Bit-Serial and Digit-Serial GF(2^m) Montgomery Multipliers using Linear Feedback Shift Registers

M. Morales-Sandoval[⋄,1], C. Feregrino-Uribe[2,*]and P. Kitsos[3,†]

[⋄]Embedded and Distributed Systems Group
Polytechnic University of Victoria, Tamaulipas, Mexico.

[1]mmoraless@upv.edu.mx, [2]cferegrino@inaoep.mx,[3]pkitsos@ieee.org

Abstract

This work presents novel multipliers for Montgomery multiplication defined on binary fields GF(2^m). Different to state of the art Montgomery multipliers, this work uses a Linear Feedback Shift Register (LFSR) as the main building block. We studied different architectures for bit-serial and digit-serial Montgomery multipliers using the LFSR and the Montgomery factors x^m and x^{m-1}. The proposed multipliers are for different classes of irreducible polynomials: general, all one polynomials (AOP), pentanomials and trinomials. The results show that the use of LFSRs simplifies the design of the multipliers architecture **reducing area resources and retaining** high performance compared to related works.

1 Introduction

The theory of finite fields is a branch of modern algebra that has come to the fore in recent years mainly due to their importance in several areas such as information theory, algebraic coding theory, number theory, and cryptography, particularly Public key cryptography [1]. Most of the cryptographic algorithms used for ensuring information security services like integrity and confidentiality are based on arithmetic in finite fields. The efficient implementation of this arithmetic impacts directly on efficiency of these cryptographic algorithms.

The finite field GF(2^m) is predominantly used because a field addition operation can be readily implemented with a logic gate. However, field multiplication is more time demanding and usually it is the bottleneck of cryptographic algorithms. This fact has motivated its efficient implementation either in software or hardware. In 1985, Peter L. Montgomery published a method well suited for multiplication in prime

[*] C. Feregrino is with the FPGAs Laboratory, Computer Science Department at National Institute for Astrophysics, Optics and Electronics (INAOE), Puebla, Mexico

[†] P. Kitsos is with the Digital Systems & Media Computing Laboratory, School of Science & Technology at the Hellenic Open University, Patras, Greece

field GF(p) that avoided division by p [2]. Later, Montgomery's method was extended for field multiplication in GF(2^m) [3].

Different Montgomery multipliers have been constructed using different implementation techniques, like systolic or semi-systolic arrays [4, 5]. An important feature for optimizing a Montgomery multiplier in GF(2^m) is the type of the irreducible polynomial used, which could be an all one polynomial (AOP), a trinomial or a pentanomial. Montgomery algorithms have been implemented in three different versions: bit-serial, digit-serial and bit-parallel. Bit-serial multipliers have the smallest area requirements but they are the slowest implementations. On the contrary, bit-parallel multipliers are the fastest circuits but they have the highest area requirements. Digit-serial multipliers allow to explore area-performance trade-offs by processing more than one bit per clock cycle.

Different to the approaches previously addressed in the literature [4], [5], [6], [7], [8], [9], [10], [11], this work describes hardware architectures of new Montgomery multipliers for arbitrary finite fields of the form GF(2^m) where the main component is a Linear Feedback Shift Register (LFSR) [12]. The objective of using an LFSR is to reduce the time complexity of Montgomery multiplier, improving both the latency and the critical path delay. The LFSR allows to design digit-serial multipliers for exploring area-performance trade-offs, it is identified from a new formulation of Montgomery multiplication. First, we study a new bit-serial algorithm and its hardware architecture, which has many differences from the work reported in [6]. Then, generalizing the idea of the LFSR, we design a parallel linear feedback shift register (PLFSR), that processes D bits at a time and allows to implement a new digit-serial multiplication architecture which is different from the architectures previously proposed in [5, 11]. This digit-serial multiplier architecture can be used to implement a new GF(2^m) bit-parallel multiplier using $D = m$.

The bit-serial and digit-serial architectures are designed for general irreducible polynomials $f(x)$ and optimized for special classes like AOP, trinomials and pentanomials. The results presented in this article show that the use of LFSR for constructing multipliers results in efficient architectures of Montgomery multiplication.

The rest of this paper is organized as follows: next section overviews Montgomery algorithm and linear feedback shift registers, sections 3 and 4 describe the design of the new bit-serial and digit-serial Montgomery multipliers respectively. The results and comparison of the proposed Montgomery multipliers in terms of area usage and critical path delay are presented and discussed in section 5. Finally, the conclusions are pointed out at section 6.

Algorithm 1 Montgomery Algorithm for GF(2^m) multiplication

```
Require: $A(x), B(x), R(x), \tilde{f}(x) \in \mathrm{GF}(2^m)$, $f(x)$ an irreducible $m$-grade polynomial
Ensure: $C(x) = A(x)B(x)R^{-1}(x) \bmod f(x)$
  $T(x) = A(x)B(x)$
  $U(x) = T(x)\tilde{f}(x) \bmod R(x)$
  $C(x) = \dfrac{(T(x) + U(x)f(x))}{R(x)} \bmod f(x)$
```

2 Preliminaries

2.1 $GF(2^m)$ Montgomery Multiplication

The arithmetic operations in $GF(2^m)$ are well suited to be implemented in hardware using polynomial basis. In such a representation, each element $e \in \mathrm{GF}(2^m)$ corresponds to a binary polynomial $e(x)$ of degree less than m, that is, $e(x) = e_{m-1}x^{m-1} + \cdots + e_1x + e_0$ with $e_i \in \{1, 0\}$. The element e is usually denoted by the bit-vector $(e_{m-1}, e_{m-2}, \cdots, e_1, e_0)$ of length m.

The GF(2^m) Montgomery multiplication [3] of $A(x)$ and $B(x)$ (see algorithm 1) is defined as $A(x) \times B(x) \times R^{-1}(x) \bmod f(x)$, where $f(x)$ is an irreducible polynomial that generates the field GF(2^m) and $R(x)$ is a fixed field element in GF(2^m). The field element $R^{-1}(x)$ denotes the multiplicative inverse of the element $R(x) \in \mathrm{GF}(2^m)$. Efficient Montgomery multipliers over GF(2^m) can be obtained if $R(x)$ is selected as $R(x) = x^m$ (in case of prime fields GF(p), $R = 2^n$, where n is the size in bits of the prime p) [3]. Recent work [6] has shown that better multipliers could be obtained using $R(x) = x^{m-1}$ as Montgomery factor.

