j(x)$. However, it should be noted that this set of multiplications need to be performed for words of the polynomial $P'(x)$, with at least one term present. Therefore the maximum number of such multiplications necessary is $(m_c+1)m_p$. Clearly $(m_c+1)m_p < 2m_cp + (m_c-1)p+\Sigma_{j=0}^{m_p-1}(m_c+1)(p_j-1)$. Let us take this comparison a little further by making a set of assumptions. Let us assume that on an average p' terms are present in each of the words that constitute the irreducible polynomial. In that case the total number of basic arithmetic-logic operations involved can be simplified to $2m_cp+(m_c+1)(p'-1)m_p$. Using a conventional $w \times w$ polynomial multiplier will require m_cm_p

multiplications and $(m_c-1)m_p$ additional XOR operations. Using the MIGF multiplier brings down the total number of operations to $(m_c+1)m_p$. Assuming m_c is large enough so that $m_c \approx m_c+1$ the reduction in operation count is $2p/m_p+(p'-1)$ times when compared to a purely software realization and $2\times$ compared to hybrid realization using a conventional polynomial multiplier.

3.2 Flexibility of the Reduction Method

In this section we will analyze the flexibility of the reduction method. As discussed in section 2.3 the reduction operation is realized as series of polynomial multiplications. The number of individual multiplication operations is determined by two factors: the order of the finite field in consideration (2^m) and the word-length of the architecture (w). The RMR method of reduction is flexible by nature, since it does not impose any restriction on the order of the finite field or the nature of the irreducible polynomial $P(x)$. The Barrett reduction method, which is be shown to be a special case of RMR method imposes the restriction $deg(P(x)-x^m) < \frac{m}{2}$ on the irreducible polynomial $P(x)$, in order to improve performance. The elliptic curves suggested by NIST follow this restriction and therefore only two iterations of multiplications are sufficient for the result to converge. We evaluated the decrease in number of instructions brought about by usage of an MIGF multiplier as a hardware assist for multi-word multiplication in section 3.1. This analysis is completely general in nature and we have not made any assumption regarding the nature of the irreducible polynomial. Therefore the speed-up we computed applies in general to reduction with any arbitrary irreducible polynomial.

3.3 Hardware Complexity of the Modified IGF Multiplier

As shown in figure 2, we introduced a set of two-input AND gates in each stage of the MIGF Multiplier to enable two-word shift operations. In a w-bit instance of the multiplier, two sets of w two input AND gates are introduced. The first set of w two input AND gates are used for masking the irreducible polynomial input to the multiplier to zero. The second set of w two input AND gates are used for enabling two-word shift operation. This increase in hardware complexity is compensated by the significant reduction in the number of operations brought about by using this multiplier as a hardware assist for reduction.

Table 1: Synthesis Results of a IGF Multiplier and an MIGF Multiplier.

Type of Multiplier	Area in μm^2	Max. Operating Freq. in MHz
IGF Multiplier	42228	270
MIGF Multiplier	42255	256

Table 2: NIST recommended Irreducible Polynomials.

Size	Recommended Irreducible Polynomial
163	$x^{163}+x^7+x^6+x^3+1$
233	$x^{233}+x^{74}+1$
283	$x^{283}+x^{12}+x^7+x^5+1$
409	$x^{409}+x^{87}+1$
571	$x^{571}+x^{10}+x^5+x^2+1$

4 RESULTS

In this section we present the synthesis results of a 32-bit MIGF multiplier and evaluate the improvement in performance of reduction operations over the NIST curves using the MIGF multiplier.

4.1 Synthesis Results of a 32-bit Modified IGF Multiplier

We implemented a 32-bit instance of an MIGF multiplier using verilog HDL and synthesized with Faraday Tech 90nm standard performance library, using Synopsys Design Vision. We compared the increase in area and drop in maximum operating frequency (due to addition of 2×32 extra two input AND gates) with a IGF multiplier synthesized using same parameters. The comparison is presented in table 1. Since these results are not post-layout results, they are not accurate, but are indicative of the fact that increase in hardware complexity of the IGF multiplier for enabling $2w \times w$ multiplication, is marginal.

4.2 Performance Improvement of Reduction over NIST Curves

Table 2 lists the NIST recommended irreducible polynomials over binary fields of different orders. We evaluate the improvement in performance of reduction operation over these fields using a 32-bit MIGF multiplier. Note that each of the polynomials adhere to the restriction $deg(P(x)-x^m) < \frac{m}{2}$. Therefore only two iterations of multiplications are sufficient for completion of the reduction operation.

