Journal of Low Power Electronics
Vol. 8, 1–12, 2012

A Low-Power Parallel Architecture for Finite Galois Field GF(2^m) Arithmetic Operations for Elliptic Curve Cryptography

Esmaeil Amini[1,*], Zahra Jeddi[1], Ahmed Khattab[2], and Magdy Bayoumi[1]

[1] *The Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA 70504, USA*
[2] *Department of Electronics and Electrical Communications Engineering, Cairo University, Egypt*

(Received: 16 February 2012; Accepted: 8 June 2012)

In this paper, a parallel, power-efficient and scalable word-based crypto architecture is proposed that performs the operations required for scalar point multiplication including add, multiplication and inversion operations on GF(2^m) operands. The proposed architecture distinguishes itself from exiting architectures, including our prior architecture, by the fact that its resource usage and power-consumption is based on the input data. Hence, such architecture might be used for various operand sizes without modifying or reconfiguring the underlying hardware. Besides, the architecture has the ability to perform several different operations in parallel when each operation requires a small key size which significantly increases the overall performance and throughput of the system. In the absence of parallel requests, the remaining unused modules will be turned off in order to save power. The experimental results show significant improvement in the timing, throughput and energy performances with a slight overhead in the circuit area.

Keywords: ECC, Public Key Encryption, Parallel Architecture, GF(2^m), Field Arithmetic, Modular Multiplication, Modular Inversion.

1. INTRODUCTION

Public key encryption systems such as Rivest-Shamir-Adleman (RSA)[1] and Elliptic Curve Cryptography (ECC)[2,3] play a vital role in contemporary secure systems as they ensure the privacy, message integrity, authenticity, and non-repudiation requirements of secure communications. The security strength of any encryption algorithm depends on its key size.[4] ECC is becoming increasingly the promising public-key method because it uses shorter keys at a security level equivalent to other public-key algorithms like RSA.[4]

For real-time applications such as SSL server connections, a software implementation of an ECC scheme may not provide the desired performance level and hardware implementations are needed as they better meet the performance requirements. There are many factors which must be considered in the hardware implementation of ECC system designs such as:

• *Power consumption as it is becoming the main design constraint by the advent of portable and contactless devices.*
• *Flexibility as the device is neither tied to a certain ECC curve nor needs a specific type of irreducible polynomial.*
• *Scalability as the device can provide various levels of security without changing the underlying hardware. For instance, the system supports different operand sizes to accommodate different levels of security. Otherwise, the system will become outdated within a short period of time.*
• *Throughput of the system is a very important factor in cases the system likely needs to handle thousands of operations per second like servers, router gateways, etc.*

The operation that dictates the execution time of an elliptic curve cryptographic protocol is the point multiplication of the Galois Fields (GF) operands, and its hardware implementation would have a significant impact on the system performance.[4]

One of the other crucial parameters in the implementation of ECC architectures is the type of the underlying finite field upon which the elliptic curve operations are based. ECC implementations could either use prime fields GF(p) of binary finite fields GF(2^m) in which field elements are usually represented as binary polynomials. The latter ones are often chosen for hardware realizations as they require smaller hardware circuits for implementation.[5]

*Author to whom correspondence should be addressed.
Email: exa2685@cacs.louisiana.edu

1.1. Related Work

Due to the increased significance of cryptography, several public key cryptographic hardware for GF(2^m) have been proposed in the literature. The authors of Ref. [6] have targeted a compact architecture that performs three different cryptographic algorithms: RSA, ECC and paring-based cryptography. They have tried to improve the time/area metrics by designing reusable functional units which can be shared among different modules. They have used the similarity of these three cryptographic algorithms for their basic arithmetic operations such that it allows diverse utilizations of the functional units in the design. However, they have not reported any results to show how much their design improved these metrics. The authors of Ref. [7] proposed an architecture that supports arbitrary operand sizes and provides multiplication, modular squaring and inversion operations. However, the computation time for the modular inversion of that architecture is in the order of $O(m^2)$. In contrast, our proposed architecture performs the inversion operation in the order of $O(m)$. Meanwhile, the architecture proposed in Ref. [8] performs multiplication in GF(2^m) for any value of m less than 256. Such architecture offers the flexibility in selecting the field size at the expense of being limited to only few fixed irreducible polynomials. Furthermore, the proposed architecture does not support the inversion operation. In Ref. [9] a reconfigurable design has been presented which supports variable operand sizes. The advantage of this architecture is its support for both GF(2^m) and GF(p). However, it is restricted to specific types of elliptical curves. An architectures for implementing LSD multipliers for binary fields GF(2^m) is presented in Ref. [10]. In this architecture, internal accumulators have been deployed for storing intermediate results and then, these extra accumulators were used to increase the maximum operating frequency by reducing the critical path delay of the multipliers. However, no hardware implementation report has been given in Ref. [10] and just analytical area and timing reports have been presented. Alternative modular designs have been proposed in Refs. [10, 11] based on offline reconfiguration. For instance,[10] presents a modular Field Programmable Gate Array (FPGA) based architecture that uses very simple hardware components. However, it does not perform the inversion operation. Meanwhile, the architecture presented in Ref. [11] supports both binary and polynomial field multiplication but only considers a specific type of irreducible polynomials and reconfiguration is offline. Hence, it can only be used for modular multiplications and does not support other important crucial operations in ECC, such as inversion and addition. In Ref. [12], the authors have proposed a reordered normal bias multiplier which gives the designer the ability to set a trade-off between the area and speed performance and have implemented their architecture on a 780-pin FPGA circuit. Another reconfigurable design was proposed in Ref. [13] to support various operand sizes for both binary and prime fields. This design implements RSA operations as well. It also supports power-gating approach to reduce the power wastage. However, it requires the data to be aligned before being exchanged with the outside world which complicates the design and increases the delay. A Graphics Processing Units (GPU) implementation is proposed in Ref. [14]. In this work, Least Significant Bit (LSB) invariant scalar point multiplication for binary elliptic curves is implemented on NVidia graphics cards by implementing parallel algorithms for GPU. Hardware/Software Co-Design implementations are also presented in the literature. For instance, the authors of Ref. [15] have proposed Hardware/Software Co-Design of ECC operations on an 8051 Microcontroller. They have tried to minimize the communication overhead due to operand transfers by the integration of a small DMA unit and inclusion of an additional I/O register into the hardware accelerator. Their design supports operations over binary fields of degree up to 192.

