Abstract—We present an FPGA implementation of a new multiplier for binary finite fields that combines two previously known methods. The multiplier is designed for polynomial bases which allow more flexibility in hardware and is dedicated to efficient implementations of elliptic curve cryptography. An extension to a digit-serial architecture is also sketched. For the introduced architecture we also discuss resistance to side-channel attacks.

I. INTRODUCTION

Most public key cryptosystems, including Elliptic Curve Cryptosystems (ECC), heavily rely on arithmetic operations in finite fields, and more in particular on a finite field multiplier. This article proposes an efficient FPGA implementation of a new serial multiplier that combines two previously known methods, the classical and Montgomery’s modular multiplication algorithm. We require no special properties for the irreducible polynomial defining the finite field. This design can handle arbitrary bit-lengths from 160 to 300 bits, which are suitable for applications of ECC. Several algorithms and architectures for multiplication in \( GF(2^n) \) have been proposed \cite{4, 23}. In the context of an EC processor for binary fields, multipliers have been discussed in \cite{1, 22}, and recently in \cite{8, 9, 17}. The first bit serial multiplier is discussed by Beth and Gollmann \cite{4}. This multiplier is using convolution and reduction modulo an irreducible polynomial taking \( n \) clock cycles to compute a multiplication. One of the earliest implementations is \cite{1}, wherein the authors described an efficient implementation of ECC over \( GF(2^{155}) \) in optimal normal basis. Although normal basis representations usually result in efficient implementations the major disadvantage is that these bases do not offer a scalable and flexible platform. The implementations in \cite{7, 22} also use normal basis representation. In \cite{17}, a complete processor architecture for elliptic curve cryptosystems over \( GF(2^n) \) in polynomial bases is proposed. The proposed architecture is also scalable with separated squarer (bit-parallel) and multiplier (digit-serial). Goodman and Chandrakasan proposed a domain-specific reconfigurable cryptographic processor (DSRCP) in \cite{8}. The DSRCP performs a variety of algorithms ranging from modular integer arithmetic to elliptic curve arithmetic over finite field. All operations are universal and they can be performed using any \( n \)-bit modulus \((8 \leq n \leq 1024)\), irreducible polynomial and non-supersingular elliptic curve over \( GF(2^n) \). The various complex modular arithmetic operations (multiplication, reduction, inversion and exponentiation) are implemented using microcode, while simple operations (addition and subtraction) are implemented directly in hardware. Multiplication is performed using Montgomery multiplication \cite{16}. The multiplier is doing a bit-serial processing, so this design would be suitable for low power devices. The work of Kitsos et al \cite{20} presented the MSB-first, bit-serial multiplier for binary fields that also features a flexibility and low hardware complexity. The multiplier presented here aims also to wireless applications and is using the method of Montgomery in combination with the classical method.

For a detailed survey on finite-field multipliers for Public Key Cryptography see \cite{3}.

However, very few efficient hardware implementations present a completely generic solution which allows an arbitrary choice for all parameters: field, basis representation, irreducible polynomial, bit-lengths etc. The flexibility of our architecture forms an important advantage for cryptographic applications. More precisely, the architecture introduced here is scalable to every desired bit-length and it allows any choice of binary field, coordinates or irreducible polynomial. These properties make the proposed multiplier as a suitable choice for all applications of ECC.

II. BACKGROUND INFORMATION

Public key cryptosystem allow secure communications over insecure channels without prior agreement of a shared secret; they also enable digital signatures. ECC were proposed in the mid 1980’s \cite{10, 15}; they are becoming increasingly popular in the last years, which can be illustrated by their presence in cryptographic standards. Common fields to implement ECC are \( GF(p) \) (for a prime \( p \) with at least 160 bits) and \( GF(2^n) \) \((n \geq 160)\). ECC rely on a group structure induced on the elliptic curve. A set of points on an elliptic curve (with one special point added, the so-called point at infinity) together with a point addition as a binary operation has the structure of an Abelian group. It is then natural to define a point or scalar multiplication as a multiplication of an arbitrary point on an elliptic curve with some integer; this operation is the basic operation for cryptographic protocols.
Efficient implementations of a point multiplication rely on a modular multiplication in the underlying finite field. More details on ECC can be found in [5], [11], [13].