Let us consider the first polynomial in the list $x^{163}+x^7+x^6+x^3+1$. Since $deg(P(x)-x^{163}) < 31$, all the terms in $P'(x)$ fit within one 32-bit word. Below we evaluate the number of operations involved in the two iterations of reduction using the aforementioned polynomial.

- Iteration One:
  $P'(x) = x^7+x^6+x^3+1$
  Number of words in $C_{h,0}(x)$ is $m_c = \lceil \frac{163}{32} \rceil = 6$
  Number of words in $P'(x)$ is $m_p = \lceil \frac{7}{32} \rceil = 1$
  Number of terms present in $P'(x)$ other than 1 is $p = 3$
  Total number of terms present in $P'(x)$ is $p' = 4$
  $\#SHIFT = 2m_c p = 36$
  $\#XOR = (m_c-1)p + (m_c+1)(p'-1) = 36$
  $\#Multiplications$ using a conventional Polynomial Multiplier is $m_c = 6$
  $\#XOR$ using a conventional Polynomial Multiplier is $(m_c-1) = 5$
  $\#Multiplications$ using an MIGF Multiplier is given by $m_c+1 = 7$
- Iteration Two:
  Number of words in $C_{h,1}(x)$ is $m_c = \lceil \frac{7}{32} \rceil = 1$
  Number of words in $P'(x)$ is $m_p = \lceil \frac{7}{32} \rceil = 1$
  $\#SHIFT = 2m_c p = 6$
  $\#XOR = (m_c-1)p + (m_c+1)(p'-1) = 6$
  $\#Multiplications$ using a conventional Polynomial Multiplier is $m_c = 1$
  $\#XOR$ using a conventional Polynomial Multiplier is $(m_c-1) = 0$
  $\#Multiplications$ using an MIGF Multiplier is given by $m_c+1 = 2$

Now let us consider the second polynomial $x^{233}+x^{74}+1$ from table 2. Clearly $(P(x)-x^{233})$ spans multiple words in a 32-bit environment. Below we evaluate the number of operations involved in the two iterations of reduction using the aforementioned polynomial.

- Iteration One:
  $P'(x) = x^{74}+1$
  Number of words in $C_{h,0}(x)$ is $m_c = \lceil \frac{233}{32} \rceil = 8$
  Number of words in $P'(x)$ with at least one non-zero term is $m_p = 2$
  Number of distinct terms present in $P'(x)$ other than 1 is $p = 1$
  Total number of terms present in individual words of $P'(x)$ is $p'_0 = 1$ and $p'_1 = 1$
  $\#SHIFT = 2m_c p = 16$
  $\#XOR = (m_c-1)p + \Sigma_j (m_c+1)(p'_j-1) = 14$
  $\#Multiplications$ using a conventional Polynomial Multiplier is $m_c m_p = 16$
  $\#XOR$ using a conventional Polynomial Multiplier is $(m_c-1)m_p = 14$
  $\#Multiplications$ using an MIGF Multiplier is given by $(m_c+1)m_p = 18$
- Iteration Two:
  Number of words in $C_{h,1}(x)$ is $m_c = \lceil \frac{74}{32} \rceil = 3$
  Number of words in $P'(x)$ with at least one non-zero term is $m_p = 2$
  $\#SHIFT = 2m_c p = 6$
  $\#XOR = (m_c-1)p + \Sigma_j (m_c+1)(p'_j-1) = 2$
  $\#Multiplications$ using a conventional Polynomial Multiplier is $m_c m_p = 6$
  $\#XOR$ using a conventional Polynomial Multiplier is $(m_c-1)m_p = 4$
  $\#Multiplications$ using an MIGF Multiplier is given by $(m_c+1)m_p = 8$

Table 3: Number of Operations involved in reduction over various NIST curves using three different techniques.

Size		ITRERATION I		ITERATION II	
163	SW	#SHIFT	36	#SHIFT	6
		#XOR	36	#XOR	6
	POLY	#MULT	6	#MULT	1
		#XOR	5	#XOR	0
	MIGF	#MULT	7	#MULT	2
233	SW	#SHIFT	16	#SHIFT	6
		#XOR	6	#XOR	2
	POLY	#MULT	16	#MULT	6
		#XOR	14	#XOR	4
	MIGF	#MULT	18	#MULT	8
283	SW	#SHIFT	54	#SHIFT	6
		#XOR	54	#XOR	6
	POLY	#MULT	9	#MULT	1
		#XOR	8	#XOR	0
	MIGF	#MULT	10	#MULT	2
409	SW	#SHIFT	26	#SHIFT	6
		#XOR	12	#XOR	2
	POLY	#MULT	26	#MULT	6
		#XOR	24	#XOR	4
	MIGF	#MULT	28	#MULT	8
571	SW	#SHIFT	108	#SHIFT	6
		#XOR	108	#XOR	6
	POLY	#MULT	18	#MULT	1
		#XOR	17	#XOR	0
	MIGF	#MULT	19	#MULT	2