1.2. Paper Contributions and Organization

Based on the above discussion of the related literature, only few of the proposed architectures support key size selection feature and also save power for smaller key applications. In our previous work,[17] a modular architecture has been proposed which is not only area efficient but also offers these features. However, when a small key size is used, the proposed architecture sets the other unutilized modules deactivated even if there are other requests to the system, as the case with the aforementioned related architectures. Such a common deficiency makes the system inefficient in terms of throughput and performance when there are many concurrent requests to the system which is a typical feature of secure server systems.

In contrast, this paper alleviates such performance deficiencies by presenting a new parallel architecture that has the unutilized modules simultaneously operating to handle other requests in order to use the system more efficiently and increase the performance and throughput of the system. The contributions of the paper are summarized as follows:

The paper presents a new parallel architecture that is neither tied to a certain ECC curve nor need a specific type of irreducible polynomial. The proposed design significantly reduces the power consumption by deactivating the unused modules by power and/or clock gating. Furthermore, the proposed design handles multiple different operations in parallel in order to increase the overall performance and throughput of the system. A unique feature of the proposed architecture is that it is not restricted to a specific key size for parallel requests. In contrast, it is capable of handling different key sizes depending on the executed parallel operations without the need to explicitly reconfigure the hardware parameters. Besides, the paper provides an exhaustive simulation-based study of the different performance aspects of the proposed architecture. Using randomly generated point doubling requests as a comparison benchmark, it is shown that the proposed architecture imposes slight overhead in power consumption and area while it has significant impact on throughput performance of the system.

The organization of the rest of this paper is as follows: the necessary background discuss of multiplication and inversion algorithms in GF(2^m) is given in Section 2. Then, the base architecture to be used in this paper is briefly discussed in section 3. The new parallel architecture is discussed in Section 4. Simulation setup and results are analyzed in Sections 5 and 6 concludes the paper.

2. MULTIPLICATION AND INVERSION OPERATIONS IN GF(2^m)

Elliptic curve cryptography (ECC) is a public-key cryptography algorithm which is based on the algebraic structure of elliptic curves over finite fields. Point addition/subtraction, point doubling, and scalar point multiplication are geometrically-defined operations for elliptic curves. Two implementation alternatives, namely, prime field GF(p) and binary field GF(2^m) are used for ECC systems. We only discuss the GF(2^m) arithmetic as our proposed architecture is based on it. Equation (1) represents an elliptic curve on binary field GF(2^m):

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where x, y, a, b are m-bit binary numbers and b is nonzero. More details about the underlying theory behind of ECC operations can be found in Ref. [4].

The creation of public key in ECC requires scalar point multiplication on the base point, $\boldsymbol{P}$. Scalar point multiplication could be done by repetitively performing point doubling and point addition. The basic operations for scalar point multiplication boil down to modular addition, subtraction, multiplication, and division on GF(2^m) operands.[4] Addition and subtraction are trivial operations which are simply done by bit-wise XOR operation on operands. Division operation is more complicated than multiplication and inversion and therefore, it is normally substituted with an inversion of divisor followed by a multiplication.[4] In the following subsections, implementing multiplication and inversion operations are briefly discussed.

2.1. Modular Multiplication in GF(2^m)

A modular multiplication in GF(2^m) can be denoted by: $C = A \times B \bmod F$, where A, B and C are m-bit binary polynomials in GF(2^m) and F is an $(m+1)$-bit irreducible binary polynomial for the corresponding GF(2^m). Table I shows the pseudo-code of the algorithm presented in Ref. [11] which is the basis of our proposed architecture too. More details about this algorithm could be found in Ref. [11]. In this paper, a modified version of this algorithm is used in order to support selectable operand size.

2.2. Modular Inversion in GF(2^m)

A modular inversion operation in GF(2^m) finds a multiplicative inverse, A^{-1} of field element, A such that $A \times A^{-1} = 1 \bmod F$, where, F is the irreducible polynomial of corresponding GF(2^m). In Ref. [17], we have presented an architecture that uses the algorithm presented in Ref. [18] as it can be easily modified to achieve selectable key size architecture. The pseudo-code of the algorithm is given in Table II. Both multiplication and inversion algorithms, depend on very simple operations such as shift right, shift left, addition and XOR. Using this feature, a hardware component has been designed that performs both multiplication and inversion operations which saves the system in terms of area and power consumption.

Table I. Modular multiplication algorithm. Adapted with permission from Ref. [11], P. Kitsos et al., An efficient reconfigurable, multiplier architecture for Galois Field GF(2^m). *Elsevier Microelectronics Journal* 34, 975 (**2003**). © 2003.

```
/* This algorithm reads m-bit operands and then
performs the computation in m iteration and
finally outputs the m-bit output.*/
Inputs:
A: m-bit input // multiplicand
B: m-bit input // multiplier
F: m-bit input // modulus; although it is an
               // (m+1)binary number, the MSB
               // is always 1, hencenot
               // needed for this algorithm
Output
C: m-bit output// the result

Read A, B, and F
C = 0, MSB_C = 0
for i = m-1 down to 0 begin
  if (B[m-1] = 1)
    C = C XOR A
  if(i > 0) begin
    MSB_C = C[m-1]
    C = C << 1 //Shift left once
    if(MSB_C = 1)
      C = C XOR F
  endif
  B = B << 1 //Shift left once
endfor
Output C as the result
```

Table II. Modular inversion algorithm. Adapted with permission from Ref. [18], H. Brunner, A. Curiger, and M. Hofstetter, On computing multiplicative inverses in GF(2^m). *IEEE Transactions on Computers* 42, 1010 (**1993**). © 1993.

```
/* This algorithm reads m-bit operands and then
performs the computation in 2m iteration and finally
outputs the m-bit output.*/
Inputs:
A: m-bit input  // operand to be inverted
F: m-bit input  // modulus; although it is an (m+1)
                // binary number, the MSB is always 1,
                // hencenot needed for this algorithm
Output
C: m-bit output// the result
Read A and F
B = 0, C = 1, and Deg = 0
Am = 0, Bm = 0, Cm = 0, Fm = 1 //Initialize MSBs
for i = 1 to 2m begin
  if(Am = 0) then begin
    {Am, A} = {Am, A} << 1 //Shift left once
    {Cm, C} = {Cm, C} << 1 //Shift left once
    Deg = Deg + 1
  endif
  else begin
    if (Fm = 1) begin
      {Fm, F} = {Fm, F} XOR {Am, A}
      {Bm, B} = {Bm, B} XOR {Cm, C}
    endif
    {Fm, F} = {Fm, F} << 1 //Shift left once
    if(Deg = 0) begin
      Swap({Am, A}, {Fm, F})
      Swap(({Bm, B} << 1), {Cm,C}) //left shift, swap
      Deg = 1
    else begin
      {Cm, C} = {Cm, C} >> 1 //right shift once
      Deg = Deg - 1
    endif
  endif
endfor
Output C as the result
```

