VLSI Performance Evaluation and Analysis of Systolic and Semisystolic Finite Field Multipliers

Ravi Kumar Satzoda and Chip-Hong Chang

Centre for High Performance Embedded Systems,
Nanyang Technological University, Singapore, 639798
{rksatzoda, echchang}@ntu.edu.sg

Abstract. Finite field multiplication in $GF(2^m)$ is an ineluctable operation in elliptic curve cryptography. The objective of this paper is to survey fast and efficient hardware implementations of systolic and semisystolic finite field multipliers in $GF(2^m)$ with two algorithmic schemes – LSB-first and MSB-first. These algorithms have been mapped to seven variants of recently proposed array-type finite-field multiplier implementations with different input-output configurations. The relative VLSI performance merits of these ASIC prototypes with respect to their field orders are evaluated and compared under uniform constraints and in properly defined simulation runs on a Synopsys environment using the TSMC 0.18μm CMOS standard cell library. The results of the simulation provide an insight into the behavior of various configurations of array-type finite-field multiplier so that system architect can use them to determine the most appropriate finite field multiplier topology for required design features.

1 Introduction

In recent years, there has been a resurgence of interest in the pursuit of VLSI efficient finite field arithmetic due mainly to its application in public key cryptography. One strong merit of public key cryptography in information security lies in its ease of key management [1]. RSA, Deffie-Hellman and Elliptic Curve Cryptography (ECC) are few proven and prevailingly used public key encryption algorithms reported in the literature [1, 2]. ECC has been an active field of research because ECC provides higher level of security at smaller key lengths than RSA [4]. This helps to reduce the cost of implementation both in software and hardware. However, software implementations of ECC are comparatively slower. Therefore, it is desired to design efficient architectures in hardware for cryptographic algorithms.

All the operations in ECC are done in finite fields $GF(p^m)$. When $p = 2$, elements in Galois field $GF(2^m)$ are represented as binary numbers and all the operations can be performed using standard logic gates in binary domain. Addition in $GF(2^m)$ is simpler than normal binary addition as the sum can be obtained by bit-wise XOR operation of the two operands and the latency is independent of the operand sizes since there is no carry chain [16]. However, multiplication is more complicated because it involves calculating the modulus with

© Springer-Verlag Berlin Heidelberg 2005

respect to an irreducible polynomial $f(x)$, i.e. $a(x) \times b(x) \bmod f(x)$ if the field elements are expressed in polynomial bases. Important techniques for software and hardware implementations of finite field multiplication have been reported in literature [3,4,5,6,7,8,9,10,11,12,13,14,15], [18], [20]. Several software implementations of finite field multiplication are listed in [3] whereas efficient hardware architectures are described separately in [4,5,6,7,8,9,10,11,12,13,14,15], [18], [20]. There are generally three different categories into which reported multipliers can be sieved. [4] illustrates a multiplier of the first category where reduction is given higher priority than multiplication. Multiplication is performed by simple *AND-XOR* network. However, reduction is more complicated and efficient implementations for reduction are designed. In [5,6,7,8,9,20], a variety of implementations of Mastrovito, Karatsuba and Massey-Omura multipliers have been reported, which form the second category. The third category comprises systolic and semisystolic multipliers which perform multiplication and reduction simultaneously.

Systolic and semisystolic finite field multipliers are reported separately in [10, 11, 12, 13, 14, 15]. In this paper, we survey some recently reported systolic and semisystolic finite field multipliers. We describe these architectures briefly followed by important results reported in [11, 12, 13, 14]. Different input-output configurations like bit-serial, bit-parallel and digit-serial are discussed. In contrast to performance results reported for these multipliers, which are determined by analytical models of area-time complexity, the VLSI performance metrics of the reported designs are realistically evaluated for different field sizes by mapping the various architectures to ASICs using the 0.18μm CMOS technology libraries. Based on the results of our simulation, a system architect can decide on a suitable systolic or semisystolic array-type finite field multiplier for designing an optimized ECC cryptosystem with the desired characteristics of chip area, throughput rate and power consumption.

This paper is structured as follows. Section 2 summarizes multiplication in $GF(2^m)$ followed by the description of various systolic and semisystolic architectures we intend to synthesize and compare in Section 3. In Section 4, simulation and synthesis results of ASIC implementations are analyzed and discussed in detail. We conclude and discuss future research in Section 5.

2 Multiplication in $GF(2^m)$

In this section we describe multiplication in Galois field $GF(2^m)$. More details on finite field arithmetic can be found in [16], [19]. Finite field $GF(2^m)$ contains 2^m elements which is an extension of $GF(2)$ where the elements $\in \{0, 1\}$. Elements in $GF(2^m)$ can be expressed in two different bases – polynomial and normal bases [16]. All the architectures described in this paper are based on polynomial basis representation. Each element in $GF(2^m)$ is represented as a binary polynomial of degree less than m. If $a \in GF(2^m)$, then a is represented as

$$a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 \tag{1}$$

where $a_i \in \{0,1\}, 0 \leq i \leq m-1$. Thus a field element, $a(x)$ can be denoted by binary string $(a_{m-1}a_{m-2}\ldots a_1a_0)$.