This paper deals with polynomial bases, where the elements of $GF(2^n)$ are polynomials of degree at most $n - 1$ over $GF(2)$, and arithmetic is carried out modulo an irreducible polynomial $f(x)$ of degree $n$ over $GF(2)$. In this case the basis elements have the form $1, \omega, \omega^2, \ldots, \omega^{n-1}$ where $\omega$ is a root in $GF(2^n)$ of the irreducible polynomial $f(x)$ of degree $n$ over $GF(2)$. According to this representation an element of $GF(2^n)$ is a polynomial of length $n$ and can be written as: $a(x) = \sum_{i=0}^{n-1} a_i x^i = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0$, where $a_i \in GF(2)$.

A point addition (or doubling) requires one field inversion ($I$), two field multiplications ($M$) and one squaring ($S$), or $1I+2M+1S$. Here, we consider squaring as a special case of multiplication; by using projective coordinates $a$, $b$ and $f$ are polynomials of length $n$, $a_i \in GF(2)$, $b_i \in GF(2)$ and $f_i \in GF(2)$ respectively. The outputs $c(x)$ become inputs to the systolic arrays in the next clock cycle. Finally, the result of the multiplication is obtained by XOR-ing the outputs of both systolic arrays. We show details of the bit-serial version; the extension to digit-serial version is straightforward. The Montgomery part is presented in [18] as a complete multiplier performing only MMM in $GF(2^n)$.

Another possibility to calculate the product of two polynomials in $GF(2^n)$ is Montgomery’s multiplication algorithm as proposed in [6]. Here we define the MMM as: $\text{MMM}[(a(x), b(x))] := a(x) \cdot b(x) \cdot [r(x)]^{-1} \mod f(x)$. Before a sequence of operations can be started, all operands have to be converted to the form $a(x)r(x) \mod f(x)$, the so-called $M$-residue of the operand. This is done by performing a Montgomery multiplication of the element and $[r(x)]^{-1}$. The result of a Montgomery multiplication on $M$-residues will once again be an $M$-residue, which is converted back to the normal domain with MMM(Res, 1).

The classical and the Montgomery modular multiplication algorithms can be combined into one algorithm. The main idea is to perform both algorithms in parallel, as proposed in [21]. The author prefers this solution to either of the methods alone, but the prototype architecture in [21] is not very efficient as it is not a systolic array. The detailed algorithms are described in [2].

III. ON SERIAL FPGA IMPLEMENTATIONS

A. Bit-serial version of the multiplier

Our circuit implements Algorithm 1; it includes two parts, classical and Montgomery, each of which is a systolic array. The parts look quite similar as their cells are performing similar operations i.e. multiplication and XOR. The difference is that they shift in opposite directions and they start from the opposite parts of the loop. While the classical multiplier starts the shift-and-process from the MSB of one of the operands, and shifts the cumulative result left, the Montgomery based multiplier starts at the LSB, and shifts the result right. They process the operand $a(x)$ from different sides and they stop after exactly $n/2$ cycles. The classical part has still to perform a shift over $n/2$ bits but this is taken care of by the conversion of the $M$-residue of the result. More precisely, the $M$-residue is of the form $a(x)b(x)r^{-1}(x) \mod f(x)$ where $r(x) = x^{n/2}$ (for $n$ even). The multiplication is completed by calculating $\text{MMM}[\text{Res}(x), 1] = a(x)b(x)x^{-n/2} \cdot 1 \cdot x^{n/2} \mod f(x) = a(x)b(x) \mod f(x)$.

A schematic of the multiplier is presented in Figure 1. $a_i$, $b_i$ and $f_i$ are the coefficients of $a(x)$, $b(x)$ and $f(x)$ respectively. The equations of the multiplier are described in Algorithm 1 and 2. The result of the multiplier is obtained by XOR-ing the outputs of both systolic arrays. The result is of the form $a(x)b(x) \mod f(x)$.