3. OVERVIEW OF BASIC ARCHITECTURE

In our prior work,[17] a modular multiplier had been designed which implements all the necessary arithmetic operations over GF(2^m) required to do ECC computations. In this architecture, users can select the operand size according to their security needs. Let us assume that we want to design an m-bit multiplier, where m is the maximum operand size that would be needed in the foreseeable future. In this architecture, the m-bit binary operands are split into k smaller n-bit words ($m = k \times n$) as follows:

$A = A_1 + 2^n A_2 + \ldots + 2^{(k-2)n} A_k - 1 + 2^{(k-1)n} A_k$, where $A_i = \sum_{j=0}^{n-1} a_{ij} 2^j$, and a_{ij} are binary bits of multiplicand A

$B = B_1 + 2^n B_2 + \ldots + 2^{(k-2)n} B_{k-1} + 2^{(k-1)n} B_k$, where $B_i = \sum_{j=0}^{n-1} b_{ij} 2^j$, and b_{ij} are binary bits of multiplier B

$F = F_1 + 2^n F_2 + \ldots + 2^{(k-2)n} F_{k-1} + 2^{(k-1)n} F_k$, where $F_i = \sum_{j=0}^{n-1} f_{ij} 2^j$, and f_{ij} are binary bits of modulus F.

Regarding above discussions, the basic multiplier architecture is composed of k modules each has n-bit size as shown in Figure 1. Each of the modules $[0 - k - 1]$ is responsible for operating on one n-bit word of the operand. In this architecture two design parameters must be defined: (i) number of words k, and *(ii)* word-width n. Suppose that $k = 8$ and $n = 8$. If a 24-bit computation is required, then, the first three modules will be activated by controller and remaining modules will remain inactive.

Referring to the multiplication and inversion algorithms (Tables I and II), a mechanism is required for selecting the appropriate MSBs of operands which is one of the

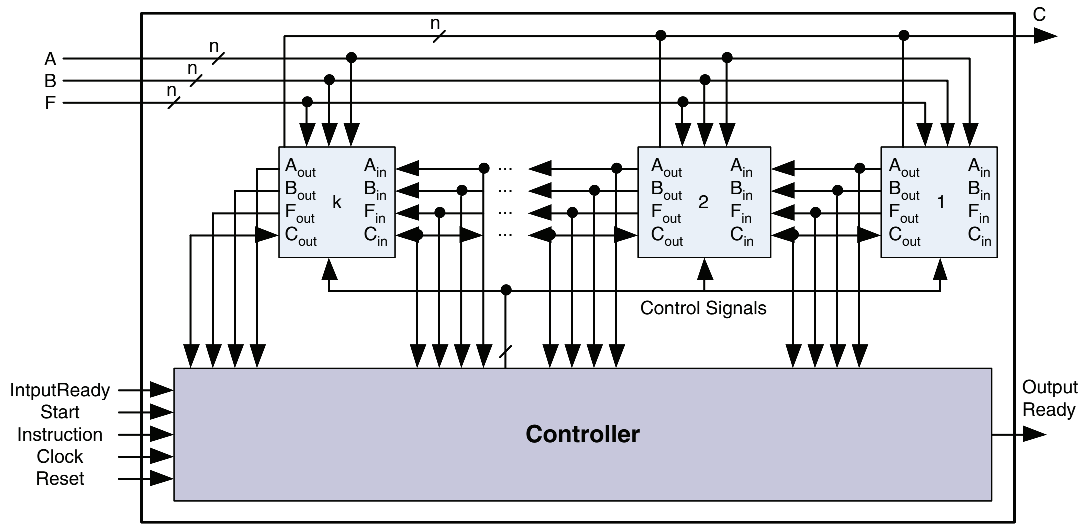

Fig. 1. Block diagram of the basic multiplier architecture.

tasks of controller module. For example, if a three word computation is needed, then the MSBs coming from the third module will be used for the calculation.

Inside each module, there are four shift registers to hold four input/output words of the operands. The MSBs of each register are also connected to the controller, which selects the appropriate MSBs for operands. Besides, each module has an ALU that is responsible for performing various operations such as, XOR, shifting, etc. The controller unit generates appropriate signals to coordinate the operations performed inside this ALU. The block diagram of the controller unit is presented in Figure 2. The controller has three sub-blocks:

• *Load/Store, which is used during the operations of the input and output loops. It keeps a counter, WordCount, to count how many modules are required for a particular computation. It also turns on and off modules by generating appropriate module select signal for each module. During input cycle, the controller also gets "Instruction" signal, which tells the arithmetic processor which operation among add, multiplication, and inversion is requested.*

• *MSB Selector, which is composed of four multiplexers for A, B, F, and C operands. Multiplexers are connected to the MSBs coming from k modules. This module selects the appropriate MSBs based on the value of WordCount register.*

• *Finite State Machine, which controls the operation of the proposed architecture. The controller implements a finite state machine and generates control signals for various sub tasks, such as XOR, shift left, shift right, etc. It should also be noted that during the computation loop, the module select signal selects just the needed modules and keeps the rest turned off in order to save power.*

Choosing $n = 32$ and $k = 8$, the design was synthesized by 45 nm technology cells with clock frequency adjusted to 1 GHz. The main goal in this design was introducing a low-power and area-efficient design. As Figure 3, the power consumption is directly proportional to number of used modules. Hence, the design saves power when a smaller key size operation is requested.

