International Journal of Research
Available at https://edupediapublications.org/journals

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

Factoring Technique Based Low Power Finite Field Multiplier Digit Serial Polynomial

Ms. Ms. Sirangi Sai Chathurya & Mr. S.Ranjith Kumar
1. Department of ECE, Ganapathy Engineering College , Warangal, India.
2. Associate Professor, Department of ECE , Ganapathy Engineering College , Warangal, India

ABSTRACT

This paper presents a bit-serial architecture for efficient addition and multiplication in binary finite fields GF (2^m) using a polynomial basis representation. Moreover, a low-voltage/low power implementation of the arithmetic circuits and the registers is pro-posed. The introduced multiplier operates over a variety of binary fields up to an order of 2^m. We detail that the bit-serial multiplier architecture can be implemented with only 28m gate equivalents, and that it is scalable, highly regular, and simple to design.

Index terms: Finite field arithmetic, bit-serial multiplier architec-ture, smart card crypto-coprocessor, low-power VLSI design.

1. INTRODUCTION

Finite fields are increasingly important for many applications in cryptography and algebraic coding theory [4]. Certain properties of the binary finite field GF(2^m) like its “carry-free” arithmetic make it very attractive for hardware implementation. Another advantage of GF(2^m) is the availability of different equivalent rep-resentations of the field elements, e.g. polynomial bases, normal bases, or dual bases. Depending on the applications, the degree *m* of the field can vary over a wide range. In *elliptic curve cryptography* [3], *m* has to be prime and is usually between 160 and 200, but it can also be as large as 500 or even more. The performance of an elliptic curve (EC) cryptosystem is primarily determined by the efficient implementation of the field arithmetic. In environments with limited computational power, such as smart cards, a hardware acceleration of the field arithmetic is necessary to reach high performance, especially if *m* is beyond 200.

According to the different basis representations, a variety of algorithms and architectures for multiplication in GF(2^m) have been proposed. As an example, we refer to [1] for a report on a Normal basis multiplier. However, in order to be compliant to well accepted standards for EC cryptography [5], the polynomial basis representation seems to be the best choice. From an architectural point of view, a polynomial basis multiplier can be realized in a bit-serial, digit-serial, or bit-parallel fashion. For area-restricted devices like smart cards, the bit-serial architecture offers a fair area/performance trade-off. Bit-serial architectures for polynomial basis multiplication are well known since the early 1970s due to their exploration in coding theory [6], and later they have also been proposed for use in cryptography [2].

This paper presents a bit-serial multiplier architecture for the finite field GF(2^m) which is optimized for elliptic curve cryptogra-phy. Contrary to other designs, the proposed architecture can also perform addition of field elements, and it operates over a wide range of finite fields since it is neither restricted to a field of a given order, nor does it favor special irreducible polynomials like trinomials or pentanomials. The field multiplication is performed according to the shift-and-add principle and requires *m* clock cycles. The reduction modulo the irreducible polynomial (IP) is realized by interleaved subtractions of the IP. Major advantages of the bit-serial architecture are low power consumption and a high degree of regularity, which makes it very attractive for VLSI implementation.

The remainder of this paper is structured as

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

follows: Section 2 provides some mathematical background information on the finite field GF(2^m) with an emphasis on the basic arithmetic operations (addition, multiplication). In Section 3, the bit-serial multiplier architecture is presented. Additionally, the execution of a field-multiplication is explained in Section 3.1. VLSI design related issues with a special focus on low-area and low-power implementation of the field arithmetic are sketched in Section 4. The paper finishes with a summary of results (i.e. estimation of the silicon area) and conclusions in Section 5.

2. ARITHMETIC IN THE FINITE FIELD GF(2^m)

Abstractly, a *finite field* (or *Galois field*) consists of a finite set of elements together with the description of two operations (addition and multiplication) that can be performed on pairs of field elements. These operations must possess certain properties associativity and commutativity of both addition and multiplication, the distributive law, existence of an additive identity and a multiplicative identity, and existence of additive inverses as well as multiplicative inverses. GF(2) is the smallest possible finite field; it just contains the integers 0 and 1 as field elements. Addition and multiplication are performed modulo 2, therefore the addition is equivalent to the logical XOR, and the multiplication corresponds to the logical AND.