Based on the fact that $f(x)$ and $R(x)$ are relatively prime, two polynomials $R^{-1}(x)$ and $\tilde{f}(x)$ exist with the characteristic that $R(x)R^{-1}(x) + \tilde{f}(x)f(x) = 1$. Polynomials $R^{-1}(x)$ and $\tilde{f}(x)$ can be computed using the Extended Euclidean Algorithm [3].

In algorithm 1, the computation of $C(x)$ involves a regular multiplication in step 1, a modulo $R(x)$ multiplication in step 2, and finally a regular multiplication and a division by $R(x)$ operation in step 3. The modular multiplication and division operations in steps 2 and 3 are intrinsically fast operations since $R(x) = x^m$:

1. The remainder operation in modular multiplication using modulus x^m is accomplished by simply ignoring the terms which have powers of x larger than or equal to m.

2. Division of an arbitrary polynomial by x^m is accomplished by shifting the polynomial to the right by m places.

The computing of $\tilde{f}(x)$ constitutes an overhead for computing $C(x)$. The computation of $\tilde{f}(x)$ can be avoided if the coefficients of $A(x)$ are scanned one bit at a time. In case that $A(x)$ is parsed by words (in a digit-serial implementation), it would be only necessary to compute the least significant word $\tilde{f}_0(x)$ instead of the whole $\tilde{f}(x)$ [3].

2.2 Linear feedback shift register

An LFSR [12] is an n-bit shift register that pseudo-randomly scrolls between 2^n-1 states at high speed, due to the minimal combinational logic involved, to generate binary sequences. Once all the states are reached, the output sequence is repeated, entering in a repeating cycle.

An LFSR of length n has n memory cells which together form the initial state $(s_0, s_1, \cdots s_{n-1})$ of the shift register. The input bit for the LFSR is a linear function of its current (or previous) state, and as the only linear function of single bits is XOR and inverse-XOR, so, the shift register is driven by the XOR of some bits of the overall shift register value. The selection of those bits are represented by a polynomial or characteristic polynomial over {0,1}. That is, if the input bit for the LFSR is a linear function of bits s_0, s_1 and s_{n-1}, then the characteristic polynomial of LFSR is $f(x) = 1 + x + x^{n-1}$. That is why any LFSR can be represented as a polynomial of variable x. In finite fields, this polynomial must be irreducible, that is, all its coefficients are relatively prime.

LFSRs perform fast in hardware, mainly in VLSI implementations. Also, LFSRs designed to work in parallel can be used for applications that include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. In this work, we use Type-II LFSR [13] to perform Montgomery Multiplication.

3 New bit-serial Montgomery multiplier

Consider the field elements $A(x), B(x), C(x) \in \mathrm{GF}(2^m)$. Using the definition of a GF(2^m) field element, we have that:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0) \tag{1}$$

$$B(x) = \sum_{i=0}^{m-1} b_i x^i = (b_{m-1}, b_{m-2}, \cdots, b_1, b_0) \tag{2}$$

$$C(x) = \sum_{i=0}^{m-1} c_i x^i = (c_{m-1}, c_{m-2}, \cdots, c_1, c_0) \tag{3}$$

The irreducible polynomial $f(x)$ generating GF(2^m) is:

$$f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i = (1, f_{m-1}, f_{m-2}, \cdots, f_1, 1) \tag{4}$$

Then, the operation $C(x) = A(x) \times B(x) \times R^{-1}(x) \bmod f(x)$, using factor $R(x) = x^m$ can be expressed as follows:

$$\begin{aligned} C(x) &= A(x) \times B(x) \times R^{-1}(x) \bmod f(x) \\ &= x^{-m} \sum_{i=0}^{m-1} a_i x^i B(x) \bmod f(x) \\ &= x^{-m} \times \left[a_{m-1}x^{m-1} + \cdots + a_1 x + a_0\right] \times B(x) \\ &= \left[a_{m-1}x^{-1} + \cdots + a_1 x^{-m+1} + a_0 x^{-m}\right] \times B(x) \\ &= a_{m-1}\frac{B(x)}{x} \bmod f(x) + \cdots \\ &\quad \cdots + a_1 \frac{B(x)}{x^{m-1}} \bmod f(x) + a_0 \frac{B(x)}{x^m} \bmod f(x) \end{aligned} \tag{5}$$

Let $B(x)^{(0)} = B(x)$, and $B(x)^{(i)} = \dfrac{B(x)^{(i-1)}}{x}$ mod $f(x)$, for $1 \le i \le m$. Then $C(x) = A(x)B(x)x^{-m}$ mod $f(x)$ becomes equation 6:

$$\begin{aligned} C(x) &= A(x) \times B(x) \times x^{-m} \bmod f(x) \\ &= a_{m-1}B(x)^{(1)} + a_{m-2}B(x)^{(2)} + \cdots \\ &\quad \cdots + a_1 B(x)^{(m-1)} + a_0 B(x)^{(m)} \end{aligned} \tag{6}$$

If the factor $R(x) = x^{m-1}$ is used, then $R^{-1}(x) = x^{-m+1} = x^{-m}x$. The Mongomery multiplication using this factor becomes $C(x) = A(x)B(x)x^{-m}x$ mod $f(x)$, that implies a multiplication of each term of equation 5 by x. Using the notation $B(x)^{(i)}$, the resulting expression is as shown in equation 7:

$$\begin{aligned} C(x) &= A(x) \times B(x) \times x^{-m}x \bmod f(x) \\ &= a_{m-1}B(x)^{(0)} + a_{m-2}B(x)^{(1)} + \cdots \\ &\quad \cdots + a_1 B(x)^{(m-2)} + a_0 B(x)^{(m-1)} \end{aligned} \tag{7}$$

Equation 7 uses the initial value $B(x)^{(0)}$ and Equation 6 does not. Equation 6 needs to compute $B(x)^{(m)}$ and equation 7 does not. At first glance, it seems that equation 7 requires one less iteration, but it is not true because all the bits from $A(x)$ are parsed and used in both equations. This means that the time complexity is the same for both equations.