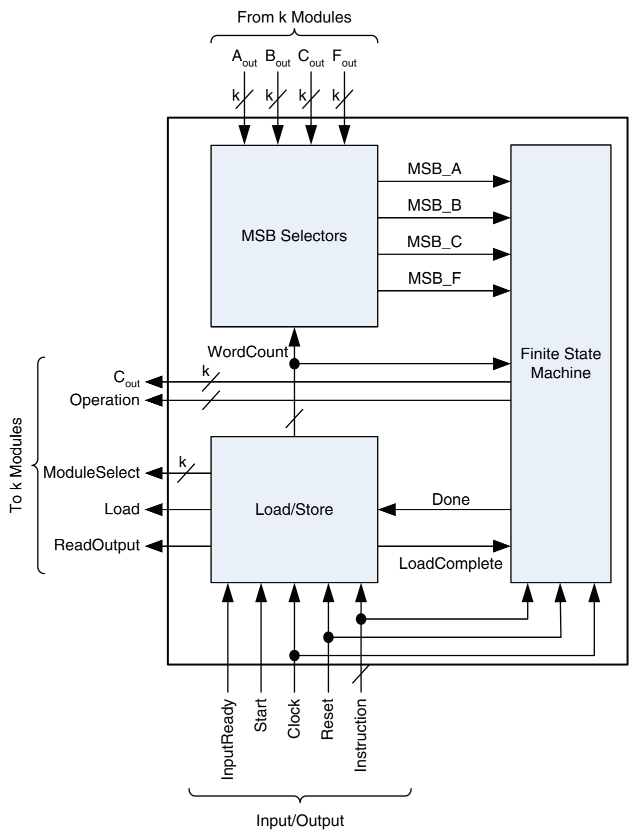

Fig. 2. Block diagram of Controller in the basic architecture.

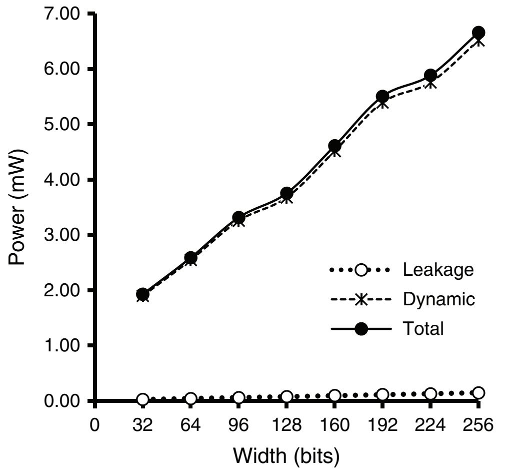

Fig. 3. The relation between power consumption and operand size.

4. PROPOSED MODULAR PARALLEL ARCHITECTURE

In our previous work,[17] the power consumption was reduced by making relation between selected operand size and power consumption. However, that system could not deal with new requests even if there are free unutilized modules which can perform new tasks. In other words, it did not efficiently use the available hardware resources. Thus motivated, the goal of this paper is to present a parallel architecture that increases the overall performance and throughput of the system by granting all requests as long as there are idle modules which can handle new requests. There are two scenarios for making the system operate in parallel. One solution is handling multiple same operations in parallel, i.e., handling multiple multiplication operations. Since high level scalar ECC operations are composed of different operations, this parallelism will not be effective. For instance, a point doubling operation is consisted of multiplication, inversion, and addition operations. This way, if two point doubling requests do not enter at the same time, the second request must wait for the first operation to be completed. Therefore, the system should handle multiple different operations to run in parallel in order to use the resources efficiently. In our developed parallel architecture, the system handles different operations with different operand sizes concurrently. For instance, several multiplication and inversion operations with different key sizes could be simultaneously handled.

The block diagram of the parallel architecture is depicted in Figure 4. Comparing Figures 1 and 4, the parallel architecture has module separators between consecutive modules. These module separators are used to dynamically partition the modules where each partition is responsible for one task. The module separator is a simple multiplexer which selects either zero or previous module's output as the input of the next module. This partitioning is online and its control signals come from Module-Assigner that will be discussed later.

The Controller unit shown in Figure 5 generates appropriate signals to coordinate operations performed in the system. This Controller has three sub-blocks:
(i) Module Assigner,
(ii) Task Controllers, and
(iii) Correlator which will be discussed in the following sub sections.

4.1. Module-Assigner

Module-Assigner connects the system to outside environment. It receives requests from outside and replies to them. Each new request states how many modules it requires (*word-count*) and also the instruction type that must be performed such as multiplication, division, or addition/subtraction. In response, the Module-Assigner will check
(i) if there is a free Task-Controller to take care of this particular task and

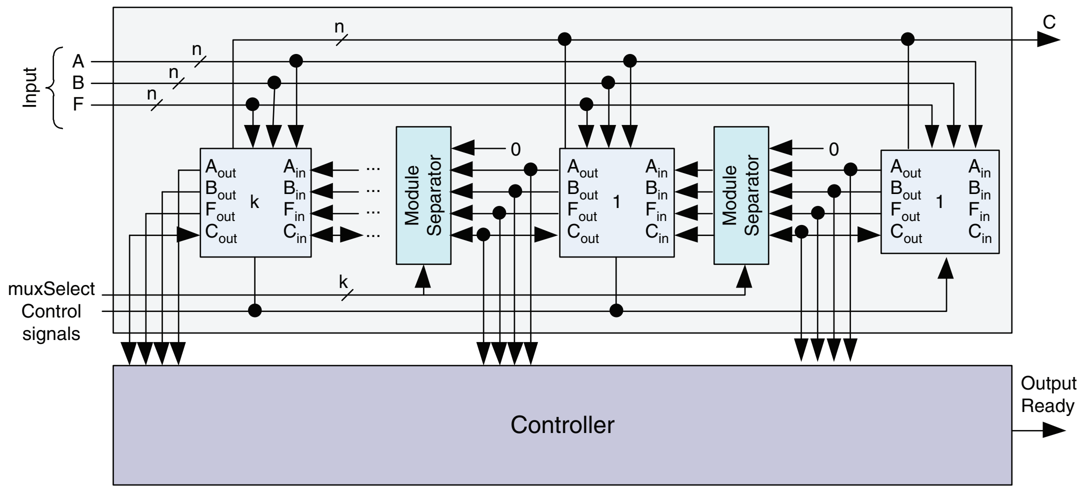

Fig. 4. Block diagram of the parallel multiplier architecture.

