Efficient Low-Energy, Digit-Serial Exponentiator for large Finite Field $GF(2^m)$

F. A. Cherigui, D. Mlynek, *Member, IEEE*

***Abstract*-- This paper presents a new area efficient, bit-level pipelined, linear systolic array architecture performing exponentiation over large Finite Field $GF(2^m)$. It is based on one multiplier and implements the square-and-multiply algorithm. The architecture is regular, expendable to any field order and programmable with respect to field-generating polynomial $P(x)$. It allows the input elements to enter a linear systolic array in the same order and the system only requires one pipelined control signal. The operations are overlapped at higher system frequency in order to reduce the total delay of the exponentiation computation. A systematic approach is applied for implementing a digit-serial architecture in order to reduce power consumption. The resulting circuit is highly regular and programmable with respect to $P(x)$. An analysis of the performance comparison is described as function of the digit-size. A comparison is made with the bit serial architecture based on the performance improvement with respect to computation delay and power/energy consumption of one exponentiation. Thus, the factor of merit, which could be a measure of performance, is defined as the product of *energy times the delay* and it is computed. The experimental results on gate level implementations shows that the resulting circuit is low-energy.**

***Index Terms*--Galois fields, Systolic arrays, Switching power, Energy-Delay product, High performance circuits.**

I. Introduction

Many popular public-key schemes including ElGamel encryption and signature schemes [2], Diffie-Hellman key-distribution Scheme [3] and other variant of discrete log based cryptosystems [4] relies on exponentiation in finite field, either over integers modulo a prime odd p (implementation over $GF(p)$), or a polynomial field (implementation over $GF(2^m)$). When first introduced as underlying finite field, $GF(2^m)$ was the preferred implementation, basically because it is easier to implement in hardware [5], [8]. Further, all practical public-key schemes require operations in relatively large finite fields; e.g., $m > 500$ including in particular schemes based on the intractable discrete logarithm in finite fields [6][7].

The $GF(2^m)$ exponentiation is a difficult task to carry out efficiently in software for large finite field since it is a time consuming operation. Furthermore, the VLSI architectures exhibit in general a great activity and dissipate consequent shares of the power supply due to the long arithmetic operators. Reducing power consumption is equally important for non-portable applications as it reduces cooling and packaging costs and increases system reliability. Therefore, for physical security and performance reasons hardware implementations of area efficient, low-energy Galois field exponentiator are very attractive, especially for these architectures, which are programmable with respect to polynomial field-generator $P(x)$ and expendable to any field order.

The usual approach to reduce the time complexity and improve the performance is to use parallel architecture. However, the hardware complexity (area and energy consumption) of bit-parallel architectures increases dramatically as the field order m increase since large number of gates and registers are mapped. Digit-serial technique an alternative to the bit-parallel, process multiple bits (digit) of an entire word, referred to as the *digit-size*, in one clock-cycle. This technique is suitable for the implementation of moderate sample rate systems where, the area and power consumption are critical. In this paper, we demonstrate that highest gain can be achieved on the behavioral and architectural levels (up to 90% saving of energy delay product) using digit-serial technique to implement partially parallel architecture and rearranging the gate topology from array-type to tree-type.

II. Finite Field Fundamentals

Knowledge of basic Finite Field concepts and properties is assumed, as covered in [8], [14].

Finite Field $GF(2^m)$ contains 2^m elements. It is an extension field of $GF(2)$, which contains two elements {0,1}. The element of $GF(2^m)$ can be represented in several equivalent forms. Mainly, there are three common types of bases, *Standard or Polynomial Basis* (SB/PB), *Normal Basis* (NB) and *Dual basis* (DB). There are many polynomial bases and normal bases from which to choose. For efficient computation of the field arithmetic we generally use an *optimal normal basis representation* or a *polynomial basis representation.*

If a standard basis $\{1, \alpha, \cdots, \alpha^{m-1}\}$ is used, where the primitive element α is a root of an irreducible polynomial of degree m, $P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0$ over $GF(2)$, then each element can be represented as a polynomial in α with a degree

Manuscript received August 2nd, 2000.
The authors are with the Electrical Engineering Department, Swiss Federal Institute of Technology, Lausanne, CH-1015 Switerland (e-mails: Feth-Allah.Cherigui@epfl.ch, Daniel.Mlynek@epfl.ch).

less then *m*, or

$$GF(2^m) = \left\{ A \middle| A = \sum_{i=0}^{m-1} a_i \alpha^i,\ a_i \in GF(2),\ 0 \le i \le m-1 \right\}$$

In addition, the operation results of additions, multiplications and exponentiation of element α are still polynomials of α with degree less than *m*. In this base, addition is defined as integer addition modulo-2 (logical XOR) and multiplication is defined as integer multiplication modulo-2 (logical AND). Element of the field represented by a normal basis $\left\{\alpha, \alpha^2, \alpha^4, ..., \alpha^{2^{m-1}}\right\}$, are expressed as polynomials of degree 2^{m-1} or less, or

$$GF(2^m) = \left\{ A \middle| A = \sum_{i=0}^{m-1} a_i \alpha^{2^i},\ a_i \in GF(2),\ 0 \le i \le m-1 \right\}$$

Since elements in one representation can be efficiently converted to elements in the other representation by using an appropriate change-of-basis matrix, the intractability of the DLP isn't affected by the choice of representation.