The term $\dfrac{B(x)}{x}$ mod $f(x)$ represents a division modulo $f(x)$ of polynomial $B(x)$ by x. If [illegible] 0, $B(x)$ is divisible by x and $\dfrac{B(x)}{x}$ mod $f(x)$ is interpreted as a decreasing of the degree of polynomial $B(x)$ by one

Table 1: Montgomery multiplication using LFSR and factors $R(x) = x^m$ and $R(x) = x^{m-1}$

Clock		$R(x) = x^{m-1}$	$R(x) = x^m$	
cycle	**LFSR**	$C(x)$	$C(x)$	C(x)
0	$B(x)^{(0)}$	0	0	0
1	$B(x)^{(1)}$	$C(x) + a_{m-1}B(x)^{(0)}$	$C(x) + a_{m-1}B(x)^{(0)}/x$	0
2	$B(x)^{(2)}$	$C(x) + a_{m-2}B(x)^{(1)}$	$C(x) + a_{m-2}B(x)^{(1)}/x$	$C(x) + a_{m-1}B(x)^{(1)}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
m	$B(x)^{(m)}$	$C(x) + a_0B(x)^{(m-1)}$	$C(x) + a_0B(x)^{(m-1)}/x$	$C(x) + a_1B(x)^{(m-1)}$
$m+1$	$B(x)^{(m+1)}$	$-$	-	$C(x) + a_0B(x)^{(m)}$

[14]. This results in a simple shift to right operation of $B(x)$ by one position. In case $b_0 = 1$, division is performed modulo $f(x)$, that is, $\frac{[B(x) + f(x)]}{x}$ is computed instead of $\frac{B(x)}{x}$. The resulting field element $[B(x) + f(x)] = (1, b_{m-1} \oplus f_{m-1}, ..., b_1 \oplus f_1, b_0 \oplus f_0)$ is now divisible by x because $f_0 = 1$ and $b_0 = 1$. So, the division operation of a polynomial $B(x)$ by x modulo $f(x)$ is defined as $(0, b_{m-1}, b_{m-2}, \cdots, b_2, b_1)$ if $b_0 = 0$ or $(1, b_{m-1} \oplus f_{m-1}, b_{m-2} \oplus f_{m-2}, \cdots, b_2 \oplus f_2, b_1 \oplus f_1)$ if $b_0 = 1$.

The operation $\frac{B(x)}{x}$ mod $f(x)$ can be generalized taking into account the value of b_0, that is:

$$\frac{B(x)}{x} \bmod f(x) = (b_0, b_{m-1} \oplus b_0 f_{m-1}, \cdots, b_1 \oplus b_0 f_1) \tag{8}$$

Equation 8 is well modeled by a Linear Feedback Shift Register (LFSR) B with m bits [15]. The definition of such LFSR B (in binary representation: $b_{m-1}, b_{m-2}, \cdots, b_1, b_0$) is:

$$b_{m-1}(i + 1) = b_0(i)$$
$$b_j(i + 1) = b_{j+1}(i) \oplus b_0(i)f_j$$

where $b_j(i)$ denotes the content of the bit j of B at clock cycle i.

A general diagram of this LFSR with all taps (XOR) and switches in order to configure the characteristic polynomial $f(x)$ is shown in figure 1 a). In most cases, the characteristic polynomial $f(x)$ is a trinomial or a pentanomial, so most of the AND gates in figure 1 could not be needed.

In figure 1, from the current state of the LFSR $B(x)^{(i)}$ stored in b_i 1-bit registers, the next state $B(x)^{(i+1)}$ is computed by the AND and XOR gates. By separating this combinatorial logic from the sequential logic, the LFSR could be re-arranged as it is shown in figure 1 b).

According to equations 6 and 7, table 1 shows the realization of Montgomery multipliers using factor $R(x) = x^{m-1}$ and $R(x) = x^m$ using the LFSR in figure 1. The main difference in these multipliers is where the output of the LFSR is taken from. This output could be taken from the registers b_i ($B(x)^{(i)}$) or from the interconnection matrix ($B(x)^{(i)}/x$). This last choice is slower because the critical path is increased due the combinatorial logic of the interconnection matrix. This implies that the best implementation of the bit-serial Montgomery multiplier is using factor $R(x) = x^{m-1}$. The circuit of this multiplier is shown in figure 2. The LFSR B is initialized with $B(x) = B(x)^{(0)}$ what implies a parallel write for all bits of B. At each iteration i ($1 \leq i \leq m$), the LFSR computes $B(x)^{(i)} = B(x)^{(i-1)}/x$ and a bit from $A(x)$ (a_{m-i}) is read for computing $C(x) = C(x) + a_i B(x)^{(i-1)}$. After m clock cycles, the multiplication $A(x)B(x)x^{-(m-1)} \bmod f(x)$ is finally performed. As it is shown in figure 2, the critical path in the circuit is $T_X + T_A$, being T_X the time delay of an XOR gate and T_A the time delay of an AND gate.

The circuit in figure 1 could implement the Montgomery multiplier using factor x^m by just delaying the multiplication one clock cycle (clock cycle no. 1), as it is shown in the fifth column of table 1. The resulting area complexity and critical path is the same but the latency increases to $m+1$. Finally, the same circuit in figure 1 could implement the Montgomery multiplier using factor x^m with latency of m clock cycles if the output of the LFSR is taken from the interconnection matrix (from XOR's of the LFSR in figure 1). The area complexity will be exactly the same but the critical path will increase to $2(T_X + T_A)$.

The latency for computing a Montgomery multiplication could be reduced if a word $A_i(x)$ from $A(x)$ is processed at each clock cycle instead of a single bit a_i. Next section discusses the design of a digit-serial Montgomery multiplier, that processes D bits from $A(x)$ per clock cycle and reduces the latency of multiplication to $\lceil m/D \rceil$ at the cost of more hardware resources.