Let $a(x) = (a_{m-1}a_{m-2}\ldots a_1a_0)$ and $b(x) = (b_{m-1}b_{m-2}\ldots b_1b_0)$ be two elements in $GF(2^m)$. Multiplication of $a(x)$ and $b(x)$ in $GF(2^m)$ involves an irreducible polynomial $f(x)$ given by

$$f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \cdots + f_1x + f_0 = x^m + \sum_{i=0}^{m-1} f_ix^i \quad (2)$$

where $f_i \in \{0,1\}$.$f(x)$ is called reduction polynomial. The product of $a(x)$ and $b(x)$ is defined as

$$c(x) = \sum_{i=0}^{m-1} c_ix^i = a(x) \times b(x) \; mod \; f(x) = (\sum_{i=0}^{m-1} a_ix^i)(\sum_{i=0}^{m-1} b_ix^i) \; mod \; f(x) \; . \quad (3)$$

In (3), $a(x) \times b(x)$ gives $c^{'}(x) = \sum_{i=0}^{2m-2} c_i^{'}x^i$, a polynomial of degree $2m-2$ or $2m-1$ bit binary string. $c^{'}(x)$ is reduced by $f(x)$ to give $c(x)$ which is a polynomial of degree $m-1$. $c(x)$ is the remainder obtained on dividing $c^{'}(x)$ by $f(x)$. Since there is no carry chain for addition in $GF(2^m)$, multiplying two m-bit vectors produces a $2m-1$ bit product, as opposed to the $2m$ bit product in normal binary multiplication. Multiplication is implemented in hardware/software in two different ways. The first method determines $c^{'}(x) = a(x) \times b(x)$ and then uses the reduction polynomial $f(x)$ to reduce $c^{'}(x)$ to get $c(x) = c^{'}(x) \; mod \; f(x)$. In the second method, the partial-product bits obtained by multiplying $a(x)$ by b_i are reduced immediately by $f(x)$.

The systolic and semisystolic multipliers are illustrated based on the second method. All these multipliers are implemented based on the following two schemes.

LSB-first method:
$c(x) = a(x)b(x) \; mod \; f(x)$
$= b_0a(x) + b_1[a(x)xmodf(x)] + b_2[a(x)x^2modf(x)] + \cdots + b_{m-1}[a(x)x^{m-1}modf(x)]$
MSB-first method:
$c(x) = a(x)b(x) \; mod \; f(x)$
$= \{\cdots[a(x)b_{m-1}x \; mod \; f(x) + a(x)b_{m-2}]x \; mod \; f(x) + \cdots + a(x)b_1\}x \; mod \; f(x) + a(x)b_0$

The two methods differ in the order the bits of the multiplier $b(x)$ are accessed. In LSB-first method, the multiplication starts with the LSB of $b(x)$ whereas the latter starts with MSB of $b(x)$. The pseudo-codes for their implementations are shown in Algorithm 1 [15] and Algorithm 2 [14].

```
Algorithm 1: LSB-first
Input: a(x),b(x),f(x)
Output: c(x) = a(x)b(x) mod f(x)
$t_j^{(0)} = 0,\ for\ 0 \leq j \leq m-1$
$a_{-1}^{(i)} = 0,\ for\ 0 \leq i \leq m$
$a_j^{(0)} = 0,\ for\ 0 \leq j \leq m-1$
$for\, i\ =\ 1\, to\, m$
$\{for\, j\ =\ m-1\, to\, 0$
$\{a_j^{(i)} = a_{j-1}^{(i-1)} + a_{m-1}^{(i-1)} f_j$
$t_j^{(i)} = a_j^{(i-1)} b_{i-1} + t_j^{(i-1)}\}\}$
$c(x) = t^{(m)}(x)$
```

```
Algorithm 2: MSB-first
Input: a(x),b(x),f(x)
Output: c(x) = a(x)b(x) mod f(x)
$t_j^{(0)} = 0,\ for\ 0 \leq j \leq m-1$
$t_0^{(i)} = 0,\ for\ 1 \leq i \leq m$
$for\, i\ =\ 1\, to\, m$
$\{for\, j\ =\ m-1\, to\, 0$
$\{t_j^{(i)} = t_{m-1}^{(j-1)} f_j + b_{m-i} a_j + t_{j-1}^{(i-1)}\}\}$
$c(x) = t^{(m)}(x)$
```

Both algorithms stated above have been realized with bit-serial and bit-parallel architectures in [10, 11, 12, 13, 14, 15] [18] [20]. Bit-parallel multipliers provide higher throughput at a cost of increased hardware. Hardware complexity is an important criterion for public key cryptography on smart card in view of the finite field operations with large word length. Therefore, bit-serial multipliers are preferred over bit-parallel counterparts in those applications. However, latency becomes an issue in bit-serial multipliers. Digit-serial multipliers have also been reported to reduce latency but digitizing the design increases hardware complexity and combinational delay as compared to bit-serial multipliers. In this paper, we survey important systolic and semisystolic architectures that have been reported in the last ten years. A comprehensive analysis of the ASIC prototypes of the seven different architectures is provided.

3 Architectures of Finite Field Multipliers

In this section, we describe some systolic and semisystolic multiplier architectures. A quantitative analysis in terms of gate count and latency is presented here. Algorithm to architecture mapping is described briefly for each of the multipliers to be evaluated.