Unlike addition and multiplication, exponentiation in $GF(2^m)$ is similar to the operation for ordinary integers. Given an arbitrary element $A = \sum_{i=0}^{m-1} a_i x^i \in GF(2^m)$, and *Y* a nonzero element in $GF(2^m)$ the exponentiation function is defined as

$$Y = A^E,\ 0 \le E \le 2^m - 1 \tag{1}$$

A popular algorithm for computing this operation, and the one that appears to be most suitable for hardware implementation, is the square-and-multiply (S&M) algorithm as described below,

Let $(e_0, e_1, \cdots, e_{m-2}, e_{m-1})$ be the binary representation of the exponent *E* such that $E = \sum_{i=0}^{m-1} e_i 2^i$. Then, by eq. (1) we have

$$Y = \prod_{i=0}^{m-1} \left(A^{2^i} \right)^{e_i} = \prod_{i=0}^{m-1} U_i \tag{2}$$

where

$$U_i = \begin{cases} A^{2^i} & \text{if } e_i = 1 \\ 1 & \text{if } e_i = 0 \end{cases} \tag{3}$$

Thus, the algorithm breaks the exponentiation operation into series of squaring and multiplication operations in $GF(2^m)$ and can be expressed as follow:

S&M algorithm

Let U_i be the square term and M_i the product term.

$U_{-1} = A$

$M_{-1} = 1$

for $i = 0, \cdots, m-1$

$$U_i = \left[U_{i-1} \right]^2 \tag{4}$$

$$M_i = \begin{cases} 1 \cdot M_{i-1} & \text{if } e_i = 0 \\ U_{i-1} \cdot M_{i-1} & \text{if } e_i = 1 \end{cases} \tag{5}$$

The final result is $Y = A^E = M_{m-1}$

Let T_s be the setup time of the exponentiator (i.e. time corresponding to serial data transfers) and *N* the number of multiplication operations (including squaring operations) per exponentiation. The exponentiation time T_e can be expressed as $T_e = T_s + NR^{-1}T_{clk}$, where *R* is the throughput rate of the multiplier(s) and T_{clk} is the clock period. In terms of computation effort, assuming a random input, the number of ones in the exponent is $m/2$, so *N* can be expressed as $N = (m-1) + m/2 \approx 3m/2$ for large *m*. However, using synchronous exponentiator *N* becomes $(m+1) + m \approx 2m$ since the multiplication in eq. (5) for $e_i = 0$ has to be carried out.

III. Low Power Architectures for Large GF(2^M) Exponentiation

One new method for computing $GF(2^m)$ exponentiation is presented in [9]. It is based on pattern matching and recognition technique. The resulting circuit is a multistage linear static pipeline and it can produce one result every clock cycle. However it has latency of $O(2^m)$. The area and complexity increase systematically when large field is used. This technique is more suitable for small field-size ($m \le 8$) as mentioned in the paper. Instead, S&M algorithm also called the binary method [10] breaks the exponentiation operation into a series of squaring and multiplication operations in $GF(2^m)$. It is the most adopted technique to design large finite field exponentiation.

A. Reducing number of operations

Some interesting approaches have been proposed in [15] and [16] in order to reduce substantially the average number of multiplications involved in the computation of A^E and thus reducing the power consumption using the canonical bit recording technique (or signed digit SD number representation). Technique in [15] can be easily applied to the exponentiation over $GF(2^m)$. Efficient implementation of α^E over $GF(2^m)$ using SD technique and based on bi-directional linear feedback shift registers is proposed in [16]. However, the algorithm is limited to the exponentiation of a Primitive Root α and generally the SD technique require the construction of canonical signed-digit vector of the exponent *E* which, introduce much extra power and area cost since one additional bit is required for each scanned exponent bit. Also, both techniques described in [15] and [16] require the availability of the inverse A^{-1}. The cost of this operation far exceeds the time gained by the use of the SD technique.

B. Speeding up the computation

The conventional approach for speeding up the exponentiation over $GF(2^m)$ and reducing the number of operations uses a lookup table (LUT). This method consist of precomputing the field elements A^{2^i} (A conjugates) in eq.(2) and storing them in circulating registers or in a RAM (area complexity ~ m^2) and multiplied together according to the exponent using fully parallel multiplication tree [8][20]. For large m the multiplication have to be performed using bit-serial multiplier (area complexity ~ m for both PB and NB). Furthermore, setup times corresponding to serial data transfers of the exponent and A's conjugates bits have to be considered which increase the latency to less than one exponentiation per m^2 clock cycles. Thus, no area and timing gain is obtained using this approach even when m is moderately large. Another approach relies on the Montgomery multiplication in $GF(2^m)$ [17] as a fast method for multiplying two polynomials. However, the algorithm includes some similar operations as the bit-serial SB multiplication algorithm [8] or digit-serial one [12] with more processing steps. This approach is more suitable for implementation in software as claimed in the paper and do not offer any obvious advantages for hardware implementation.

C. Bases choice

Since the exponentiation algorithm relies on polynomial multiplication in $GF(2^m)$, the design of area efficient low-energy finite field multipliers can lead to dramatic improvement on the overall performance of $GF(2^m)$ exponentiator. Table 1 shows some key parameters for different architectures of serial $GF(2^m)$ multipliers using different bases.

It is well known that squaring operation in normal basis NB can be achieved by a cyclic-shift circuit [8][18]. However multiplication in NB requires the computation of the f function described in [18] which must be found by computer and hardwired. The complexity of this function grows dramatically as the order of the field goes up. Massey and Omura have developed a NB multiplication algorithm, which has been implemented in a pipelined architecture by Wang *et al.* [18] based on an AND-XOR PLA and has a throughput rate of one multiplication per m clock cycles. As m increases, the signal propagation delay across the PLA also increases which increase the critical path. So, the area and total delay may actually be larger owing to the slower clock rate.

From Table 1, it is clear that normal basis multiplication using the serial Massey-Omura multiplier in [8] requires the least number of gates, unfortunately this is for optimal normal basis multipliers which can be realized for only ~23% of the fields $GF(2^m)$, $2 \le m < 1200$ [21]. Another point to consider is that the NB multiplier is not highly modular and expendable and is not programmable with respect to the $P(x)$ due to the f function which depends on the choice of normal basis thus, on the choice of $P(x)$.