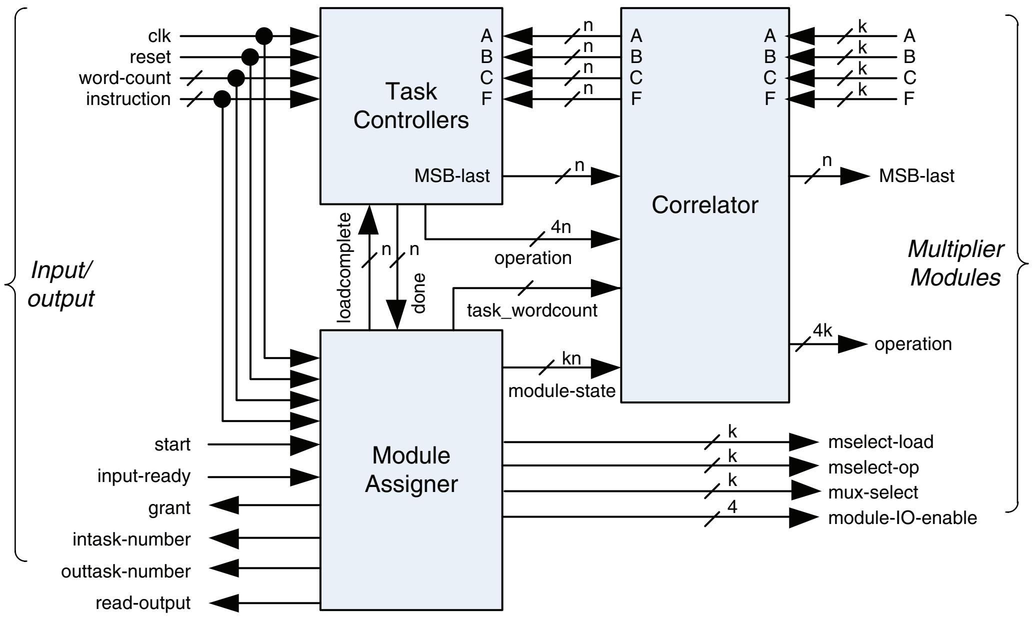

Fig. 5. Block diagram of Controller in the parallel architecture.

(ii) if there are enough free consecutive modules to handle this task.

If one of these conditions fails, then Module-Assigner will reply to the requesting part that it cannot accept this task by putting zero on *grant* signal. Otherwise, the *grant* signal will be one.

Each Task-Controller has a busy tag which is used for this purpose. To check the second condition, the Module-Assigner starts from the first module and check if it can find *word-count* free modules in a row. For example, if the task requires three modules, it will start with checking modules 0, 1 and 2. If module 0 is taken, then it will check modules 1, 2 and 3. This process continues until it finds three consecutive free modules and after that it gives a *task-number* to that particular task in order to make a distinction between tasks. Otherwise, it refuses the new task. This search procedure is implemented by a look up table. The size and complexity of this lookup table is proportional to the number of modules in the system.

After accepting the new request, the Module-Assigner will follow these steps:
(1) The Module-Assigner sets the corresponding modules' I/O signals to load the inputs (A, B and F). The I/O signals define the type of I/O operation based on instruction type. It also triggers modules' *mselect-load* signals one after another to enable them to execute the I/O operation.
(2) After the load phase, the Module-Assigner will start the following actions concurrently.

(a) It will inform the corresponding Task-Controller to start its work by triggering its *load-complete* signal.

(b) It enables all dedicated modules for the computation phase by triggering their *mselect-op* signals.

(c) The Module-Assigner also generates *mux-select* signals which are used for separating this task's modules from other tasks' modules.

(d) It triggers *module-state* and *task-wordcount* signals to correlate Task-Controllers with their assigned modules. Since Module-Assigner is the only part who knows the relation among tasks, modules, and Task-Controllers, it generates these signals dynamically to make this correlation.

(3) When the result of the task at hand becomes ready, the corresponding Task-Controller raises its *done-signal*. Then, Module-Assigner puts the task number on *outtask-number* line to notify the outside parts that the results on the output bus belong to this specific task number. Similar to the loading phase, the modules will send their results to the output bus one by one.
(4) Eventually, the corresponding Task-Controller and all assigned modules will become free.

Furthermore, suppose that task A arrives while task B is ready to send its results. Module-Assigner gives the priority to task B and rejects task A. This way, there are more free modules for new tasks and more importantly, it prevents the deadlock.

4.2. Task-Controllers

The block diagram of Task-Controllers part is shown in Figure 6. This part has t independent units where each unit is the Finite-State-Machine in the basic architecture shown in Figure 2. Each of these units generates the appropriate

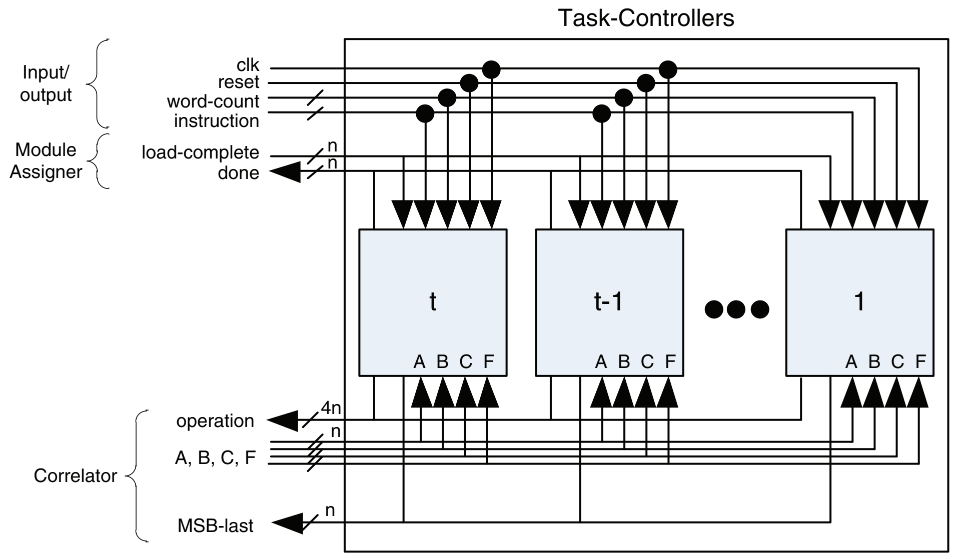

Fig. 6. Block diagram of Task-Controllers in the parallel architecture.

signals to coordinate operations performed inside modules. These units are connected to Module-Assigner and Correlator. The Module-Assigner informs each unit when it is its turn to start computation by *load-complete* signal. At the end of computation of each task, the corresponding unit will inform it to Module-Assigner by raising its *done-signal*.