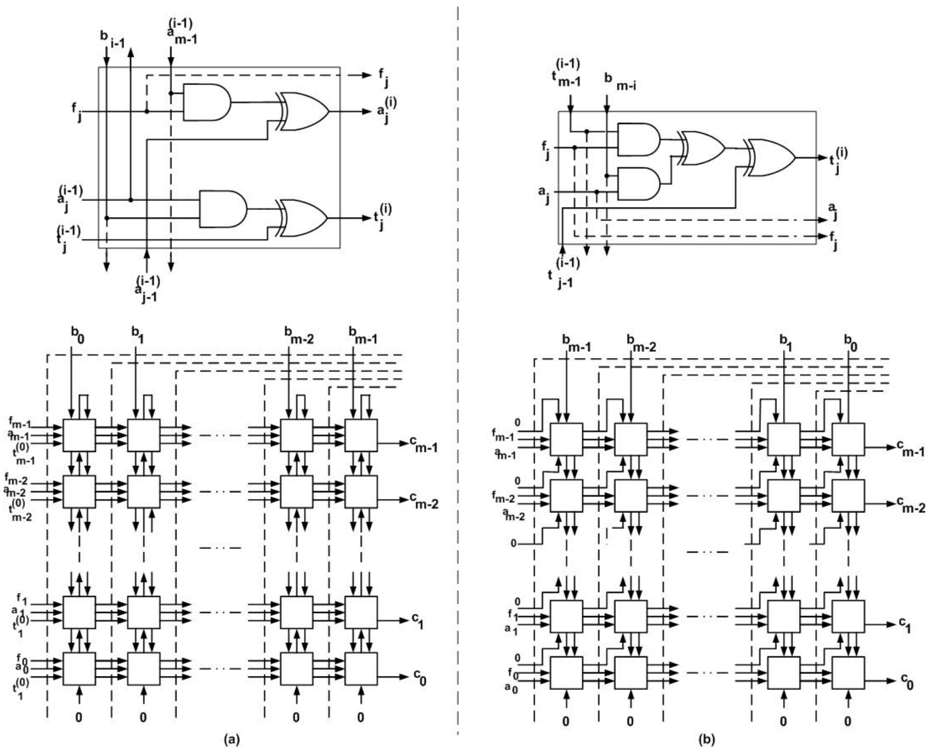

Fig. 1. (a) Basic cell and architecture for LSB-first bit-parallel finite field multiplier (b) Basic cell and architecture for MSB-first bit-parallel finite field multiplier

3.1 Bit-Serial/Parallel Array Multipliers

Bit-parallel Pipelined LSB-First Semisystolic Multiplier. In [11], a bit-level pipelined parallel-in parallel-out LSB-first semisystolic multiplier is proposed based on `Algorithm 1`. The basic cell, which computes $a_j^{(i)}$ and $t_j^{(i)}$ at `Step i` is shown in Fig. 1(a) with the architecture of an m-bit multiplier. The dotted lines indicate cutsets where latches are placed to pipeline it. This implementation comprises m^2 basic cells and $3m^2$ l-bit latches. The latency of this implementation is $m+1$ clock cycles. From Fig. 1(a), we see that the critical path consists of one two-input AND gate followed by one two-input XOR gate, i.e. $T_{2-AND} + T_{2-XOR}$.

Bit-parallel Pipelined MSB-First Semisystolic Multiplier. Based on `Algorithm 2`, a bit-level pipelined MSB-first semisystolic multiplier is implemented as shown in Fig. 1(b) which was reported in [11, 18]. The basic cell in Fig. 1(b) computes $a_j^{(i)}$ and $t_j^{(i)}$ at `Step i`. Unlike the LSB-first scheme, $a(x)$ is multiplied by b_{m-1} first instead of b_0. Moreover, $a(x)$ in $(i-1)$-th iteration ($a_j^{(i-1)}$ bits) need not be latched according to the algorithm. The critical path is limited by the delay

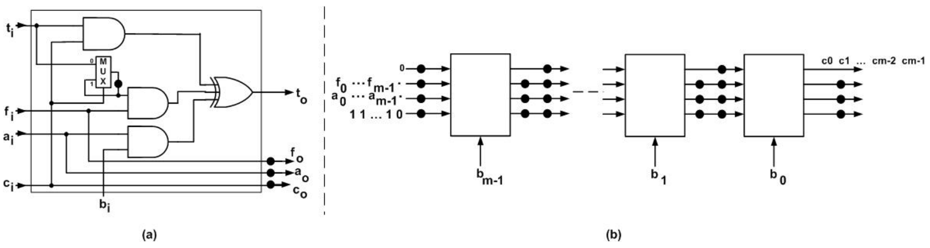

Fig. 2. (a) Basic cell and (b) Bit-serial MSB-first finite field multiplier

in the basic cell and is given by $T_{2-AND} + 2T_{2-XOR}$. Hence the delay is longer than LSB-first multiplier by T_{2-XOR}. Moreover, the number of cycles to complete the operation remains the same, i.e., $m + 1$ clock cycles. In addition, the number of 1-bit latches is reduced to m^2 in MSB-first implementation because only $t_j^{(i)}$s have to be latched in each basic cell.