TABLE 1. KEY PARAMETERS FOR DIFFERENT ARCHITECTURES OF SERIAL $GF(2^M)$ MULTIPLIERS USING DIFFERENT BASES

Multiplier/Basis	Complexity	Length of CP	Latency min/max	Regularity	Easy to expand
MSR/SB [8][19]	$6m/4m$	3*	$m/2m$	High	Yes
LSA/SB [11]	$17m$	5	$3m/3m$	High	Yes
Berlekamp/DB [23]	$\ge 6m/4m$	$\ge 2+\lceil \log_2 m \rceil$	m/m	Low	No
MO/NB [18]	$\ge 5m$	$\ge 3+\lceil \log_2 m \rceil$	m/m	Low	No

* Critical path in term of gates but in fact it is determined by the driving capability of heavily loaded signals that are distributed through the architecture.

On the other hand, the DB multiplier requires basis conversion, which requires extra gates. Although the Berlekamp DB multiplier [23] has a simple and regular structure it is not trivial to adapt it to different choice of $P(x)$ or to expand it to different field order because of the presence of two different bases dual and polynomial. Also the critical path depends on the logic for computing the inner product, which is, depends strongly on the field order and $P(x)$. This means that the speed decreases, although slowly, with m.

Even the PB multipliers have the most favourable properties for purpose of VLSI implementation, the critical path of the MSR multiplier described in [8] is limited by the driving capabilities of heavily loaded signals which degrades the performance as demonstrated in [12]. On the other hand, the linear systolic array (LSA) multiplier described in [11] and [12] shares some interesting characteristics with the MSR multiplier. It has the advantage of efficient implementation time at the cost of increased complexity in term of gates.

D. Power optimization

The total power dissipated in a CMOS gate with a capacitive load C_{load} is given by

$$P = \frac{1}{2} \cdot C_{load} \cdot V_{DD}^2 f \cdot N + Q_{sc} \cdot V_{DD} \cdot f \cdot N + I_{leak} \cdot V_{DD} \qquad (6)$$

Where V_{DD} denotes the voltage swing, and f is the frequency of operation, N the activity factor, i.e., the number of gate output transitions per clock cycle. The factor Q_{sc} represents the quantity of charge carried by the *short circuit* current per transition and I_{leak} is the leakage current.

In traditional design the average power consumption of a CMOS gate is dominated by the switching activity (dynamic power) and contributes to more than 90% of the total power consumption [23], but this may change for future developments of high-scaled integration [24]. As the device size and threshold voltage continue to decrease, the short circuit power dissipation is no longer a negligible factor since the delay increases.

Reducing the power consumption amounts to the reduction of one or more of these factors. In energy-efficient design, we seek to minimize the energy consumed per operation or the power-delay product of the circuit, which is the factor of merit for high performance architectures.

Lower supply voltage can achieve extremely low power consumption eq. (6). However, the delay time is proportional to $1/V_{DD}$ as expressed by the propagation delay equation of a CMOS circuit given by [25],

$$T_{delay} = \frac{C_{load} \cdot V_{DD}}{k \cdot (V_{DD} - V_t)^2} \tag{7}$$

Where k depends on the transistors aspect ratio (W/L) and other device parameters, V_t is the transistor threshold voltage. Lowering supply voltage leads to performance degradation since delays drastically increase as V_{DD} approaches the threshold voltages V_t of the device. The supply voltage can be reduced to a certain value, so that the chosen frequency matches with the longest critical path thus, when the propagation delay T_{delay} is less than the clock period T_{clk} by a factor δ, we can reduce the supply voltage by a factor β such that T_{clk} is equal to T_{delay}. Hence,

$$T_{clk} = \delta . T_{delay}(V_{DD}) = T_{delay}(\beta \cdot V_{DD}) = \frac{C_{load} \cdot \beta \cdot V_{DD}}{k \cdot (\beta \cdot V_{DD} - V_t)^2} \tag{8}$$

Parallelism and pipelining can be exploited to improve the performance (to compensate for the increased gate delays) of low-voltage circuits [23]. Also, much higher reductions in power consumption are possible when using clock-gating technique in order to reduce the activity factor N in eq. (6).

IV. LSA BASED ARCHITECTURE FOR $GF(2^M)$ EXPONENTIATION

In this section we present a new exponentiator circuit based on a linear systolic array multiplier LSA over $GF(2^m)$ and implements *S&M Algorithm.* The successive SB squaring and multiplication operations are performed at high system frequency a using only one multiplier in order to reduce the area complexity of the exponentiator. For VLSI implementation of high performance architecture, a digit-serial technique is applied on the multiplier in order to reduce both the switching activity and total power thus, reducing the energy-delay products of the exponentiator at the expense of increased area.

A. Linear Systolic Array Exponentiator

The LSA multiplier architecture described in [11] and [12] is bit-level pipelined and has a longest delay path independent of m. It is most suited to applications where m is large or where high clock frequencies are required. Further, it allows the input elements to enter a linear systolic array in the same order and the system only requires one pipelined control signal. The architecture is also highly regular, expendable to any field order and programmable with respect to the primitive polynomial $P(x)$.

Furthermore, the input operands (MSB-first), the control signal as well as the primitive polynomial coefficients are outputted at each pipelined stage. These signals with the result (MSB-first) could be feed-backed into the multiplier in order to perform successive multiplication and squaring operations according to *S&M* algorithm. In fact, LSA multiplier cells contain registers configured in master slave manner in which, internal data registers are used to give one more time unit delay to the operands at each cell [11]. The first output bit of the multiplication result is available after $2m$ clock cycles. After m clock cycles the multiplier operands are completely loaded into the architecture allowing the load of the operands for the next operation (squaring or multiplication) while performing the computation of the current operation. Thus, using the LSA multiplier, operations in eq. (4) and (5) could be overlapped targeting to reduce the latency and thus the exponentiation time T_e. The general structure of the LSA based exponentiator is depicted in Fig. 1 where, p_i for $0 \le i \le m-1$ are the primitive polynomial coefficients and S is a control signal (only the first bit among the m bits contains the value one).