The binary finite field GF(2^m) contains 2^m elements where m is a non-zero positive integer. Each of the 2^m elements of GF(2^m) can be uniquely represented with a polynomial of degree up to $m-1$ with coefficients in GF(2). For example, if $a(t)$ is an element in GF(2^m), then one can have $m-1$

$a(t) = {}^{X}a_i\, t^i = a_{m-1}t^{m-1} + \cdots + a_2t^2 + a_1t + a_0$ (1)

i=0 with $a_i \in \{0, 1\}$. This binary polynomial can also be written in bit-string form as $A[m-1 \,..\, 0]$, whereby $A[i]$ corresponds to the coefficient a_i. Addition in GF(2^m) is implemented as component-wise XOR, while a multiplication can be performed modulo an *irreducible polynomial* $p(t)$ of degree m with coefficients p_i in GF(2). A polynomial of degree m is said to be irreducible over GF(2) if it can not be factored into a product of polynomials each of whose degree is less than m with coefficients in GF(2).

2.1. Addition

The addition of two field elements of GF(2^m) is performed by adding the coefficients modulo 2, which is nothing else than bit-wise XOR-ing the coefficients of equal powers of t. That is, if $a(t)$, $b(t) \in$ GF(2^m), the addition is done as follows:

$(m-1)a(t) + b(t) = c(t) = c_i\, t^i$ with $c_i = a_i + b_i$ mod 2 (2) i=0

Compared to the addition of integers, the addition in GF(2^m) is much easier as it does not cause a carry propagation from less to more significant bit positions. A hardware implementation with an ensemble of m XOR gates can perform the addition in constant time. In the finite field GF(2^m), each element $a(t)$ is its own additive inverse since $a(t) + a(t) = 0$, the additive identity. Thus, addition and subtraction are equivalent operations in GF(2^m).

2.2. Multiplication

Contrary to GF(2), a number of different representations is com-monly used for the elements of a finite field GF(2^m). The simplest representation is in *polynomial basis*, where the multiplication involves multiplying the two polynomials (carry-free coefficient multiplication) and then finding the residue modulo a given ir-reducible polynomial $p(t)$. In general, the reduction modulo an irreducible polynomial requires polynomial division. However, the polynomial division is very costly to implement in hardware. For an efficient implementation, it is necessary to find a method to perform the field multiplication without division.

One possibility is to interleave the reduction modulo $p(t)$ with the multiplication operation, instead of performing the reduction separately after the multiplication of the polynomials is finished. This leads to a method which is generally known as *"shift-and-add"* multiplication, where the product is obtained by the addition of partial-products, and the reduction is interleaved with the addition steps and performed by subtractions of the irreducible polynomial. The following pseudo-code describes this algorithm for multiplying two polynomials $a(t)$, $b(t) \in GF(2^m)$ modulo an irreducible polynomial $p(t)$ of degree m.

```
1.  $r(t) \leftarrow 0$
2.  **for** $i = m - 1$ **downto** 0 **do**
2a. $r(t) \leftarrow t \cdot r(t) + a_i \cdot b(t)$
2b. **if** degree($r(t)$) = $m$ **then** $r(t) \leftarrow r(t) - p(t)$
3.  **return** $r(t)$
```

The multiplication of two polynomials $a(t)$, $b(t)$ is done by scan-ning the coefficients of the multiplier-polynomial $a(t)$ from a_{m-1} to a_0 and adding the partial-product $a_i \cdot b(t)$ to the intermediate result $r(t)$. The partial-product $a_i \cdot b(t)$ is either 0 (if $a_i = 0$) or the multiplicand-polynomial $b(t)$ (if $a_i = 1$). After each partial-product addition, the intermediate result must be multiplied by t to align it for the next partial-product. The reduction modulo the irreducible polynomial $p(t)$ is interleaved with the addition steps by subtraction of $p(t)$ whenever the degree of the intermediate result-polynomial is m.