Serial-in Serial-out MSB-First Multiplier. In [12], a serial-in serial-out multiplier is described. It is also based on the MSB-first algorithm. The basic cell for this architecture is shown in Fig. 2(a). It is a modified version of the basic cell shown in Fig. 1(b), which was used in the MSB-first parallel architecture. The two XOR gates in Fig. 1(b) are replaced with a 3-input XOR gate. There is an additional multiplexer MUX and an AND gate. The extra circuitry enables the signals to propagate in a serial-in serial-out fashion. It should be noted that the subscript i for t_i, f_i, a_i and c_i in Fig. 2(a), indicates the input bit position rather than the iteration number. • represents a delay element. c_i is the control signal to select the inputs of the multiplexer. The serial-input is controlled by a control sequence 0111· · ·11 with length m. The first bit, *0* of the control sequence enters the array one cycle ahead of the MSB of $a(x)$. This *0* control bit selects the MSB of $t(x)$ of the $(i-1)$-th iteration to be ANDed with f_j. Thereafter, when the control signal sets to *1*, the MSB of $t(x)$ of the $(i-1)$-th iteration is latched back into the multiplexer as shown in Fig. 2(a). In this way, the equation, $t_j^{(i)} = t_{m-1}^{(i-1)} f_j + b_{m-i} a_j + t_{j-1}^{(i-1)}$ in `Algorithm` 2 for the calculation of $t_j^{(i)}$ (i.e., the intermediate product bit j at the i-th iteration) is realized for a serial-input.

Fig. 2(b) shows the serial-in serial-out architecture of an m-bit multiplier in $GF(2^m)$. It comprises m basic cells and has a latency of $3m$ clock cycles with a throughput rate of $1/m$. The combinational delay is equivalent to $T_{2-AND} + 2T_{2-XOR}$ (a 3-input XOR gate is considered as equivalent to two 2-input XOR gates). The logic complexity of the basic cell is given by three 2-input AND gates, one 3-input XOR gate, nine 1-bit latches and one switch (multiplexer).

3.2 Digit Serial Systolic Multipliers

Two digit-serial systolic multipliers are described in [13] and [14]. Both of them are based on the MSB-first algorithm. Here we highlight some of the salient

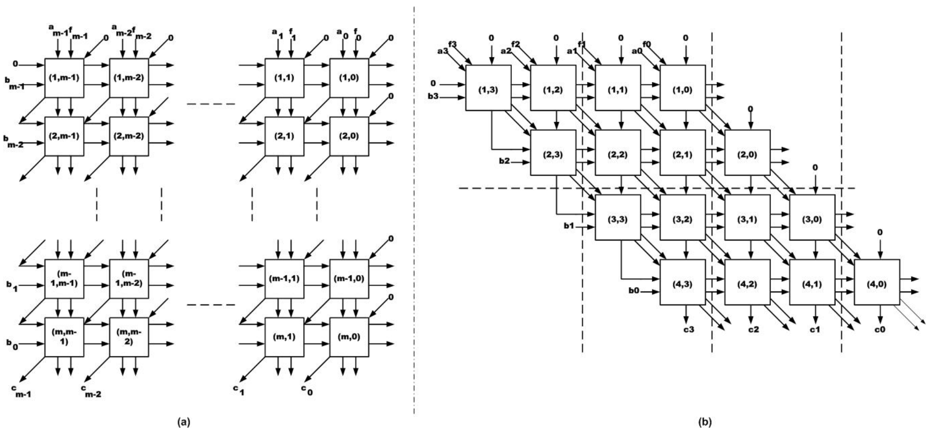

Fig. 3. (a) DG as in [13] (b) DG as in [14]

features of these architectures. The dependency graph (DG) shown in Fig. 3(a) is modified in [13] and [14] to create a digit-serial multipliers. The problem encountered in both architectures is how to project the DG in the east direction to obtain a one-dimensional signal flow graph. In [13], for a digit length L, a basic module comprising L^2 cells is selected. These cells form a square grid of L cells in the x and y directions. Extra circuitry consisting of 4-input XOR gates is used in the basic module to overcome the bidirectional signal flow in the DG of Fig. 3(a). The derivation of the architecture and the basic module are detailed in [13]. For a digit size of L, each basic cell has a logic complexity of $L-1$ 2-input XOR gates, $2L^2+L$ 2-input AND gates, $L-1$ 4-input XOR gates, $10L$ 1-bit latches and $2L$ switches (2-to-1 multiplexers) [13].

In [14], DG is modified to prevent the use of 4-input XOR gates. The authors transform indices of the DG to form a new DG where each row in DG (see Fig. 3(a)) is shifted towards the right by one basic cell, i.e., Cell (2, $m-1$) is placed under Cell (1, $m-2$) instead of Cell (1, $m-1$). The new DG is then divided into m/L parts horizontally where each part comprises L rows. Fig. 3(b) shows the DG partitioning for a 4-bit multiplier where $m=4$ and $L=2$. Each part is further divided into $m/L+1$ regions vertically. Due to the transformation of indices, the regions have different number of cells. Each basic module as shown in [14] has L^2 cells. In contrast, the cutsets in Fig. 3(b) do not form equal regions. Latches and multiplexers are introduced between cells in the basic module to accommodate the cutsets. More information pertaining to the basic module and cutsets can be obtained from [14]. The complexity of the basic module is given by $2L^2$ 2-input AND gates, L^2 3-input XOR gates, $8L+2$ 1-bit latches and $2L$ switches.