The circuit diagram of the multiplier cells is shown in Fig.2. Two internal registers b and c are used to hold the MSB inputs of B and C operands along the operation using s_{in} signal, which mark the start of the multiplication when $s_{in} = 1$. These coefficients are then used to compute the cell output at the next clock cycle when $s_{in} = 0$. Therefore, three registers a, p and s are used to give one time unit delay to the input bit coefficients a_{in}, p_{in} and s_{in} at each cell and the output coefficients are then triggered using a next register output stage in master slave manner as shown in Fig. 2. For more details about the algorithm please refer to [11] and [12].

The m-bit multiplication time takes $3m$-1 clock cycles. At $2m$ clock cycles after a_{m-1} and b_{m-1} enter the leftmost cell the results C_{out} in Fig. 1 will start coming out from the rightmost cell at the rate of one coefficient (MSB-first) every clock cycle.

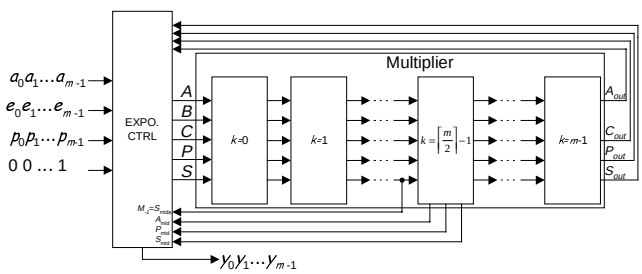

Fig. 1. LSA multiplier based architecture for exponentiation in GF(2m) when m is odd.

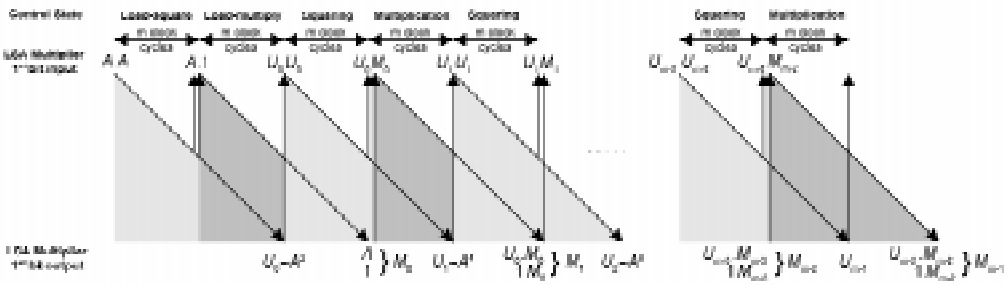

Fig. 4. Overlapping the squaring and multiplication operations.

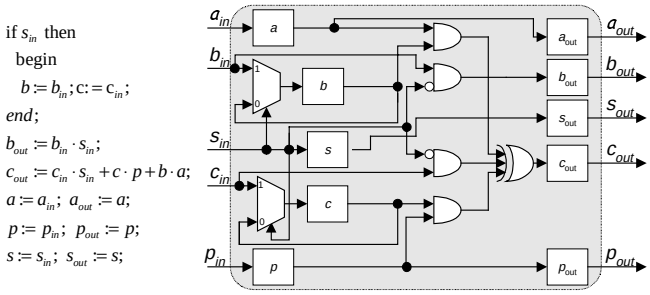

Fig. 2. LSA basic processing cell and its algorithm.

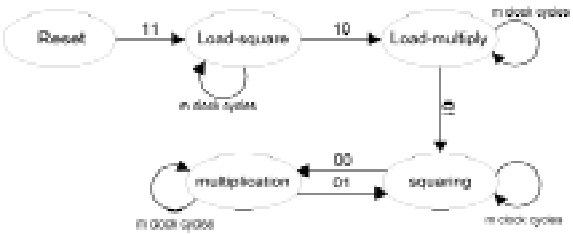

Fig. 3. The exponentiator operation modes.

The squaring is performed continuously using circuit shown in Fig. 1. The feedback signal outputted at CELL-$\lceil m/2 \rceil - 1$ contains the square term (operand) A_{mid}, which is used for computing the product $U_{i-1} \cdot M_{i-1}$ in eq. (5) according to the primitive polynomial coefficients P_{mid} and control signal s_{mid}. If m is odd, these feedback signals are outputted from the internal registers of CELL-$\lceil m/2 \rceil - 1$ as shown in Fig. 1., otherwise, they are taken at the output of this cell. The squaring and multiplication results C_{out} are outputted iteratively from the last cell and used for computing the expression in eq. (4) and (5) according to the primitive polynomial coefficients P_{out} and control signal S_{out}. The signal S_{mida} in Fig. 1 is the multiplier control signal S that defines the operating modes of the exponentiator, it generate two internal control signals, *load* and *square*, which defines 4 different operating modes as shown in Fig. 3. If m is even then S_{mida} is outputted from the internal register s of CELL-$\lceil m/2 \rceil - 1$. It is also used as $GF(2^m)$ unit operand M_{-1} for the computation of the $U_{-1} \cdot M_{-1}$ in eq. (5). A transition in this signal at the input of CELL-$\lceil m/2 \rceil - 1$ means that the LSB of the inputs operands are fed into the multiplier allowing the load of next operands. Note that the multiplication $1 \cdot M_{i-1}$ ($1 \le i \le m-1$) in eq. (5) is not performed. We can cope with this using the feedback operand A_{out} as the product term M_{i-1}.

We start the computation in *load-square* mode. In this mode we perform a squaring operation on the value of the base *A*. Once the coefficients are loaded into the multiplier after *m* clock cycles, we switch the exponentiator to the *load-multiply* mode. In this mode we perform the multiplication operation $U_{-1} \cdot M_{-1}$. Once the M_{-1} coefficients are loaded into the multiplier after *m* clock cycles, we switch to the *squaring* mode. In this mode we perform the computation in eq. (4), then we switch to the *multiplication* mode in which we perform the computation in eq. (5). We repeat successively these two last operating modes until the end of operation. This is described in the diagram reported in Fig. 4.