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 17
July 2018

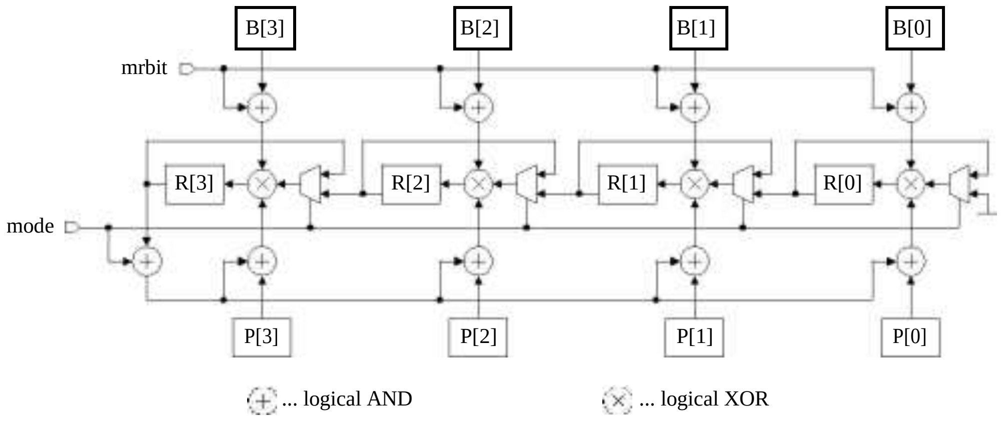

Figure 1: MSB-first bit-serial multiplier architecture for GF(2^4)

3. MULTIPLIER ARCHITECTURE

A bit-serial multiplier architecture for m-bit operands has an area complexity of $O(m)$, and also the computation time is propor-tional to m, i.e. a multiplication in GF(2^m) requires m clock cycles. In general, there exist two versions of bit-serial multiplier architectures, depending on the schedule of the operands: The LSB-first version, and the MSB-first version. Figure 1 shows a MSB-first version of a bit-serial architecture for GF(2^4) using polynomial basis representation. According to the bit-string notation introduced in Subsection 2.1, the coefficients b_i, r_i, p_i of the multiplicand polynomial $b(t)$, the result polynomial $r(t)$, and the irreducible polynomial $p(t)$ are symbolized as B[i], R[i], and P[i], respectively.

The multiplier architecture depicted in Figure 1 employs multiplexers to switch between two modes of operation: Addition mode and multiplication mode. In addition mode (control signal *mode* = 0), the upper inputs of the multiplexers in Figure 1 are propagated to the outputs. In this mode, the reduction (i.e. subtraction of $p(t)$) is disabled and the multiplier performs the addition $r(t) \leftarrow r(t) + b(t)$ if the multiplier bit *mr bit* is set to 1.

Alternatively, in multiplication mode, the architecture illustrated in Figure 1 is in principle a direct mapping of the “shift-and-add” algorithm described in Subsection 2.2. The generation of the partial-product $a_i \cdot b(t)$ is simply done by a bit-wise AND operation of the coefficient a_i (the multiplier bit, denoted as *mr bit* in Figure 1)

and all the coefficients of $b(t)$. The multiplication $t \cdot r(t)$ from line (2a.) of the algorithm is nothing else than a 1-bit left-shift of the bit-string representation of $r(t)$. The only point in which a multiplication in the presented multiplier architecture differs from the algorithm in Section 2.2 is the order of partial-product addition and reduction modulo $p(t)$, i.e. subtraction of the irreducible poly-nomial. In the architecture depicted in Figure 1, these operations are performed together within one clock cycle, which makes it necessary to subtract the irreducible polynomial $p(t)$ when the degree of $r(t)$ is $m - 1$. However, the value to be subtracted from $r(t)$ is either 0 (if the coefficient $r_{m-1} = 0$) or $p(t)$ (if $r_{m-1} = 1$). Both cases require a bit-wise AND operation between r_{m-1} and all coefficients p_i of the irreducible polynomial $p(t)$. It turns out that the computation of $r(t) = a(t) \cdot b(t) \bmod p(t)$ requires m steps; at each step we perform the following operations:

– Calculation of $t \cdot r(t)$ (a 1-bit left-shift).

– Generation of the partial-product (logical AND between a_i and the coefficients of $b(t)$).
– Addition of the partial-product (m-bit XOR operation).
– Generation of the subtrahend (logical AND between r_{m-1} and the coefficients of $p(t)$).
– Subtraction of subtrahend (m-bit XOR operation).

Thus, the required logical operations are reduced to AND, XOR and 1-bit left-shift operations, which makes an implementation of the bit-serial architecture very simple.