Similarly we evaluated the number of operations in each of the iterations of reduction operation over the NIST curves. The numbers of operations are listed in table 3. The fields SW, POLY and MIGF refer to implementation of reduction using pure software algorithm, a conventional polynomial multiplier and an MIGF multiplier respectively. From table 3 it is evident that total number of operations is the least when using an MIGF multiplier as hardware accelerator for reduction.

We present the instruction count in reduction over the various NIST curves using the three different techniques in figure 3. It is evident form figure 3 that the advantage of using hardware assists for reduc-

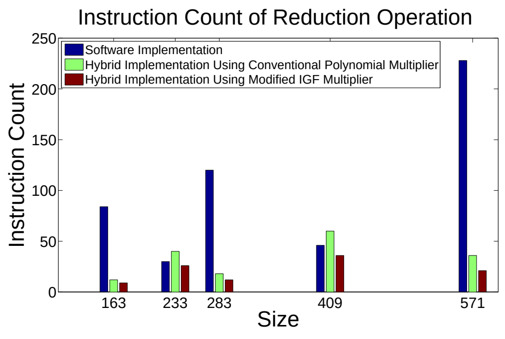

Figure 3: Instruction count of Reduction Operation in three different implementations.

tion is prominent when the there are large number of terms present in the irreducible polynomial. This is attributed to the fact that hardware assists in the form of multipliers combine a number of shift and XOR operations into a single multiplication. If there are less number of terms present in $P'(x)$, the effects of this combination is less prominent. In fact, as in the case of the two polynomials $x^{409}+x^{87}+1$ and $x^{233}+x^{74}+1$, the hybrid technique using a polynomial multiplier may perform worse than a simple software realization for certain irreducible polynomials. However, the absence of XOR operations in forming the partial products makes the proposed technique (using an MIGF multiplier) perform better than both the other techniques, for all the NIST polynomials.

5 CONCLUSIONS

In the context of efficient realization of elliptic Curve Cryptography algorithms, we recognized the importance of an efficient and flexible solution for reduction operations over binary fields. In this paper we presented a method for flexible reduction. The method is especially suitable for coarse-grained platforms where, granularity of data and operations play a major role in the computations. We identified that efficiency of the underlying polynomial multiplication operations determines the speed of reduction algorithms like the Repeated Multiplication Reduction method or the Barrett Reduction method. In this context we proposed a design of a polynomial multiplier based on the well-known Interleaved Galois Field (IGF) multiplier. This MIGF multiplier is shown to achieve a significant improvement in throughput over a purely software realization or a hybrid implementation using a conventional polynomial multiplier.

REFERENCES

Barrett, P. (1987). Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In Odlyzko, A., editor, *Advances in Cryptology CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer Berlin / Heidelberg. 10.1007/3-540-47721-7_24.

Eberle, H., Gura, N., Shantz, S. C., and Gupta, V. (2003). A cryptographic processor for arbitrary elliptic curves over GF(2^m). Technical report, Mountain View, CA, USA.

Hinkelmann, H., Zipf, P., Li, J., Liu, G., and Glesner, M. (2009). On the design of reconfigurable multipliers for integer and galois field multiplication. *Microprocessors and Microsystems - Embedded Hardware Design*, 33(1):2–12.

Karatsuba, A. and Ofman, Y. (1963). Multiplication of multidigit numbers on automata. *Soviet Physics—Doklady*, 7(7):595–596.

Knezevic, M., Sakiyama, K., Fan, J., and Verbauwhede, I. (2008). Modular reduction in GF(2^n) without precomputational phase. In von zur Gathen, J., Imaña, J. L., and Çetin Kaya Koç, editors, *WAIFI*, volume 5130 of *Lecture Notes in Computer Science*, pages 77–87. Springer.

Peter, S., Langendörfer, P., and Piotrowski, K. (2007). Flexible hardware reduction for elliptic curve cryptography in GF(2^m). In Lauwereins, R. and Madsen, J., editors, *DATE*, pages 1259–1264. ACM.

Saqib, N. A., Rodriguez-Henriquez, F., and Diaz-Pirez, A. (2004). A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^m). *Parallel and Distributed Processing Symposium, International*, 4:144a.

Satoh, A. and Takano, K. (2003). A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52:449–460.