For signals stability, different clock edges are used to handle the feedback signals in Fig. 1. Control states are then generated within the clock pulse and feedback signals are triggered by the inverted clock at the exponentiator controller. The exponent *E* is fed into a LIFO stack (buffer) of size *m*. The buffer content is shifted one time when a test in eq. (5) is performed. Hence, the buffer is completely empty after the output of the final result.

The setup time T_s of the exponentiator is equal to *m* clock cycles corresponding to serial data transfers of the inputs. The throughput rate *R* of the multiplier is equal to one multiplication operation per *m* clock cycles. Thus, at *m* clock cycles, after the LSB of the inputs *A*, *E*, *P* and *S* enter the exponentiator architecture, the exponentiation result Y will start coming out from the rightmost cell after $2m \cdot m$ clock cycles at the rate of one bit per clock cycle. Hence, the exponentiation time T_e becomes $2(m+1) \cdot m \cdot T_{clk}$. Compared with the MSR based exponentiator described in [8], witch perform the computation in $(m+2) \cdot m$ clock cycles from the first-in bit to the last-out bit, the latency of the LSA based exponentiator is increased by m^2 clock cycles. However, the MSR based multiplier presents long heavily loaded signals that increase the critical path and the total delay for one multiplication operation becomes 96% higher compared to LSA multiplier [12]. Thus, the latency gain of the MSR based exponentiator is counterbalanced by the clock period.

Furthermore, the area complexity of both exponentiators is almost the same ~17*m*.

A. Digit-Serial Linear Systolic Array Exponentiator

The generated digit-serial architecture is obtained from the LSA digit-serial multiplier. The resulting exponentiator circuit is linear array-type at the digit-level based on the parallel multiplication algorithm inside of each digit cells. The corresponding algorithm is obtained by grouping each set of *D* cells from the LSA multiplier in Fig. 1, then computing the outputs of each of these grouped cells after *D* steps (clock cycles). The algorithm of the LSA digit-serial multiplier have been detailed in [12].

The successive squaring and multiplication operation are then implemented in digit-serial manner using the architecture depicted in Fig. 5. The operands are digit-word inputs. Thus, the inputs bit a_i, e_i and p_i for $0 \le i \le m-1$ in Fig.1., are replaced by digits forms A_i, P_i, and E_i for $0 \le i \le d-1$ where

$$(A_i, P_i, E_i) = \begin{cases} \sum_{j=0}^{D-1} (a_{Di+j}, p_{Di+j}, e_{Di+j}) \cdot x^j, & 0 \le i < d-1 \\ \sum_{j=0}^{m-1-D(d-1)} (a_{Di+j}, p_{Di+j}, e_{Di+j}) \cdot x^j, & i = d-1 \end{cases} \quad (6)$$

and

$$(A, P, E) = \sum_{i=0}^{m-1} (a_i, p_i, e_i) \cdot x^i = \sum_{i=0}^{d-1} (A_i, P_i, E_i) \cdot x^{Di} \quad (7)$$

where, *D* denotes the digit-size and *d* the total number of digits $d = \lceil m/D \rceil$.

The exponent *E* is expressed in binary form and fed into a LIFO stack (buffer) of size $\lceil m/D \rceil D$. The input digit-words A_i, E_i and P_i, are fed into the multiplier in the same order for i deceasing and from the MSB to the LSB. If *m* is not divisible per *D*, the zero padding is performed at the LSB positions for $i = 0$ for A_i, and P_i, and MSB positions for $i = d-1$ for the exponent E_i. The *s* signal is used to denote the start of a multiplication.

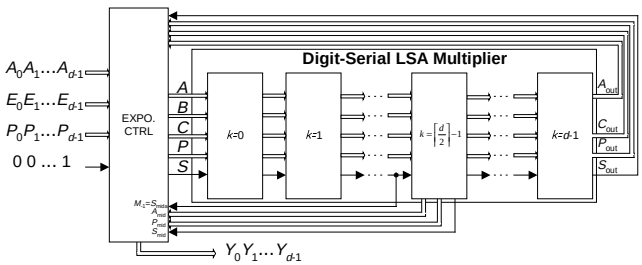

Fig. 5. Digit-Serial LSA multiplier based architecture for exponentiation in GF(2^m) .

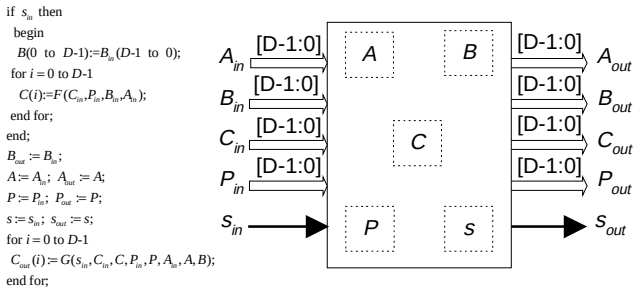

Fig. 6. LSA digit-serial basic processing cell and its algorithm.

The exponentiator operates as the previously described bit-serial LSA based where each CELL-k of the multiplier process one digit of an entire word in one clock-cycle. Note that the architecture shown in Fig 5., is still programmable with respect to the primitive polynomial *P*(*x*).

The basic processing element CELL-K of the multiplier is shown in Fig. 6. Two *D*-bit registers *A*, *P* and 1-bit *s* registers are used to give one time unit delay to the input data A_i, P_i and S_i at each CELL-k. In Fig. 6, *F* denotes the function processing the state of the *C* internal register; its circuit diagram is shown in Fig. 7., where *FF* denotes a flip-flop. Note that the critical path is proportional to the digit-size and expressed as $(D\text{-}1)(T_{xor\text{-}3}+T_{NAND\text{-}2})$. The *G* function process the state of the output register C_{out}. The corresponding circuit diagram for one bit output coefficient is shown in Fig. 8. The critical path in this architecture becomes $DT_{XOR\text{-}3}+T_{XOR\text{-}2}+2T_{NAND\text{-}2}$.