4 Digit-serial Montgomery Multiplier

The main idea in a digit-serial multiplier (word level) is to process a group of D bits from $A(x)$ instead of one bit at each clock cycle. The word level description of the polynomial $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0$ implies a partition of $A(x)$ into blocks of equal length. Let D be the size of these blocks such that $A(x)$ has s blocks, $s = \lceil m/D \rceil$. Thus, $A(x) = A_0(x) + A_1(x) + \cdots + A_{s-2}(x) + A_{s-1}(x)$, where each $A_i(x)$ is of length D and defined as in equation 9.

$$\begin{aligned} A_i(x) \quad = \quad & a_{m-iD-1}x^{m-iD-1} \\ & + a_{m-iD-2}x^{m-iD-2} \\ & + \dots \\ & + a_{m-iD-D-1}x^{m-iD-D-1} \\ & + a_{m-iD-D}x^{m-iD-D} \end{aligned} \tag{9}$$

Using the word level representation of $A(x)$, we can express $C(x) = A(x) \times B(x) \times R^{-1}(x) \bmod f(x)$ as

$$C(x) = \sum_{i=0}^{s-1} A_i(x) \times B(x) \times R^{-1}(x) \bmod f(x) \tag{10}$$

Let $C_i(x) = A_i(x)B(x)R^{-1}(x) \bmod f(x)$ and $R(x) = x^m$. Equation 11 defines $C_i(x)$:

$$\begin{aligned} C_i(x) &= \frac{a_{m-iD-1}x^{m-iD-1}B(x)}{x^m} \\ &\quad + \frac{a_{m-iD-2}x^{m-iD-2}B(x)}{x^m} \\ &\quad + \dots \\ &\quad + \frac{a_{m-iD-D}x^{m-iD-D}B(x)}{x^m} \\ &= a_{m-iD-1}\frac{B(x)}{x^{iD+1}} \\ &\quad + a_{m-iD-2}\frac{B(x)}{x^{iD+2}} \\ &\quad + \dots \\ &\quad + a_{m-iD-D}\frac{B(x)}{x^{iD+D}} \\ &= a_{m-iD-1}B(x)^{(iD+1)} \\ &\quad + a_{m-iD-2}B(x)^{(iD+1)} \\ &\quad + \dots \\ &\quad + a_{m-iD-D}B(x)^{(iD+D)} \end{aligned} \tag{11}$$

According to the last expression of $A_i(x) \times B(x) \times x^{-m} \bmod f(x)$ in equation 11, j $(1 \leq j \leq D)$ consecutive outputs of the LFSR are processed instead of a single one. Each output $B(x)^{(iD+j)}$ is multiplied by the bit a_{m-iD-j} from $A_i(x)$. As in the serial implementation, this multiplication is implemented by ANDing each bit value of polynomial $B(x)^{(iD+j)}$ with the bit a_{m-iD-j}. At each clock cycle i $(0 \leq i \leq s-1)$ j multiplications $a_{m-iD-j}B(x)^{(iD+j)}$ are performed in parallel and added all together to get $C_i(x) = A_i(x)B(x)x^{-m} \bmod f(x)$.

If the factor $R(x) = x^{m-1}$ is used, then the definition of $C_i(x)$ is:

$$
\begin{aligned}
C_i(x) &= \frac{a_{m-iD-1}x^{m-iD-1}B(x)}{x^{(m-1)}} \\
&\quad + \frac{a_{m-iD-2}x^{m-iD-2}B(x)}{x^{(m-1)}} \\
&\quad + \dots \\
&\quad + \frac{a_{m-iD-D}x^{m-iD-D}B(x)}{x^{(m-1)}} \\
&= a_{m-iD-1}\frac{B(x)}{x^{iD}} \\
&\quad + a_{m-iD-2}\frac{B(x)}{x^{iD+1}} \\
&\quad + \dots \\
&\quad + a_{m-iD-D}\frac{B(x)}{x^{iD+D-1}} \\
&= a_{m-iD-1}B(x)^{(iD)} \\
&\quad + a_{m-iD-2}B(x)^{(iD+1)} \\
&\quad + \dots \\
&\quad + a_{m-iD-D}B(x)^{(iD+D-1)}
\end{aligned}
\tag{12}
$$

Again, as in the case of the bit-serial multiplier, the use of factor $R(x) = x^{(m-1)}$ implies to take the first output $B(x)^{(iD)}$ in equation 12 from the register B instead of the first output $B(x)^{(iD+1)}$ from the interconnection matrix.

The hardware architecture of the digit-serial Montgomery multiplier is built using the circuit b) of LFSR in figure 1. The interconnection matrix computes $B(x)^{(i+1)}$ from $B(x)^{(i)}$. By replicating this logic, and connecting the output $B(x)^{(i+1)}$ to it, it could be obtained $B(x)^{(i+2)}$ in the same clock cycle. Generalizing this idea, D consecutive outputs of the LFSR could be obtained in one clock cycle by connecting in cascade D consecutive interconnection matrix modules. Figure 3 shows the block diagram of digit-serial Montgomery multipliers using $R(x) = x^m$ and $R(x) = x^{m-1}$. The word $A_i(x) = (a_{m-iD-1}, a_{m-iD-2}, \cdots, a_{m-iD-D})$ could be obtained by a D-bit shift register. Each bit a_{m-iD-j} is multiplied by the corresponding polynomial $B(x)^{(i+j)}$, $(1 \leq j \leq D)$ and all these partial multiplications are added to get $C_i(x)$. After $s = \lceil m/D \rceil$ clock cycles the whole multiplication $A(x) \times B(x) \times R^{-1}(x) \bmod f(x)$ is finally computed.

From figure 3, it is clear that both architectures for digit-serial Montgomery multiplication have exactly the same area complexity. The critical path is shorter when $R(x) = x^{m-1}$ because the last interconnection matrix is not used in the accumulative sum of $C_i(x)$. The path delay of the digit-serial multiplier is the one obtained by adding the delay of the replicating logic of the interconnection matrix and the delay of the logic for the accumulative multiplicative-adding operation of each $B(x)^{(iD+j)}$ together with the bit-value a_{m-iD+j} $(1 \leq j \leq D)$.