4.3. Correlator

The Correlator creates a channel between Task-Controller units and their corresponding modules as shown in Figure 7. The Correlator sends the generated op-codes by Task-Controller to modules and in the opposite side, it sends the MSB of operands inside modules to their respective Task-Controller. Per each Task-Controller unit, there is an MSB-Selector circuit inside Correlator. These MSB-Selectors are exactly as the MSB-Selector circuit in the basic architecture. The appropriate MBSs selected by these circuits will be forwarded to the corresponding Task-Controller units.

In the basic architecture,[17] there was only one Finite-State-Machine and one MSB-Selector circuit. In the new parallel architecture, there are many of these units. Besides, the Load/Store module in the basic design is replaced by a more complex Module-Assigner unit, which handles the task assignment in addition to controlling the load and store operations.

Figure 8 demonstrates a sample operation of the proposed architecture. Let us assume a 256-bit architecture, composed of eight 32-bit word modules. In this Figure, the thick-border means that the module is busy with input/output operation. The gray and black rectangles mean that the module is busy with multiplication or inversion operation respectively. Whenever the modules are presented in multiple lines, it means that there are parallel operations in process. Initially, the system is in power save mode, only the Controller is active and all the modules are turned off using *power-gating* signals. Now suppose that a 96-bit multiplication request arrives and accordingly modules 1, 2, and 3 will be activated and will load the input data in cycle 2. In the next cycle, all the input data are read into modules' registers and they will perform the multiplication operation. In cycle 4, a 128-bit inversion request arrives which requires four modules. At this cycle, modules 4, 5, 6 and 7 will load the corresponding input data where modules 1, 2 and 3 are busy with multiplication operation. The modules are presented in separate lines in cycle 4 to indicate the parallel behavior of the system. At cycle 5, modules 1, 2 and 3 are still busy with multiplication operation and modules 4, 5, 6 and 7 are busy with inversion operation. In the next cycle, the multiplication process computation is over and corresponding modules will send their result. Eventually in cycle 8, the inversion process is over and modules

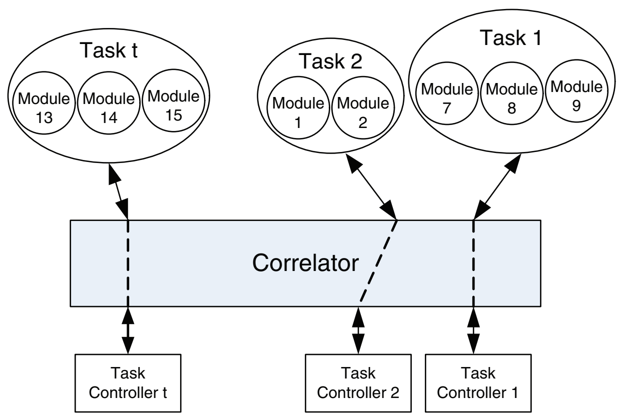

Fig. 7. The relation between the Correlator, Tasks and Task-Controllers.

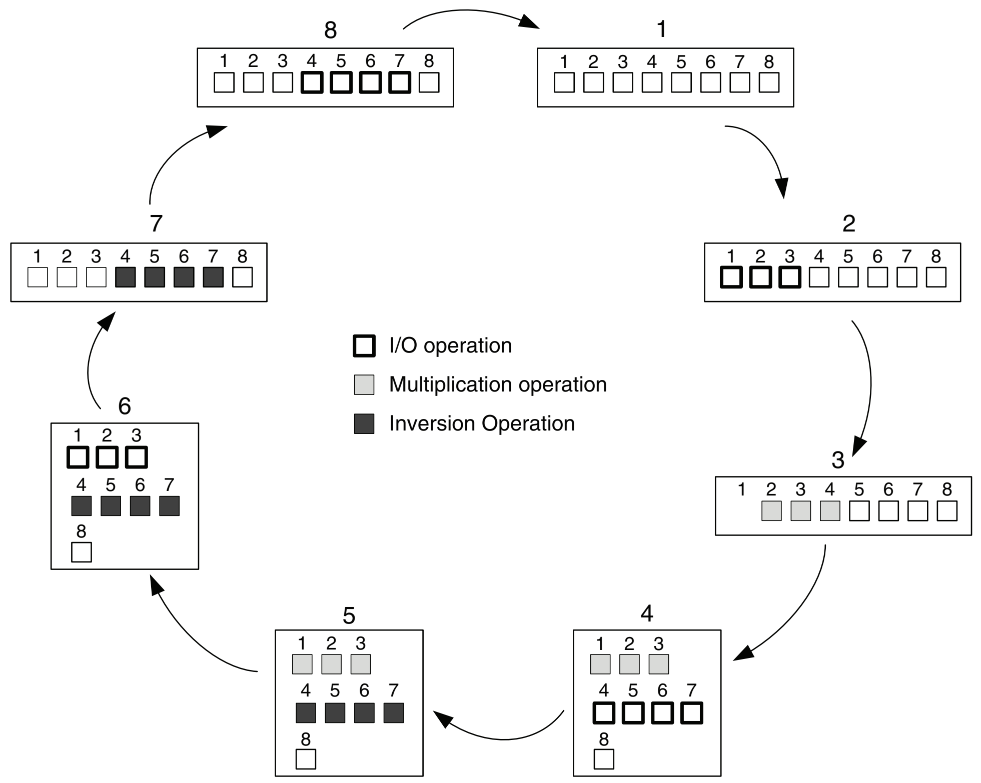

Fig. 8. An example of the operation of proposed parallel architecture.

4, 5, 6 and 7 will send their results as well and afterwards, the processor enters idle state again.