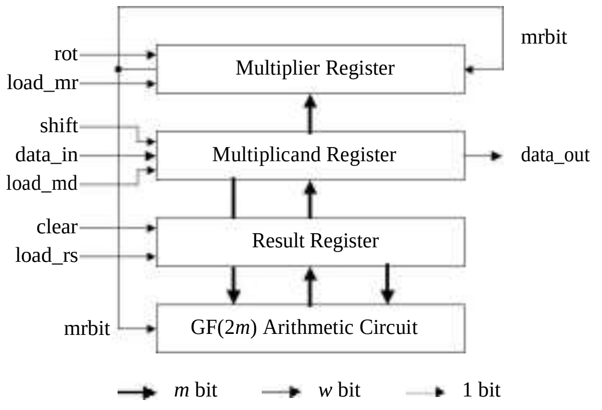

Figure 2: Block diagram of the proposed GF(2^m) multiplier

In the following we propose an implementation of the bit-serial architecture which can be used as a coprocessor in a smart card. The main design goal is to achieve the smallest possible silicon area. Some “tricks” employed to reach this goal will be presented later in this section. Figure 2 shows the major components of our bit-serial GF(2^m)-multiplier. The introduced design can be imple-mented to provide a classical coprocessor functionality: first the operands are loaded into registers within the multiplier, then the arithmetic operation (addition or multiplication) is executed, and finally the result can be fetched from register *Result*. Note that in Figure 2, the register for the irreducible polynomial is not shown.

The register *Multiplier* is used to store the bit-string represen-tation of the multiplier-polynomial $a(t)$. This register must be able to perform 1-bit left-shift operations in MSB direction and an m-bit parallel load operation. The multiplicand-polynomial $b(t)$ is stored in the register *Multiplicand*. The register *Multipli-cand* is supposed to be able to carry out an m-bit parallel load operation and additionally a w-bit right-shift operation, whereby w denotes the wordsize of the data bus (typically 8 or 16 bit). All data transfers from the multiplier core to registers outside the crypto-coprocessor and vice versa are performed via the register *Multiplicand*. Therefore, register *Result* supports parallel data transfer to register *Multiplicand*, and register *Multiplier* is able to perform parallel data transfer from register *Multiplicand*.

3.1. Execution of a Multiplication

In order to explain how a multiplication is carried out in the presented bit-serial multiplier, let us assume that at the beginning of the multiplication the multiplier-polynomial $a(t)$ resides in register *Multiplier* and the multiplicand-polynomial $b(t)$ is stored in register *Multiplicand*. After the register *Result* has been cleared, the register *Multiplier* is shifted bit by bit in MSB direction, which delivers the multiplier-bits (i.e. the coefficients a_i) to the GF(2^m) arithmetic unit, beginning with a_{m-1}. The calculation of the intermediate result $r(t)$ is performed as described earlier in this section. After the final coefficient a_0 has been processed, the result of the multiplication resides within register *Result*. From there it can be loaded into register *Multiplicand*, where it might act as multiplicand-polynomial for the next multiplication, or from where it can be transferred to the world outside the multiplier.

4. IMPLEMENTATION ISSUES

In this section we present a possible implementation of the regis-ters and the arithmetic circuits. Due to the high regularity of the multiplier architecture, the register cells and the arithmetic cells essentially determine the overall size of the crypto-coprocessor. It turns out that transmissions gates are very useful for an area and power optimized implementation of the proposed field multiplier.

4.1. Implementation of the Field Arithmetic

Transmission gates are often used in digital design as they allow simple realizations of complex circuits. But when designing trans-mission gate based devices, one has to be aware of some problems which are specific for that circuit class. A transmission gate is not an ideal switch, has limited drive strength, and causes delay.

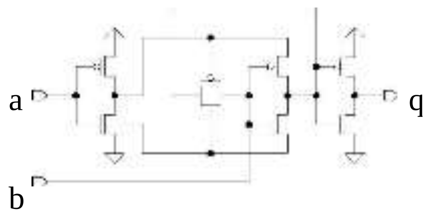

Figure 3: Transmission gate XOR

An example of the effective use of transmission gates is the popular XOR circuit shown in Figure 3 (see [7]). This XOR gate requires only eight transistors (including the inverter at the output). compared to the twelve transistors required for a complementary implementation. Moreover, the presented XOR circuit does not require complementary input signals and is very well suited for low-voltage low-power applications. The XNOR function can be realized in a similar way as the presented XOR circuit.