Table 2: Area and time complexity of bit-serial Montgomery multiplier

$f(x)$	General		AOP		Pentanomial		Trinomial	
Factor	x^m	x^{m-1}	x^m	x^{m-1}	x^m	x^{m-1}	x^m	x^{m-1}
1-bit FF	$2m$	$2m$	$2m$	$2m$	$2m$	$2m$	$2m$	$2m$
2-in AND	$2m$-1	$2m$-1	m	m	m	m	m	m
2-in XOR	$2m$-1	$2m$-1	$2m$-1	$2m$-1	$m+3$	$m+3$	$m+1$	$m+1$
Latency	$m+1$ $[m]$	m	$m+1$ $[m]$	m	$m+1$ $[m]$	m	$m+1$ $[m]$	m
Delay	T_A+T_X $[2(T_A+T_X)]$	T_A+T_X	T_A+T_X $[T_A+2T_X]$	T_A+T_X	T_A+T_X $[T_A+2T_X]$	T_A+T_X	T_A+T_X $[T_A+2T_X]$	T_A+T_X

T_X: Time delay of an XOR gate
T_A: Time delay of an AND gate

Table 3: Area and time complexity of digit-serial Montgomery multiplier

$f(x)$	General		AOP		Trinomials	
Factor	x^m	x^{m-1}	x^m	x^{m-1}	x^m	x^{m-1}
1-bit FF	$2m$	$2m$	$2m$	$2m$	$2m$	$2m$
2-in AND	$D(2m-1)$	$D(2m-1)$	Dm	Dm	Dm	Dm
2-in XOR	$D(2m-1)$	$D(2m-1)$	$D(2m-1)$	$D(2m-1)$	$D(m+1)$	$D(m+1)$
Latency	$\lceil m/D \rceil$	$\lceil m/D \rceil$	$\lceil m/D \rceil$	$\lceil m/D \rceil$	$\lceil m/D \rceil$	$\lceil m/D \rceil$
Delay	$(D+1)T_A+$ $[D+P]\,T_X$	DT_A+ $[(D-1)+P]\,T_X$	T_A+ $[D+P]\,T_X$	T_A+ $[(D-1)+P]\,T_X$	T_A+ $[1+P]\,T_X$	T_A+ $[1+P]\,T_X$

T_X: Time delay of an XOR gate
T_A: Time delay of an AND gate
P: Depth of a binary tree of D XOR gates = $log(D)$

5 Results

Tables 2 and 3 summarize the time and area complexity of bit-serial and digit-serial multiplier discussed in previous sections using four kinds of irreducible polynomials: general, AOP, trinomials and pentanomials. These results are for both factors $R(x) = x^m$ and $R(x) = x^{m-1}$.

For general irreducible polynomials, the circuit of bit-serial multiplier is the same for both factors x^m and x^{m-1}. The only difference is that it takes one more clock cycle for factor x^m. By just taking the output of the LFSR from the interconnection matrix instead of the registers, the multiplier could be implemented for factor x^m using exactly the same area resources and latency equal to the multiplier with factor x^{m-1} but with an increased critical path from $(T_A + T_X)$ to $2(T_A + T_X)$.

The advantage of using AOP instead of general irreducible polynomial is the reduction of AND gates from $2m-1$ to m. AOP improves the critical path of the multiplier for factor x^m taking the output of LFSR from the interconnection matrix. The use of trinomials or pentanomials reduces the number of ANDs and

XORs. The latency and critical path remains the same as in case of general and AOP polynomials.

In case of the digit-serial multiplier, the improvement in latency is achieved at the expense of higher area resources and an increased critical path. For general and AOP irreducible polynomials, the use of factor x^{m-1} outperforms the multiplier using the factor x^m by decreasing the critical path. But in case of trinomials, there is no difference and both multipliers have the same area and time complexities. Compared to general irreducible polynomials, the use of AOP and trinomials has the effect of reducing area, latency and critical path. In all cases, the critical path of the digit-serial multiplier is obtained by adding the delay of the replicating logic of the interconnection matrix and the delay of the logic for the accumulative multiplicative-adding operation of each $C_i(x)$.

5.1 Comparison

Table 4 shows comparisons of our proposed bit-serial Montgomery multiplier against the best multipliers previously reported in the literature.

Table 4: Bit-serial Montgomery multipliers comparison

Work	**Technique**	**Size**	**#AND**	**#XOR**	**#FF**	**Latency**	$R(x)$	**Critical path delay**
	$f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_1x + 1$ (General polynomial)							
[6]	MSB first	m	$2m-1$	$2m-1$	$2m$	m	x^{m-1}	$T_A + T_X$
						$m+1$	x^m	$T_A + T_X$
[7]	Unified	m	-	-	-	$m+2$	x^m	$T_A + 2T_X + 2T_{MUX} + T_{ADDER}$
[8]	Unified	m	-	-	-	$m+2$	x^m	$T_A + 4T_X + 3T_{MUX}$
	LSB first	m	$2m-1$	$2m-1$	$2m$	$m+1$	x^m	$T_A + 2T_X$
[16]	PCA	m	$3m-1$	$3m-1$	-	m	x^m	$2T_A + 2T_X$
[17]	Systolic	m	$2m^2+3m$	m^2+m	$3m^2+4m$	$m+1$	x^m	$2T_A + 2T_X + T_{LATCH}$
This	LFSR	m	$2m-1$	$2m-1$	$2m$	m	x^{m-1}	$T_A + T_X$
						$m+1$	x^m	$T_A + T_X$
		m	$2m-1$	$2m-1$	$2m$	m	x^m	$T_A + 2T_X$
	$f(x) = x^m + x^k + 1$ (Trinomials)							
[4]	Systolic	m	m^2	$1.5m^2+m$	$4m^2+m$	$m+1$	x^k	$T_A + T_X$
[5]	Systolic	m	m^2	m^2+m-1	$2m^2$	$2m-1$	x^m	$T_A + T_X + T_{LATCH}$
[18]	Systolic	m	m^2	m^2+m	$3.5m^2+3m$	$m+2$	x^k	-
This	LFSR	m	m	$m+1$	$2m$	m	x^{m-1}	$T_A + T_X$
	$f(x) = x^m + x^{m-1} + \cdots + x + 1$ (AOP)							
[4]	Systolic	m	$(m+1)^2$	$(m+1)^2$	$3(m+1)^2$	$m+1$	-	$T_A + T_X$
[9]	LFSR	m	$m+1$	$m+1$	$2m+4$	$2m+1$	-	$T_A + mT_X$
[19]	PCA	m	$m/6$	$2m-1$	$3m$	m	-	$T_{SWITCH} + 2T_X$
This	LFSR	m	m	$2m-1$	$2m$	m	x^{m-1}	$T_A + T_X$