3.3 Generalized Cellular-Array Multipliers

In [11], two generalized cellular-array multipliers are proposed based on the LSB-first and MSB-first algorithms presented earlier. The prefix 'generalized' refers

f_i	y_{in}	h_i	y_{out}
0	0	0	0
0	1	0	1
1	0	1	1
1	1	0	1

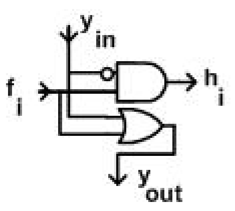

Fig. 4. Truth table to compute H vector and LCELL

to the applicability of the structure to varying field order, i.e., the multiplier is designed for multiplication over $GF(2^M)$ but the primitive polynomial $f(x)$ is of order m such that $m \leq M$. The higher order bits of primitive polynomial are padded with zeros.

The authors of [11] first determine the order of the primitive polynomial by bit-locator cells – LCELL. The LCELL is derived from the truth table of Fig. 4 as given in [11]. The H vector from LCELLs is an M-bit binary string with only one non-zero bit. The only *1* bit corresponds to the field order. For example, if $p(x) = x^5 + x + 1$ and $M = 9$, then $H = 00010000$. This is followed by modifying both algorithms to incorporate programmability of field order m. The array is now made up of MCELLs (multiplier cells), which are based on the modified algorithms in [11]. Based on the architectures presented in [11], the complexity of the basic cell in the generalized LSB-first array multiplier is increased by four gates compared to the fixed order LSB-first multiplier described in Sec. 3.1. Similarly the complexity is increased by 3 gates in MSB-first programmable multiplier compared to its fixed order counterpart. LCELLs form extra circuitries that are needed for the precomputation for the generalized architectures. One disadvantage of generalized architectures is that the critical path is longer. It runs vertically in the array due to signals propagated vertically by y_{out} in the array. As a result, the clocking frequency decreases. The critical path is equivalent to $(m-1)T_{2-OR}$ in both cases.

4 ASIC Implementation Results

In this section, we present simulation results of the architectures discussed in Sec. 3. These simulation results give a better estimate of VLSI performance metrics – silicon area, critical path delay and dynamic power dissipation. Structural VHDL codes were generated automatically using C programs to facilitate more rigorous simulations of different architectures under varying field size. The designs were synthesized, optimized and simulated using Synopsys Design Compiler version 2004.06. The finite field multipliers were optimized with consistent constraints set in Design Compiler. The input and output loads were set to 0.8pF and 0.9pF respectively. Cross boundary optimization was enabled to harness further area

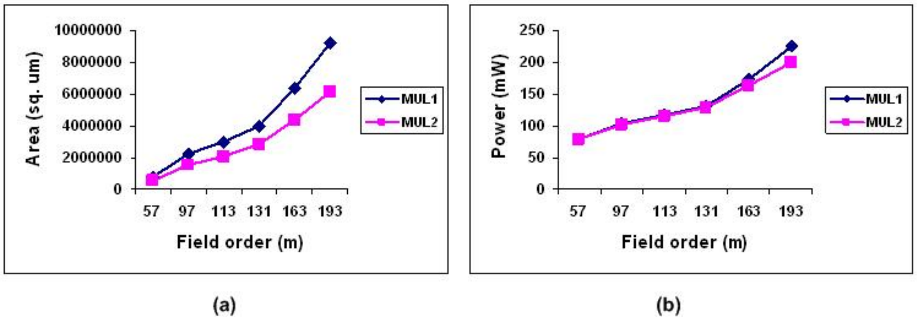

Fig. 5. (a) Area results and (b) Power results of MUL1 and MUL2

reduction. Power simulations were performed using Synopsys Power Compiler with a set of thousand random input vectors at a clock frequency of 20MHz. The area results are also obtained at this clock frequency. However, the circuits can meet this timing constraint comfortably. Thus, they can be further optimized for faster clocks at the expense of higher area. Nine different derivatives of finite field multipliers from Sec. 3 were simulated.

MUL1 and MUL2 represent the parallel-in parallel-out LSB-first and MSB-first multipliers respectively from [11]. The bit-serial architecture in [12] is named MUL3. Two additional bit-serial multipliers MUL4_BS and MUL5_BS, were obtained from the digit serial multipliers stated in [13] and [14] by setting the digit size to 1. MUL1, MUL2, MUL3, MUL4_BS, MUL5_BS, MUL4_DS and MUL5_DS were developed for different field orders 113, 131, 163 and 193 as specified by SEC2 [17]. The strength of ECC with a key length of 193 bits is equivalent to that of a 1536-bit key length in RSA [17]. The generalized bit-parallel architectures reported in [11] – MUL6 and MUL7, were designed for $M = 193$ as that is the maximum order we are interested in this paper. In addition to the above field orders, the first seven multipliers for lower field orders - 57 and 97 were also implemented.