5. EXPERIMENTAL RESULTS

Based on the proposed architectures, a 1024-bit multiplier is designed in three different modes:

- *16–64 Mode: 16 modules each one is 64-bit wide.*
- *32–32 Mode: 32 modules each one is 32-bit wide.*
- *64–16 Mode: 64 modules each one is 16-bit wide.*

The designs are synthesized by Synopsys Design Compiler in the 45 nm technology mode and the FreePDK45 Process Design Kit (PDK). The clock frequency is set to 1 GHz. The operating conditions are set to typical, the supply voltage is fixed at 1.1 V, and the temperature is set to 27 °C. For place and route, the Cadence SOC Encounter tool in 45 nm technology mode is used. Post-layout analysis is used for power and area estimation. Static timing verification method is used for evaluating the design for set up and holds time violations. For timing estimation, a software tool is also developed for generating random data vectors in the test bench codes and these test bench codes are simulated by ModelSim tool.

5.1. Area

Table III summarizes the reported area figures from post-layout die area for 1024-bit implementation. As Table III presents, the basic design is more efficient in terms of area overhead when the number of modules increases as expected. In the new parallel design, the Module-Assigner part has a lookup table to keep track of the idle modules. The size of the lookup table grows with increasing the number of modules and clearly its area will increase accordingly. In the basic design, the complexity of the controller is much less relevant to the number of modules. Increasing the number of modules, the number of I/O pins also decreases, but, the load input and store output phases will increase as well.

We compare our designs' area in 32–32 mode with three other 1024-bit designs which is presented in Table IV. It must be noted that our architectures use newer technology compared to other designs which has considerable effect in area as well.

Table III. Granularity versus controller area overhead.

	Design mode	#ofI/O pins	Total die area (um^2)	Controller overhead (%)
Basic design	16–64	265	195,000	3.5
	32–32	137	201,000	4
	64–16	73	214,000	4.7
Parallel architecture	16–64	275	211,000	11
	32–32	148	230,000	17
	64–16	85	282,000	25

Table IV. Area comparison with other works.

Design	Area (mm^2)	Comments
[9]	4.608 mm^2	0.25 um technology
[16]	2.34 mm^2	0.13 um technology
[7]	O (1024)	Analytic result
Basic architecture	0.2	45 nm technology
Parallel architecture	0.23	45 nm technology

5.2. Timing

Each operation is done in four phases, namely, check availability, load operands, computation and store result. The check availability phase does not exist in the basic design since it was designed to handle just one task at a given time. The duration of check-availability phase is fixed and it is always 4 clock cycles regardless of the granularity of the system and the number of controllers. The durations of the load and store phases depend on the number of required modules for that particular operation. The addition operation always needs one clock cycle and the multiplication always needs twice the operand size number of cycles. The operation cycle of the inversion operation depends on the input data bit-pattern. We perform a large number of simulations by changing bit patterns of input vectors to figure out the average number of clock cycles needed to perform an inversion operation. The average number is 5.5 times the operand size. Changing the module-size and keeping the maximum operand size the same, only the durations of load and store phases will change which is negligible. Consequently, the main computation cycle is proportional to the operand width $O(m)$ and is irrespective of the size of module. This discussion is summarized in Table V.

The main improvement in our parallel architecture is the throughput performance of the system. To evaluate our claim, we simulate both systems in 32–32 mode and generate 100,000 point doubling operations where each operation requires limited random number of modules for completion. This is a typical situation for today secure systems as they likely must handle hundreds of ECC operations per second. Point doubling operation is a good candidate for testing the utilization and throughput of the system because it has all basic operations. Each scalar point multiplication is composed of a series of point doubling and additions. Considering the involved basic operations, point doubling is very similar to point addition and therefore, it is a good approximation for scalar point multiplication as well. Each point doubling operation is composed of seven additions, three multiplications and one inversion. Each point doubling operation has random field-size between 160-bit and 256-bit. In 32–32 mode these operations will require between 5 to 8 modules. 160-bit ECC is secure today and most of typical applications use this scheme and 256 bit ECC in secure until 2030.[19] So this way, a wide range of secure applications' data will be injected to the system.

Table V. # of cycles per operation versus operand size.

	Check-availability	Load/store	Inversion	Multiplication	Addition
Basic design	—	RM*	5.5 m**	2 m**	1
Parallel design	4	RM*	5.5 m**	2 m**	1

**m: Operand size. *RM: Number of required modules.

Table VI. Comparison of throughput between basic and parallel designs.

	# of tasks	Completion time
Basic architecture	100,000	1.22 s
Parallel architecture	100,000	0.357 s

Table VI presents the differences between the results of the two architectures. As the results in Table VI shows, the new architecture completes the task almost 3.47 times faster than the basic architecture. Hence, the throughput of parallel architecture is 3.47 times more than throughput of basic design.

Where the change in key size is limited, the system response time and controller area overhead deteriorates by increasing the granularity of the system as Table VII shows. Making the modules smaller, the response time is increasing because the load and store operations take more time. Therefore, it is better to choose granularity low in typical cases where the change in the key size is limited. However, the flexibility of the system in terms of supporting different key sizes will be low as it does not support keys that are not submultiples of module size when the granularity of the system is low. This implies the existence of a trade-off between the flexibility and area, performance similar to other existing parallel systems.

5.3. Power and Energy

Similar to the area evaluation, our designs' power consumption in 32–32 mode are being compared with three

Table VII. Relation between granularity and performance.

Design mode	Response time (cycle)	Controller area overhead (%)
16–64	0.347 s	11
32–32	0.357 s	17
64–16	0.37 s	25

Table VIII. Power comparison with other works.

Design	Power consumption	Comments
[9]	75 mW	Moderate power
[16]	455 mW	High power
[7]	n/a	Analytic result
Basic architecture	23 mW	Low power
Parallel architecture	28 mW	Low power

Table IX. Energy comparison considering concurrent tasks.

Design	Completion time	Power consumption	Energy	Comment
Basic architecture	1.22 s	9.17 mW	11.19 mJ	~25% of modules are active
Parallel architecture	0.357 s	27 mW	9.64 mJ	~95% of modules are active

other 1024-bit designs which is presented in Table VIII. The proposed architecture in Ref. [7] does not report any actual power figures and has reported just analytical results. Our proposed designs offer very low-power compared to two other designs because of (i) implementing power and/or clock gating techniques and (ii) use of simple components.