In case of general irreducible polynomials, our multiplier outperforms the multipliers reported in [7], [8], [16] and [17], where techniques used have not considered LFSR. Our results could be equally comparable to that presented by [6]. However, there are some important differences to remark: *i*) we conceive the operation $B(x)^{(i)}$ as an LFSR, they conceive it just like an arithmetic operation, *ii*) before starting the Montgomery multiplication, they require the precomputed value $A(x)x^{-1}$ in one register A' when using factor

x^m. Contrary, we do not need a precomputed value when using x^m, saving one clock cycle, *iii*) our multiplier based on LFSR can operate for both factors x^m and x^{m-1} without any changes in the architecture, they use two different architectures for each one of these factors, *iv*) our bit-serial multiplier based on LFSR is easily adapted for designing digit-serial multipliers and exploring area-time trade-offs. They do not consider digits and, *v*) our approach for digit-serial implementation allows to implement bit-parallel multipliers considering m as the digit size.

For trinomials, in [8] authors proposed an unified multiplier for both GF(p) and GF(2^m) at the costs of higher area requirements and an increased critical path. In [5] and [18], the systolic approach for constructing Montgomery multipliers required more area resources and additional clock cycles.

In [9], authors have used bit-serial LFSR and AOP but their latency is greater as well as the area resources (additionally, $(m+3)$MUX). The work in [4] uses AOP but also the area resources are high, regarding the critical path it is the same as the work presented here. Although the multiplier reported in [19] stated to use only $m/6$ AND gates it uses m additional flip flops and the critical path is greater.

Table 5: **Digit-serial and** Bit-parallel Montgomery multipliers comparison

Work	**Size**	**#AND**	**#XOR**	**#FF**	$R(x)$	**Critical path delay**
	Digit-serial multipliers using Trinomials $f(x) = x^m + x^k + 1$					
[5]	m	D^2	$D^2 + 3D - 1$	$2D^2 + 5\lceil m/D \rceil D - 2\lceil m/D \rceil + D$	x^m	$T_A + T_X + T_L$
[11]	m	$(m+k+1)D + (k+1)(D-1)$	$(m+k)D + (k+1)(D-1)$	$2m + D + k$	x^k	$T_A + [\log(D+1)]\, T_X$
This	m	Dm	$D(m+1)$	$2m$	x^m	$T_A + [\log(D) + 1]\, T_X$
	Bit-parallel multipliers using Trinomials $f(x) = x^m + x^k + 1$					
[6]	m	m^2	$m^2 - 1$	-	x^k	$T_A + [\log(m+k)]\, T_X$
[10]	m	m^2	$m^2 - 1$	$3m$	x^k	$T_A + [\log(m-2) + 2]\, T_X$
This	m	m^2	$m^2 + m$	$2m$	x^{m-1}	$T_A + [\log(m) + 1]\, T_X$
	Bit-parallel multipliers using AOP $f(x) = x^m + x^{m-1} + \cdots + x + 1$					
[4]	m	$(m+1)^2$	$(m+1)^2$	$3(m+1)^2$	-	$(m+1)(T_A + T_X)$
[20]	m	m^2	$m^2 + m - 2$	-	-	$T_A + [\log(m-1) + m]\, T_X$
[21]	m	$m^2 + 2m + 1$	$m^2 + 2m$	-	-	$T_A + [\log(2m) + log(m+2)]\, T_X$
[22]	m	m^2	$m^2 - 1$	-	-	$T_A + [\log(m-1) + 2]\, T_X$
This	m	m^2	$2m^2 - m$	$2m$	x^{m-1}	$T_A + [\log(m) + m - 1]\, T_X$

Table 5 shows comparisons of our proposed digit-serial Montgomery multiplier against representative works. We designed hardware architectures for different kinds of irreducible multipliers. However, in the literature most of the Montgomery multipliers are designed and optimized mainly for trinomials or AOP. For this reason our comparison is limited to these kinds of multipliers. Our digit-serial multiplier has a better area usage compared to the work reported in [5] and [11].

The digit-serial multiplier reported in this work can implement bit-parallel Montgomery multipliers using

$D = m$, performing a Montgomery multiplication in just one clock cycle. Table 5 compares our design against several bit-parallel multipliers. In case of trinomials, we achieve better critical path than the multipliers reported in [6] and [10] at cost of $m + 1$ additional XOR gates. In the case of multipliers defined for AOP, our design exhibits better critical path than [20] and [21], and uses fewer resources than [4].

According to the previous comparisons, the digit-serial Montgomery multiplier presented in this work is the best reported in the literature.

6 Concluding remarks

In this paper, we have presented a new way to realize bit-serial, digit-serial and bit-parallel Montgomery multipliers over GF(2^m) based on Linear Feedback Shift Registers. The proposed multipliers are for several kinds of irreducible polynomials and both Montgomery factors x^m and x^{m-1}. Our designs outperform previous approaches due the low complexity and high performance of the LFSR, which is the main building block in the design. Further, the LFSR allowed to construct digit-serial Montgomery multipliers that could implement bit-parallel multipliers. Our architectures are well suited to VLSI systems because of their regular and modular structures and fully inherent parallelism, and are suitable for many applications, such as Elliptic Curve Cryptography.