At *d* clock cycles after the least significant digit-words A_0, E_0, P_0 and the LSB of *s* enter the exponentiator architecture shown in Fig. 5, the result Y_i will start coming out after $2m \cdot d$ at the rate of one digit every clock cycle. Thus, the exponentiation time T_e takes $2(m+1) \cdot d$ clock cycles.

V. MPLEMENTATION AND COMPARISON

Clock gating technique can be used for power-efficient implementation of registers that are disabled during some clock cycles, when such registers maintain the same value through multiple cycles such as the internal slave registers *c* and *b* in Fig. 2 and *C* and *B* in the digit-serial cell shown in Fig. 6. These registers have their own load controlled by s_{in} signal. This technique works well for data-flow logic, where clocking requirements can be predetermined at least one cycle ahead. Thus, the clock gating enable signal s_{in} must be valid halfway into the cycle to gate off the capture clock. To overcome this problem, we require that these internal registers be triggered faster than the master registers c_{out} and b_{out} (B_{out} and C_{out} respectively) using different clock edges that is pipelining within the clock cycle. This requires one more clock pulse, resulting in 2-phase non-overlapping clocking scheme.

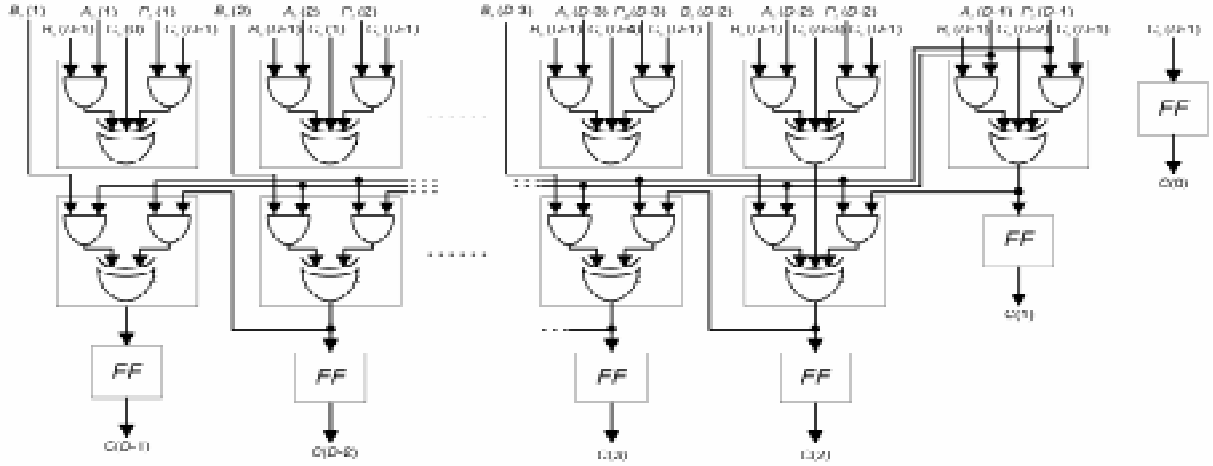

Fig. 7 Circuit diagram of the *F* function.

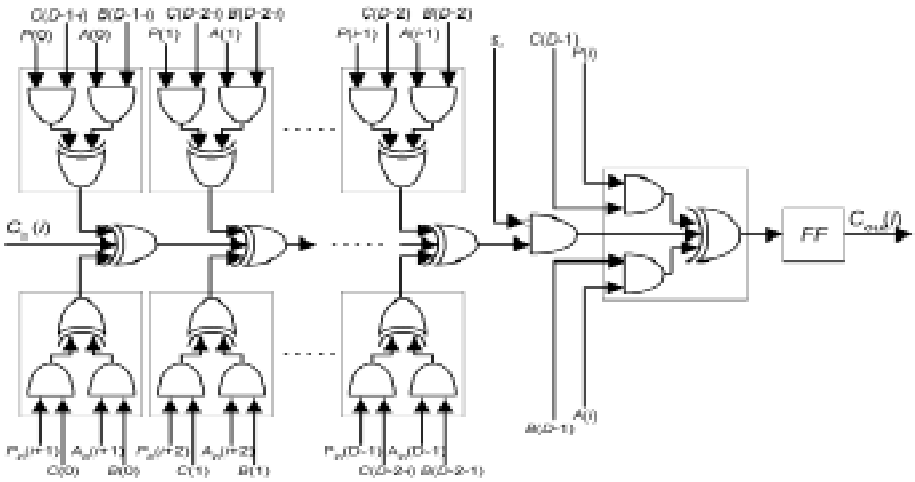

Fig. 8. Circuit diagram of the G function for one bit output.

The free and clock gating LSA exponentiator have been implemented at the gate level using different digit-size D=1,4,8 in order to perform a comparison in terms of delay, area and energy consumption for one exponentiation over $GF(2^{607})$ with $P(x)=1+x^{273}+x^{607}$ as primitive polynomial. These architectures are mapped on TSMC CMOS 0.18µm technology at a targeted supply, V_{DD}=1.8V *and* 0.9V. A low power design-flow has been validated using Synopsys tools for power analysis and optimization. Different types of power dissipation components are estimated using gate level simulations on a set of random stimulus. Since *low-energy* design is more important than *low-power* design, the energy and energy-delay product is computed. The performance characteristics including total delay, area in term of gates and energy-delay product are reported in Fig. 9, 10 and 12 respectively. The cost function is reported in Fig. 13 and defined as the Energy-Delay-Area products versus the digit-size.

On the one hand, the critical path in the architecture shown in Fig. 2 is just the sum of one full-adder and one NAND gate delay. For larger digit-size the critical path is limited by the *F* function in Fig. 7 and increases linearly with the digit-size to $(D\text{-}1)(T_{XOR\text{-}3}+T_{NAND\text{-}2})$. In the same time, the latency decreases linearly with the digit-size in almost the same rate resulting in a constant total delay as shown in Fig. 9.