Algorithm 1 Combined Modular Mult. in $GF(2^n)$

\begin{itemize}
  \item[Input:] polynomials $a(x)$, $b(x)$ and $f(x)$,
  \item[Output:] $\text{Res}(x) = a(x) \cdot b(x) \cdot x^{-\frac{\nu}{2}} \mod f(x)$
\end{itemize}

1: $\text{Res}(x) = 0$, $\text{Res}_C(x) = 0$, $\text{Res}_M(x) = 0$
2: for $i$ from $0$ to $\frac{n}{2} - 1$ do
3: \quad $\text{Res}_C(x) \leftarrow \text{Res}_C_{n-i-1} \cdot f(x) + \text{Res}_C(x) \cdot x + a_{n-i-1} \cdot b(x)$
4: \quad $\text{Res}_M(x) \leftarrow \text{Res}_C(x) + \text{Res}_M(x)$
5: \quad $\text{Res}_M(x) \leftarrow \text{Res}_M(x) + \text{Res}_M \cdot f(x)$
6: \quad $\text{Res}_M(x) \leftarrow \text{Res}_M(x) \cdot \text{div} x$
7: \quad $\text{Res}(x) = \text{Res}_C(x) + \text{Res}_M(x)$
8: end for
9: Return $\text{Res}(x)$

Fig. 1. Schematic of combined multiplier. Classical and Montgomery’s part are calculating in parallel and the result is XOR-ed afterwards.
B. Results

The implementation results of the combined multiplier on a Xilinx Virtex XCV800 FPGA are given in Table 1. These results are obtained with Xilinx Foundation software. As mentioned above, the extension to a digit-serial multiplier is straightforward. The digit-serial version of Montgomery’s algorithm is also presented in [6].

<table>
<thead>
<tr>
<th></th>
<th>MMM mult. [18]</th>
<th>comb. mult.</th>
</tr>
</thead>
<tbody>
<tr>
<td># clock cycles</td>
<td>160</td>
<td>80</td>
</tr>
<tr>
<td>Min. clock period (ns)</td>
<td>10.775</td>
<td>13.860</td>
</tr>
<tr>
<td>Total MMM latency (µs)</td>
<td>1.66</td>
<td>1.109</td>
</tr>
<tr>
<td># of Slices</td>
<td>1427</td>
<td>1089</td>
</tr>
</tbody>
</table>

IV. Side-channel Security

When proposing an efficient implementation of a cryptographic algorithm one should also consider side-channel security. Namely, an implementation of a cryptographic algorithm can be subjected to side-channel attacks such as power analysis attacks [12], [14]. These attacks present a realistic threat for wireless applications and have been demonstrated to be very effective against smart cards without specific countermeasures. A power analysis attack exploits the fact that the power consumption during a cryptographic operation is related to the function being performed and to the (possibly sensitive) data being processed. The advantage of our multiplier is that it processes 2 bits (or 2 words) in parallel. This feature does not only improve performance but helps in protecting against side-channel attacks because of the parallel processing. To support this claim we present two power consumption graphs for 480-bit multiplications, which show the power consumption for Montgomery’s algorithm (Figure 2) and for the combined algorithm (Figure 3). These graphs are obtained using the measurement setup discussed in [19].

The power consumption graph for the combined multiplier contains traces of two overlapping multiplications and appears to include more noise. One can anticipate that it will be harder to handle this with power analysis attacks. The general idea is that when more calculations are performed at the same time, the attacker has a more difficult task in front of him. Strong resistance against side-channel analysis attacks will of course require additional countermeasures.

V. Conclusions

This paper presents an FPGA implementation of a new multiplier for the finite field $GF(2^n)$ using a polynomial basis representation. Performance data are given, showing a reduction in the number of cycles with a factor 2, without increasing the gate complexity. We can also show some improved resistance against power analysis.

Fig. 2. Power consumption curve for multiplication via Montgomery’s algorithm.

Fig. 3. Power consumption curve for multiplication via combined algorithm. The peak at the end corresponds to the XOR of the two parts.

ACKNOWLEDGMENT

Lejla Batina, Nele Mentens and Siddika Berna Örs are funded by research grants of the Katholieke Universiteit Leuven, Belgium. This work was supported by Concerted Research Action GOA-MEFISTO-666 of the Flemish Government and by the FWO “Identification and Cryptography” project (G.0141.03).

REFERENCES