Bit-level parallel-in parallel-out implementations [11]: MUL1 and MUL2 fall in this category. Fig. 5(a) and (b) show the comparison of the silicon area and dynamic power dissipation of MUL1 and MUL2, respectively. From Fig. 5(a), we see that MUL1 (LSB-first) occupies higher silicon area than MUL2 (MSB-first). The difference is attributed mainly to the extra latches in MUL2. It was shown previously in Sec. 3 that MUL2 employs m^2 latches whereas MUL1 has $3m^2$ latches. However, there is a marked increase in the difference between the two designs as m increases. As m increases towards 193, the area of MUL2 approaches that of MUL1 with a lower field order. From Fig. 5(b), we see that both MUL1 and MUL2 dissipate around the same amount of dynamic power with MUL2 having a slight edge over MUL1. This shows the ascendancy of MSB-first algorithm in terms of silicon area and dynamic power dissipation, particularly for higher field orders. The critical path delays of both designs are comparable and is approximately 1.36 ns.

Serial-in serial-out implementations [12, 13, 14]: MUL3 in [12] is a bit serial multiplier whereas those reported in [13] and [14] are digit serial multipliers. Thus

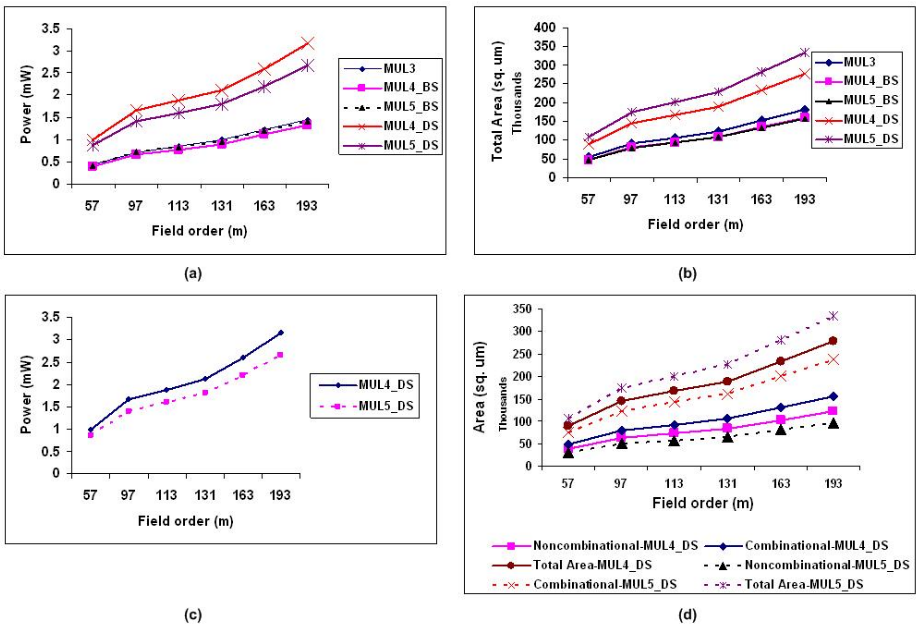

Fig. 6. (a) Power and (b) Area results of bit-serial multipliers (c) Power and (d) Area results of digit-serial multipliers

based on the architectures of [13] and [14], we create four variants, bit-serial multipliers – MUL4_BS, MUL5_BS and digit-serial multipliers – MUL4_DS and MUL5_DS. The delay in all the bit-serial implementations is equal to 1.36 ns as the critical path is $T_{2-AND} + 2T_{2-XOR}$ in all the designs. In contrast to the bit-serial implementations, the delay of digit-serial multipliers MUL4_DS and MUL5_DS is 4.16 ns and 9.18 ns, respectively for a digit length of 8.

The serial multipliers are compared for power and area in Fig. 6(a) and Fig. 6(b), respectively. MUL3, MUL4_BS and MUL5_BS show almost equal power results for all field orders. MUL3 occupies more area than MUL4_BS and MUL5_BS. It is interesting to note that the bit-serial multipliers, MUL4_BS and MUL5_BS, which are the bit-serial derivatives of the digit-serial implementations, outperform MUL3 in terms of area which was designed with an intention to operate as a bit-serial multiplier. As expected, digit-serial implementations are worse off compared to their bit-serial counterparts.

Fig. 6(c) and (d) compare power and area of the two digit serial implementations, MUL4_DS and MUL5_DS. The two multipliers were compared at gate level in [14]. The ASIC implementation of the two designs shows that MUL4_DS is better than MUL5_DS in terms of area but worse off with respect to dynamic power dissipation. In Fig. 6(d), combinational, noncombinational and total areas of MUL4_DS and MUL5_DS are compared. The dotted lines correspond to the area components of MUL5_DS and the solid line curves are used for those of

Table 1. Results for MUL6 and MUL7

Design	Critical path delay (ns)	Silicon area ($\mu\, m^2$)	Dynamic power (mW)
MUL6	29.34	16358291	341.2509
MUL7	29.55	16678856	342.4379

MUL4_DS. Although MUL5_DS has lower non-combinational area, its outweighing combinational area makes it less area efficient than MUL4_DS. This somewhat unexpected synthesis result also explains the higher critical path delay of MUL5_DS (9.18 ns as opposed to 4.16 ns of MUL4_DS).

Generalized bit-level parallel implementations [11]: MUL6 and MUL7 are the generalized bit-parallel implementations of LSB-first and MSB-first algorithms reported in [11] which are implemented for the highest order that is considered in this paper, i.e., M = 193. Table 1 shows the results of the two designs. The results show no significant deviation in all VLSI metrics between the two implementations. If these results are compared with the fixed order bit-parallel architectures of MUL1 and MUL2, the critical path delay is higher. Though MUL6 and MUL7 are configurable in terms of field order, they are an order of magnitude (21 times) slower than their fixed order counterparts due to the vertical critical path discussed previously. The area of MUL6 and MUL7 is also higher than MUL1 and MUL2 (1.77 and 2.72 times higher, respectively) due to the extra circuitry (LCELLs) needed to determine the actual field order m.