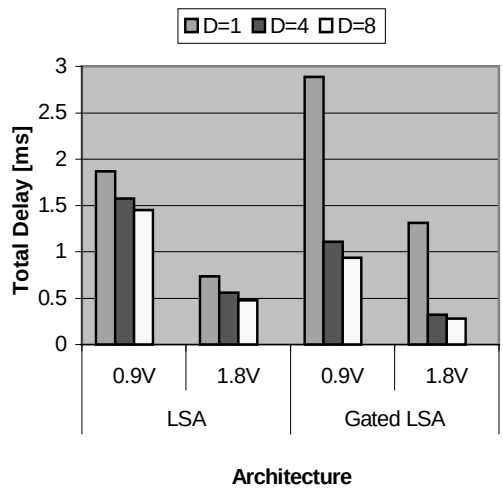

Fig. 9. Total Delay as function of the digit size for one LSA digit-serial $GF(2^{607})$ exponentiation.

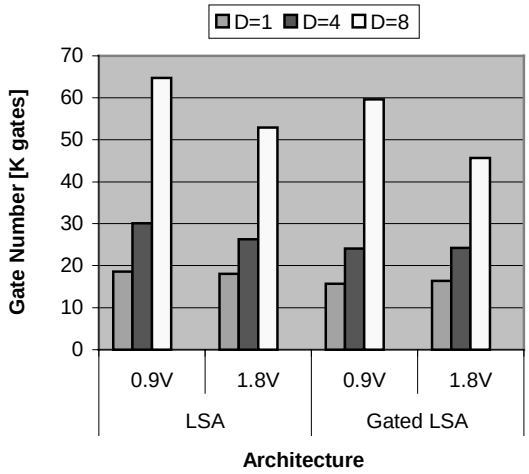

Fig. 10. Area in gates of the LSA digit-serial GF(2^{607}) exponentiator as function of the digit size.

On the other hand, Fig. 10 shows that the area increases dramatically with the digit-size due to the large number of logical gates and latches used to temporary hold the internal and the output data in master-slave manner. This means that the level of parallelism is limited by the area constraints (digit-size).

Clock gating technique achives a substantial reduction in both the total delay and number of gates, for digit-size equal or larger then 4. It helps to eliminate the feedback loops and multiplexers used to feed the output of each internal storage elements back to the input for synchronous load-enable. Such feedback loops are replaced by only one integrated cell with latch based clock gating, thus the area is reduced in all cases but for the bit-serial architecture (D=1) these clock gating celles (latches) add a substential delays which increase the critical path.

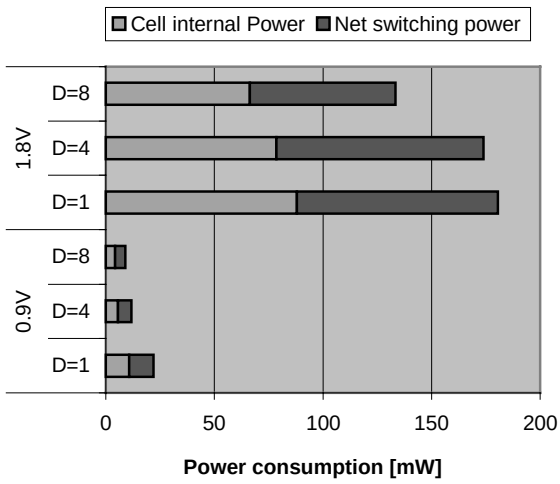

Fig. 12. Different types of power dissipation components as function of the digit size of the LSA digit-serial GF(2^{607}) exponentiator.

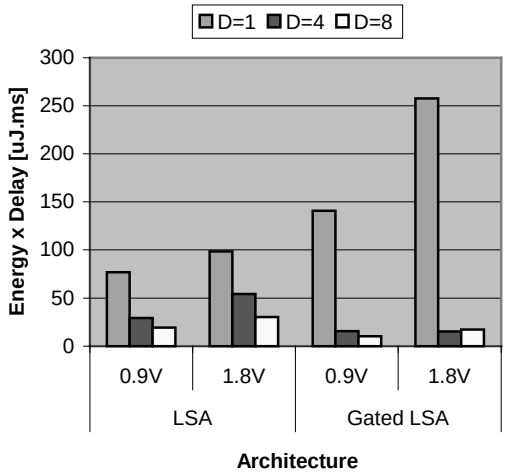

Fig. 12 Energy-Delay product as function of digit-size of the LSA digit-serial GF(2^{607}) exponentiator.

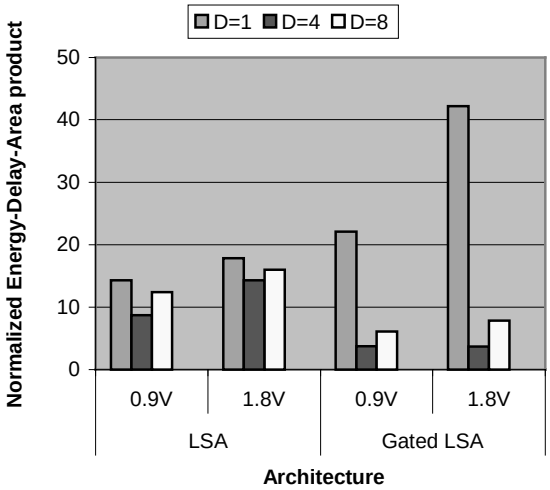

Fig. 13 Energy-Delay-Area product as function of the digit size of the LSA digit-serial GF(2^{607}) exponentiator.

If the switching power contributes to more then 90% (dominant factor) then the power consumption can be reduced by factor β^2 when reducing the operating voltage by a factor β eq. (6). From eq. (7) the delay is increased by factor β assuming that $V_{DD} >> V_t$. Therefore, the power saving is counterbalanced by the increased delay since the energy-delay product is proportional to δ^2. This is not the case for V_{DD}=1.8V and β=2 since V_{DD} is close to the threshold voltage V_t hence the delay increases and the short circuit power is no longer a negligible factor (switching power is no longer the dominant factor) as shown in Fig. 11.