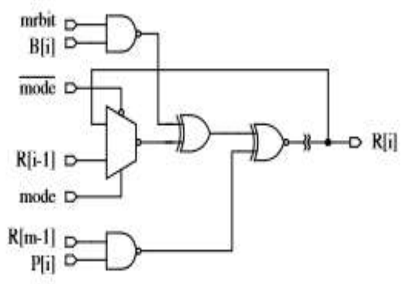

Figure 4: 1-bit arithmetic cell

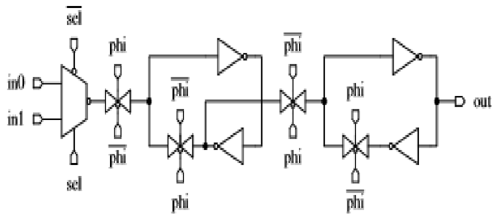

Figure 5: Static register cell with two data inputs.

Figure 4 shows a 1-bit arithmetic cell which can be used in the GF(2m)-multiplier. The arithmetic cell performs generation and addition of the partial-product, as well as subtraction of the irreducible polynomial if the degree of the result-polynomial is m−1. Each arithmetic cell contains a multiplexer to switch between addition mode and multiplication mode. A multiplexer can be implemented with two transmission gates and an inverter (see [7], p. 212). In this case, the presented arithmetic cell consists of 30 transistors.

4.2. Static Register Cell Static CMOS logic is known to be fast, offers a large noise margin, and is very easy to design. But also from the viewpoint of power consumption, static register cells provide a significant advantage compared to their dynamic counterparts: They do not require a clock signal to store their logical state. The principle of a gated clock is often used in synchronous digital systems to prevent inactive circuits from consuming unnecessary power. The basic idea of clock gating is to mask off the clock to registers which are idle. Synchronous load-enable registers can be realized in this way by inserting AND gates into the clock network. The clock pulses are prevented from propagating to the registers whenever the load enable signal is false. In summary, gating the clock of a register saves power in two ways: First, a register consumes significantly less power when it is not clocked since clock toggling is generally considered to cause substantial power dissipation in flip-flops. Second, also the corresponding portion of the clock tree does not consume power when the clock signal is masked off. Especially for the register Multiplicand, a gated

clock can be advantageous since this register is only active during operand loading and is always idle for the remaining time of the multiplication. A static register cell which provides serial and parallel load can be composed of four transmission gates, four inverters and a multiplexer,asshowninFigure5. Whenthemultiplexerisrealized by two transmission gates and an inverter, the static register cell consists of 22 transistors.

5. SUMMARY OF RESULTS AND CONCLUSIONS

A 1-bit multiplier cell of the proposed bit-serial architecture consists of 3 register cells and the 1-bit arithmetic circuit illustrated in Figure 4. Additionally, a simple register cell for the irreducible polynomial is required, which can be implemented with 16 transistors. Thus, the overall transistor count of the 1-bit multiplier cell is 112, which results in 28m gates for an m-bit multiplier. A multiplier based on the presented architecture has a linear array structure with a bit-slice feature. The multiplication requires m cycles, whereby the max. clock frequency is independent of m, but primarily determined by the delay of the arithmetic circuit. Low power consumption can be achieved by clock gating, and contrary to LSB-first architectures only two registers (Result and Multiplier) have to be clocked at any given cycle. An m-bit multiplier operates over a variety of fields. For example, a multiplier originally dimensioned for 200 bits can also be used for fields of smaller order (e.g. 163 bits) by left-aligning all operands in the registers. Therefore, the presented design is very well suited to implement a small-area and low-power GF(2m)-multiplier for elliptic curve cryptography.

REFERENCES

[1] G.B.Agnew,R.C.Mullin,andS.A.Vanstone. Animplementationofellipticcurvecryptosystemso verF2155. IEEEJournal on Sel. Areas in Communications, 11(5):804–813, June 1993. [2] T. Beth, B. M. Cook, and D. Gollmann. Architectures for exponentiation in GF(2n). Advances in Cryptology — CRYPTO ’86, LNCS 263 pp. 302–310. Springer Verlag, 1987. [3] I. F. Blake, G. Seroussi, and N. P. Smart. Elliptic Curves in Cryptography. Cambridge University Press, 1999. [4] R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge University Press, 1994. [5] National Institute of Standards and Technology (NIST). Recommended elliptic curves for federal government use, 1999. [6] W. W. Peterson and E. J. Weldon. Error-Correcting Codes. MIT Press, second edition, 1972. [7] J. M. Rabaey. Digital Integrated Circuits – A Design Perspective. Prentice Hall, 1996.