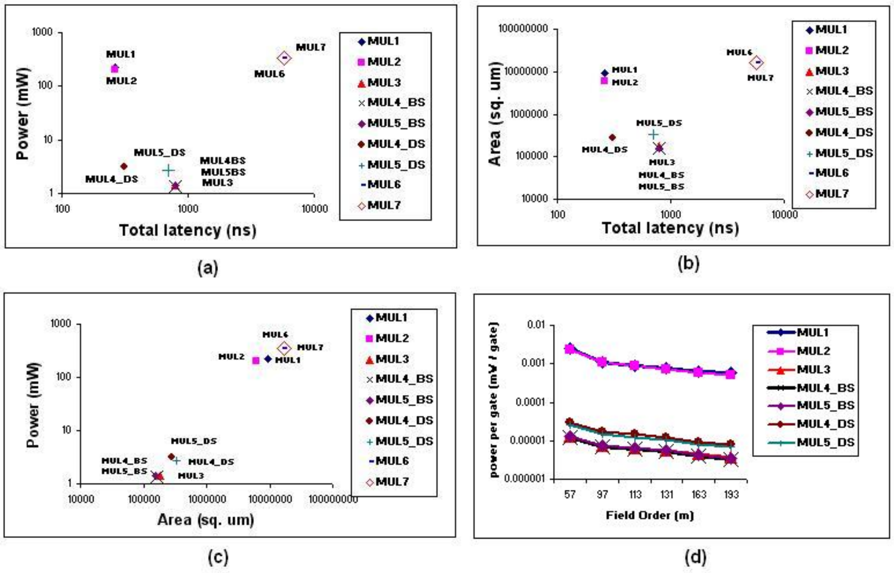

Fig. 7. Consolidated Results

Consolidated results - power vs. total latency: Fig. 7(a)shows a scatter plot of power dissipation against latency of all finite field multipliers discussed in this paper for field order of 193. The axes are scaled logarithmically for clarity. This plot provides the system architect with a choice of finite field multipliers in a large design space to trade off between power and performance. For fair proposition that involves both bit serial and digit serial multipliers, latency instead of worst case delay is used. It refers to the product of the worst case delay and number of cycles to complete a single multiplication. Serial implementations outperform parallel multipliers in terms of power dissipation. Despite having similar critical path delays (1.36 ns) due to the use of similar basic cells, the bit-serial multipliers show thrice as much total latency as the bit-parallel multipliers. This is attributable to the difference in the number of clock cycles ($3m$ clock cycles for bit-serial multipliers and $m + 1$ clock cycles for bit-parallel multipliers) needed to complete one finite field multiplication operation. MUL4_BS has twice the total latency of its digit-serial counterpart, MUL4_DS. However, the total latency of MUL5_BS is around 1.13 times that of MUL5_DS. This contrast between digit-serial multipliers is because of the critical path delays. Generalized bit-parallel multipliers MUL6 and MUL7, have a total latency of 21 times their fixed bit-parallel counterparts MUL1 and MUL3. This is due to the chain of LCELLs used to determine the field order. In terms of power dissipation, parallel implementations consume on average, around 78 times higher power than the serial implementations.

Consolidated results area vs. total latency: Silicon area of each multiplier is plotted against latency in Fig. 7(b). Serial multipliers again occupy the lower region of the graph with digit-serial multipliers occupying higher area than bit-serial implementations. Bit-serial multipliers, MUL3, MUL4_BS and MUL5_BS, on an average use only 1/58 the area of bit-parallel architectures, MUL1 and MUL2. Digit serial multipliers, MUL4_DS and MUL5_DS, occupy about twice the area of their bit-serial designs, MUL4_BS and MUL5_BS. Generalized bit-parallel designs occupy more area than fixed bit-parallel multipliers. The area of MUL6 (LSB-first architecture) is double that of MUL1, and the area of MUL7 (MSB-first architecture) is around 3 times that of MUL2.

Consolidated results power vs. area: Fig. 7(c) shows the relationship between the average power dissipation and area of each implementation. All serial multipliers are cluttered in the low power, lower area region whereas the parallel implementations occupy the other extreme.

Consolidated results power per gate vs. field order: In Fig. 7(d), we show the power consumed per gate against field order for all multipliers except generalized parallel implementations. Power per gate is calculated by dividing the average dynamic power by the number of gates used for the implementation. The total number of gates is determined from the total area divided by the area occupied by one two-input $NAND$ gate in TSMC 0.18μm technology library. Though total power increases as field order increases, power consumed by each gate is lower as the field order increases as evinced by the consistent trends of all designs in Fig. 7(d). The bit parallel multipliers exhibit higher power consumption per gate than the bit serial or digit serial multipliers. This implies the increased probability of spurious computations in some gates with more parallelism.