References

[1] Menezes, A., Van Oorschot, P., and Vanstone, S.: 'Handbook of Applied Cryptography' (CRC Press, 1997).

[2] Montgomery, P.L.: 'Modular multiplication without trial division', Mathematics of Computation, 1985, 44(170), pp. 519–521.

[3] Koc, C.K., and Acar, T.: 'Montgomery Multiplication in GF(2^k)', Designs, Codes and Cryptography, 1998, 14(1), pp. 57–69.

[4] Lee, C., Horng, J., Jou, I., and Lu, E.: 'Low-complexity bit parallel systolic Montgomery multipliers for special classes of GF(2^m)', Transactions on Computers, 2005, 54(9), pp.1061–1070.

[5] Lee, C.-Y., Chiou, C.-W., Lin, J.-M., and Chang, C.-C.: 'Scalable and systolic Montgomery multiplier over GF(2^m) generated by trinomials', IET Circuits Devices Syst, 2007, 1(6), pp. 477–484.

[6] Hariri, A., and Reyhani-Masoleh, A.: 'Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over GF(2^m)', IEEE Transactions on Computers, 2009, 58(10), pp. 1332–1345.

[7] Sudhakar, M., Kamala, R.V., and Srinivas M.B.: ‘New and improved architectures for Montgomery modular multiplication’, Mob Netw Appl., 2007, 12(4), pp. 281–291.

[8] Sudhakar, M., and Srinivas, M.B.: ‘A unified and reconfigurable Montgomery Multiplier architecture without four-to-two CSA’. Proc. 20th annual Conf. on Integrated circuits and systems design, New York, NY, 2007, pp. 147–152.

[9] Kim, H., and Lee, S.: ‘LFSR multipliers over GF(2^m) defined by all-one polynomial’, Integration, VLSI Journal, 2007(40), pp. 473–478.

[10] Wu, H.: ‘Montgomery Multiplier and Squarer for a class of finite fields’, IEEE Transactions on Computers, 2002, 51(5), pp. 521–529.

[11] Kumar, S., Wollinger, T., and Paar, C.: ‘Optimum Digit-Serial GF(2^m) Multipliers for Curve-Based Cryptography’, IEEE Transactions on Computers, 2006, 55(10), pp. 1306–13011.

[12] Abramovici, M., Breuer, M.A., and Friedman, A.D.: ‘Digital System Testing and Testable Design’ (IEEE Press, 1990).

[13] Katti, R., Ruan, X., and Khattri, H.: ‘Multiple-Output Low-Power Linear Feedback Shift Register Design’, IEEE Transactions on Circuits and Systems-I. 2006, 53(7), pp. 1487–1495.

[14] de Dormale, G., and Quisquater, J.: ‘Iterative Modular Division over GF(2^m): Novel Algorithm and Implementations on FPGA’. Proc. International Workshop on Applied Reconfigurable Computing (ARC2006), 2006, pp. 370–382.

[15] Chiou, C., Lee, C., and Lin, J.: ‘Finite Field Polynomial Multiplier with Linear Feedback Shift Register’, Journal of Science and Engineering, 2007, 10(3), pp. 253–264.

[16] Ku, K.M., Ha, K.J., Yoo, W.H., and Yoo, K.Y.: ‘Parallel Montgomery Multiplication and Squaring over GF(2^m) Based on Cellular Automata’. Proc. of ICCSA 2004, LNCS 3046, 2004, pp. 196–205.

[17] Chiou, C.W., Lee, C.Y., Deng. A.W., and Lin J.M.: ‘Structure of Parallel Multipliers for a Class of Finite Fields GF(2^m)’, Tamkang Journal of Science and Engineering, 2006, 9(4), pp. 65–372.

[18] Lee, C.-Y.: ‘Low-Complexity Parallel Systolic Montgomery Multipliers over GF(2m) Using Toeplitz Matrix-Vector Representation’, IEICE Trans Fundam Electron Commun Comput Sci. 2008, E91-A(6), pp. 1470–1477.

[19] Jeon, J.-C., and Yoo, K.-Y.: ‘Programmable cellular automata based Montgomery hardware architecture’. Applied Mathematics and Computation. 2007, 2007(186), pp. 915–922.

[20] Hasan, M.A., Wang, M.Z., and Bhargava, V.K.: 'Modular Construction of Low Complexity Parallel Multipliers for a Class of Finite Fields GF(2^m)', IEEE Transactions on Computers, 1992, 41(8), pp.962–971.

[21] Itoh, T., and Tsujii, S.: 'Structure of Parallel Multipliers for a Class of Finite Fields GF(2^m)', Information and Computation, 1989, 83, pp.21–40.

[22] Koc, C.K., and Sunar, B.: 'Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields', IEEE Transactions on Computers, 1998, 47(3), pp. 353–356.