The most interesting result is obtained when comparing the Energy-Delay and the cost function versus the digit-size. The performance characteristic reported in Fig. 12, shows that Energy-Delay products are significantly reduced when digit-size increase. High gain is obtained for D=8 when

operating at 0.9V and more than 75% reduction is noticed (and more than 92% when gating the clock) since the power is significantly reduced when operating at lower voltage. However, when comparing the characteristic reported in Fig. 13 (cost function), the optimum is obtained for D=4 when operating at both 1.8V and 0.9V due to the dramatic increase in circuit area for larger digit-size. The highest gain is obtained when gating the clock as the number of gates and delay are substantially reduced.

I. Conclusion

This paper presents a new area efficient (~17m) linear systolic array (LSA) architecture implementing exponentiation over large $GF(2^m)$ according to square-and-multiply algorithm in witch the operations are overlapped in order to reduce the latency to $2(m+1)m$. The architecture relies on Standard Basis, bit-level pipelined multiplier that is expandable to any field order. Also, the exponentiator is programmable with respect to the primitive polynomial $P(x)$.

In order to reduce the energy, we extended the LSA bit serial exponentiator to a generalized digit-serial architecture obtained by unfolding the bit-serial multiplier. The resulting circuit is array-type at the digit-level using parallel multiplication algorithm inside of each digit cells. Consequently latency is reduced to $2(m+1)d$ where d is the number of digits and the energy delay products are significantly reduced at the expanse of increasing area. The cost function defined as the energy-delay-area product versus the digit-size D has an optimum for D=4 with up to 55% saving of energy-delay product and more 94% when gating the clock.

References

[1] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press LLC, 1997.

[2] T. ElGamal, "A public key Cryptosystem and a signature scheme based on discrete logarithms", IEEE Transactions on Information Theory, vol IT-31(4), pp 469-472, July 1985.

[3] W. Diffie, M. E. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, 22:644-654, November 1976.

[4] William M. Rake, "The RPK Public-Key Cryptographic System", Technical Summary, 1993-1996.

[5] B. Schneier, "Applied Cryptography", Second Edition, Wiley, 1996.

[6] A. M. Odlyzko, "Discrete Logarithms in finite fields and their cryptographic significance", EUROCRYPT, 1984, pp. 224-147.

[7] A. M. Odlyzko, "Discrete Logarithms: The past and the future", Design, Codes and Cryptography, 19(2/3), 2000, pp. 129-147.

[8] E. D. Mastrovito, "VLSI Architectures for Computations in Galois Fields", PhD thesis, Linköping University, Departement of Electrical Engineering, Linköping, Sweden, 1991.

[9] M. Kovac, N. Ranganathan, "ACE: A VLSI Chip for Galois Field Efficient GF(2m) Based Exponentiation", IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 43, No. 4, pp. 189-297, April 1996.

[10] D. E. Knuth, "The Art of Computer Programming: Seminumerical Algorithms", Volume2, Addison-Wesley, Second edition, 1981.

[11] B. B. Zhou, "A New Bit-Serial Systolic Multiplier Over $GF(2^m)$", IEEE Transactions on Computers, Vol. 37, No. 6, pp. 749-751, June 1988.

[12] F. A. Cherigui, D. Mlynek, "Low Energy Digit Serial Architectures for large GF(2m) multiplication", IEE Transactions on Circuit-Systems and Devices, 2001.

[13] C hin-Liang Wang, "Bit-Level Systolic Array for Fast Exponentiation in $GF(2^m)$", IEEE Transactions on Computers, vol. 43, no. 7, pp. 838-841, July 1994.

[14] M. R. Schroeder, "Number Theory in Science and Communication", Vol. 2, Springer-Verlag, 1986.

[15] Ö. Egecioglu, ç. K. Koç, "Exponentiation using Canonical Recoding", Theorical Computer Science, 129(2): 407-417, 1994.

[16] H. Wu, A. Hasan, "Efficient Exponentiation of a Primitive Root in $GF(2^m)$", IEEE Transactions on Computers, Vol. 46, No. 2, pp. 162-172, February 1997.

[17] ç. K. Koç, T. Acar, "Fast Software Exponentiation in $GF(2^k)$", Proceedings, 13th Symposium on Computer Arithmetic,

[18] C. C. Wang et al., "VLSI architecture for computing multiplications and inverses in $GF(2^m)$", IEEE Transactions on Computers, Vol. C-34, No. 8, pp. 709-717, August 1985.

[19] P. A. Scott, S. E. Tarvares and L. E. Peppard, "A fast multiplier for $GF(2^m)$", IEEE J. Select. Areas Commun., Vol. SAC-4, Jan 1986.

[20] P. A. Scott, S. J. Simmons, S. E. Travers, L. E. Peppard, "Architecture for exponentiation in $GF(2^m)$", IEEE Transactions on Computers, Vol. 6, No. 3, pp. 578-586, April 1988.

[21] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in $GF(p^n)$", Discrete Applied Mathematics, pp. 149-161, 1988/1989.

[22] S. T. J. Fenn, M. Benaissa, D. Taylor, "$GF(2^m)$ Multiplication and Division Over the dual Basis", IEEE Transactions on Computers, Vol. 45, No. 3, pp. 319-327, March 1996.

[23] A. P. Chandrakasan and R. W. Brodersen, "Low Power Digital CMOS Design", Kluwer Academic Publishers, 1995.

[24] Y. Taur, Y. J. Mii, D.J. Frank, H.S. Wong, D.A. Buchanan, S.J. Wind, S. A. Rishton, G.A. Sai-Halasz and E.J. Nowak, "CMOS scaling into the 21st century: 0.1 μm and beyond", IBM Journal of Research and Development, vol. 39, no. ½, Jan/Mar 1995, pp. 245-260.

[25] A. Bellaouar, M. Elmasry, "Low-Power Digital VLSI design: Circuits and Systems", Boston, Massachusetts, Kluwer Academic Publishers.