5 Conclusion

In this paper, we consolidated some recently reported systolic and semisystolic finite field multipliers. A brief description of algorithm to architecture mapping for each implementation was shown followed by the comprehensive simulations of their ASIC prototypes based on TSMC 0.18μm technology library. We compared their performances based on various criteria such as silicon area, dynamic power dissipation and critical path delay and overall latency. It was shown that bit-parallel architectures consume higher area and power than bit-serial implementations. The two digit-serial architectures in [13] and [14] were also reduced to bit serial variants for analysis. Our evaluation also revealed the agility of generalized bit-parallel architectures to suit varying field order is achieved with a severe penalty of performance and area overhead. With the emphasis on low power design of mobile computing and performance per unit cost of smart card application, this review provides a good insight into the potential design space exploration of the array-type finite field multipliers. Future research would involve studying these architectures to overcome their shortcomings and on their resilience against differential power and differential time attacks when they are used to implement the ECC engine.

References

1. B. Schneier, *Applied Cryptography.* 2nd Edition, Wiley 1996.
2. A. Menezes, *Elliptic Curve Public Key Cryptography.* Kluwer Academic Publishers, 1993.
3. D. Hankerson, J. L. Hernandez, A. Menezes, Software Impelementation of elliptic curve cryptography over binary fields, *Cryptographic Hardware and Embedded Systems*, pp. 1-24, Springer-Verlag, Heidelberg, 2000.
4. H. Eberle, N. Gura, S. Chang-Shantz, A cryptographic processor for arbitrary elliptic curves over $GF(2^m)$, in *Proc. IEEE Intl. Conf. on Application-Specific Systems, Architectures, and Processors*, Hague, Netherlands, pp. 444-454, June, 2003.
5. M.A. Hasan, M.Z.Wang, V.K. Bhargava, A modified Massey-Omura parallel multiplier for a class of finite fields, *IEEE Trans. on Computers*, vol. 42, no. 10, pp. 1278-1280, Oct. 1993.
6. T. Zhang, K.K. Parhi, Systematic design of original and modified mastrovito multipliers for general irreducible polynomials, *IEEE Trans. on Computers*, vol. 50, no. 7, pp. 734-749, July 2001.
7. E.D. Mastrovito, VLSI designs for multiplication over finite fields $GF(2^m)$, in *Proc. Sixth Intl. Conf. Applied Algebra, Algebraic Algorithms, and ErrorCorrecting Codes (AAECC-1988)*, Rome, Italy, pp. 297-309, July 1988.
8. A. Halbutogullari, C. K. Koc, Mastrovito Multiplier for general irreducible polynomials, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Lecture Notes in Computer Science Series No. 1719*, pp. 198-507, Springer-Verlag, Berlin, 1999.

9. A. Reyhani-Masoleh, M. A. Hasan, Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$, *IEEE Trans. on Computers*, vol. 53, no. 8, pp. 945-958, Aug. 2004.
10. S. K. Jain, K. K. Parhi, Low latency standard basis $GF(2^m)$ multiplier and squarer architectures, in *Proc. IEEE Intl. Conf. on Acoustic, Speech and Signal Processing (ICASSP-1995)*, Detroit, Michigan, USA, pp. 2747-2750, May 1995.
11. S. K. Jain, L. Song, K. K. Parhi, Efficient semisystolic architectures for finite-field arithmetic, *IEEE Trans. on Very Large Scale Intergration (VLSI) Systems*, vol. 6, no. 1, pp. 101-113, Mar. 1998.
12. C.-L. Wang and J.-L. Lin, Systolic array implementation of multipliers for finite fields $GF(2^m)$, *IEEE Trans. on Circuits and Systems-I*, vol. 38, no. 7, pp. 796-800, July 1991.
13. J.-H. Guo and C.-L. Wang, Digit-serial systolic multiplier for finite fields $GF(2^m)$, *IEE Proc. Comput. Digit. Tech.*, vol. 145, no. 2, pp. 143-148, Mar. 1998.
14. K.-W. Kim, K.-J. Lee, K.-Y. Yoo, A new digit-serial multiplier for finite fields $GF(2^m)$, in *Proc. of Info-tech and Info-net (ICII 2001)*, Beijing, China, vol. 5, pp. 128-133, Oct. 2001.
15. C. H. Kim, S. D. Han, C. P. Hong, An efficient digit-serial systolic multiplier for finite fields $GF(2^m)$, in *Proc. of 14th Annual IEEE Intl. ASIC/SOC Conference*, pp. 361-165, Sept. 2001.
16. J. Lopez and R. Dahab, An overview of elliptic curve cryptography, *Technical Report*, Institute of Computing, State Uniersity of Campinas, May 2000.
17. Certicom Research, SEC 2: Recommended elliptic curve domain parameters, *Standards for Efficient Cryptography*, Technical document, Sept. 20, 2000.
18. B. A. Laws, C. K. Rushforth, A cellular-array multiplier for $GF(2^m)$, *IEEE Trans. on Computers*, vol. C-20, pp. 869-874, Nov. 1982.
19. R. Lidl, H. Niederreiter, *Introduction to finite fields and their applications*, Revised Edition, Cambridge University Press, 1994.
20. N. Nedjah, L. de Macedo Mourelle, A reconfigurable recursive and efficient hardware for Karatsuba-Ofman's multiplication algorithm, in *Proc. IEEE Int. Conf. on Control Applications (ICCA 2003)*, Istanbul, Turkey, vol. 2, pp. 1076-1081, June 2003