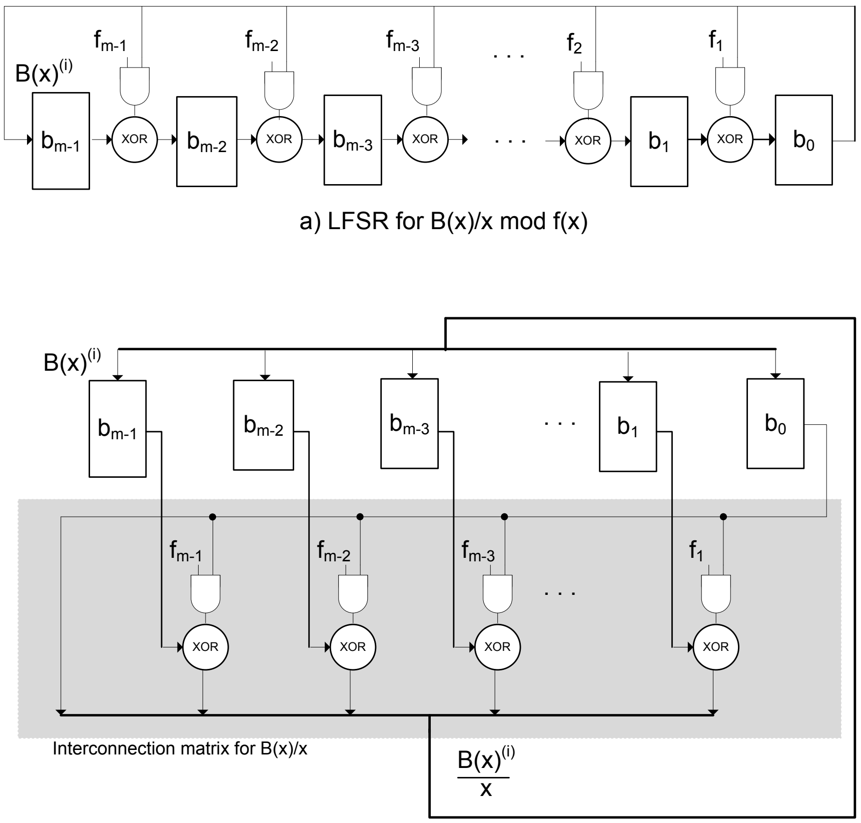
7 Figures and list of figures

List of Figures

1 Linear feedback shift register for computing $\frac{B(x)}{x}$ mod $f(x)$ 16

2 Serial Montgomery multiplier hardware architecture 17

3 Digit-serial Montgomery multiplier using the parallel LFSR for a) $R(x) = x^m$ and b) $R(x) = x^{m-1}$ 17



a) LFSR for B(x)/x mod f(x)

b) Equivalent architecture to a) for B(x)/x mod f(x)

Figure 1: Linear feedback shift register for computing $\dfrac{B(x)}{x} \bmod f(x)$

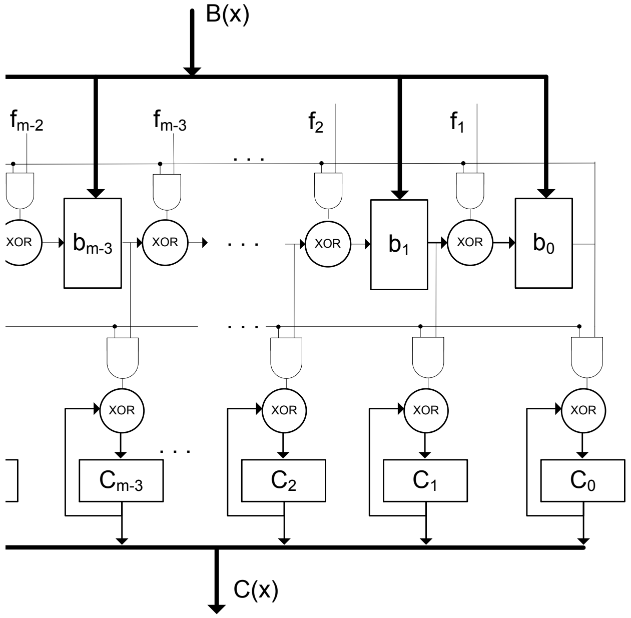


Figure 2: Serial Montgomery multiplier hardware architecture

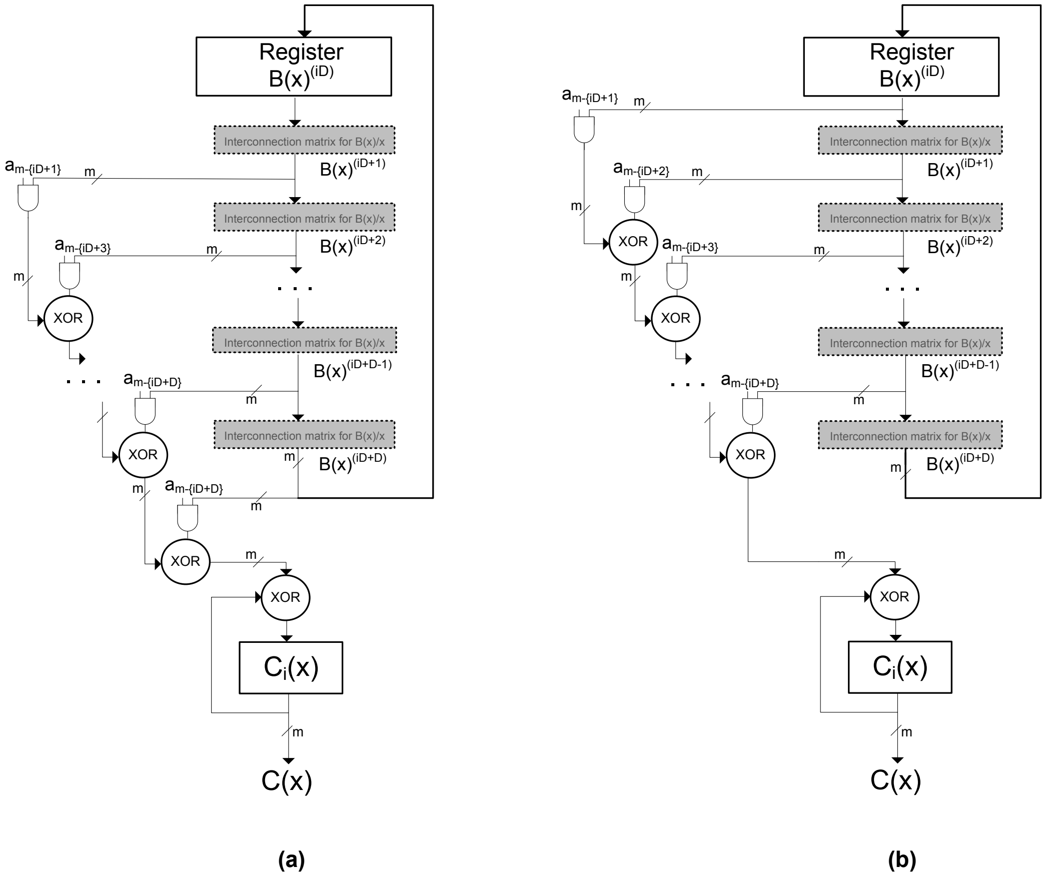


Figure 3: Digit-serial Montgomery multiplier using the parallel LFSR for a) $R(x) = x^m$ and b) $R(x) = x^{m-1}$