Although, the power consumption in parallel architecture is more than the basic design, it requires less energy to accomplish tasks when there are several concurrent tasks. To illustrate the difference, assume that there are 100,000 point doubling tasks like what was assumed in timing evaluation. As Table IX presents, the basic design consumes less power than the parallel design because in average only 25% of its modules are active whereas 95% of the modules are active in parallel design. However, the parallel design completes the tasks in much less time and requires 9% less energy compared to basic architecture.

6. CONCLUSIONS

In this paper, a high-performance parallel arithmetic processor architecture for GF(2^m) has been proposed which supports all essential ECC operations. The architecture is modular, supports arbitrary operand sizes and is scalable for very large operand sizes. When a small key size operation is requested, the system deactivates the remaining modules to reduce the power consumption if there is no parallel request to the system. Otherwise, it handles new requests by the remaining unutilized modules in parallel with other modules. The new request to the system could be different operations with different operand sizes that let the system to use its resources very efficiently which considerably increases the overall performance and throughput of the system. Adding other operations such as point addition and point doubling to our architecture is our future work in our ultimate goal to design a complete ECC processor.

References

1. R. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21, 120 (**1978**).
2. V. Miller, Uses of elliptic curves in cryptography, *Proceedings of the Advances in Cryptography* (**1986**), Vol. 218, pp. 417–426.
3. N. Kobilitz, Elliptic curve cryptosystems. *Mathematics of Computation* 48, 203 (**1987**).
4. D. Hankerson, A. J. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer-Verlag (**2004**), pp. 259–260.
5. P. K. Meher, On Efficient implementation of accumulation in finite field over GF(2^m) and its applications. *IEEE Transactions on VLSI Systems* 17, 541 (**2009**).
6. E. Savaş and Ç. K. Koç, Finite field arithmetic for cryptography. *IEEE Circuits and Systems Magazine* 40 (**2010**).
7. M. Hasan and M. Ebtedaei, Efficient architectures for computations over variable dimensional Galois Field. *IEEE Transactions on Circuits Systems—I: Fundamental Theory and Applications* 45, 1205 (**1998**).
8. N. Gura et al., An end-to-end systems approach to elliptic curve cryptography. *Cryptographic Hardware and Embedded Systems* 349 (**2002**).
9. J. Goodman and A. Chandrakasan, An energy-Efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-State Circuits* 36, 1808 (**2001**).
10. S. Kumar, T. Wollinger, and C. Paar, Optimum digit serial GF(2^m) multipliers for curve-based cryptography. *IEEE Transactions on Computers* 55, 1306 (**2006**).
11. P. Kitsos, G. Theodoridis, and O. Koufopavlou, An efficient reconfigurable, multiplier architecture for Galois Field GF(2^m). *Elsevier Microelectronics Journal* 34, 975 (**2003**).
12. C. Chiou, C. Lee, and J. Lin, Unified dual-field multiplier in GF(p) and GF(2^k). *Institute of Engineering and Technology (IET) Information Security* 3, 45 (**2009**).
13. A. H. Namin, H. Wu, and M. Ahmadi, Comb architectures for finite field multiplication in $IF_2{}^m$. *IEEE Transactions on Computers* 56, 909 (**2007**).
14. A. E. Cohen and K. K. Parhi, GPU Accelerated Elliptic Curve Cryptography in GF(2^m), *Proceedings of IEEE International Midwest Symposium on Circuits and Systems* (**2010**), pp. 57–60.
15. M. Koschuch, J. Lechner, A. Weitzer et al., Hardware/software co-design of elliptic curve cryptography on an 8051 microcontroller, *Proceeding of Workshop of Cryptographic Hardware and Embedded System* (**2006**), pp. 430–444.
16. J. Chen and M. Shieh, A high-performance unified-field reconfigurable cryptographic processor. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems* 18, 1145 (**2010**).
17. M. I. Faisal, Z. Jeddi, E. Amini, and M. Bayoumi, A power-aware selectable operand-size modular multiplication architecture for GF(2^m). *J. Low Power Electron.* 7, 314 (**2011**).
18. H. Brunner, A. Curiger, and M. Hofstetter, On computing multiplicative inverses in GF(2^m). *IEEE Transactions on Computers* 42, 1010 (**1993**).
19. V. Gupta, D. Stebila, and S. Chang, Integrating elliptic curve cryptography (ECC) into the web's security infrastructure, *Proceedings of International World Wide Web Conference on Alternate Track Papers and Posters* (**2004**), pp. 402–403.

Esmaeil Amini

Esmaeil Amini *received his B.S. degree from Sharif University of Technology and M.S. degree from Amirkabir University of Technology both in Computer Engineering. He is currently a Ph.D. student at the Center for Advanced Computer Studies (CACS) at the University of Louisiana at Lafayette. His research interests include computer architecture, security and low power design.*

Zahra Jeddi

Zahra Jeddi *received her B.S. degree in electrical engineering from Iran University of Science and Technology and her M.S. degree in Computer Engineering from Amirkabir University of Technology. She is currently a Ph.D. student at the Center for Advanced Computer Studies (CACS) at the University of Louisiana at Lafayette. Her research interests include low power design, computer architecture and security.*

Ahmed Khattab

Ahmed Khattab *received his B.Sc. (honors) and M.Sc. in Electronics and Communications Engineering from Cairo University, Egypt, in 2002 and 2004, respectively. He received the Master of Electrical Engineering degree from Rice University, and his Ph.D. in Computer Engineering from the University of Louisiana at Lafayette, USA, in 2009 and 2011, respectively. He is currently an Assistant Professor at the Electronics and Electrical Communications Engineering Department in Cairo University. He was a Research Associate at the Center for Advanced Computer Studies (CACS) at the University of Louisiana at Lafayette. His research interests are the design and implementation of cross layer PHY-MAC protocols and radio resource management for high performance wireless networks. He has multiple coauthored and single-authored books, patents, and journal and peer-reviewed conference publications. He won the best student paper award of the ULL IEEE Computer Society in 2010 and was finalist in the IEEE ICCCN 2008 best paper contest.*

Magdy Bayoumi

Magdy Bayoumi *received the B.Sc. and M.Sc. degrees in electrical engineering from Cairo University, Egypt. He received the M.Sc. degree in computer engineering from Washington University, St. Louis, MO, and the Ph.D. degree in electrical engineering from the University of Windsor, ON, Canada. He is currently Director of the Center for Advanced Computer Studies (CACS) and Department Head of the Computer Science Department, University of Louisiana, Lafayette. He is also the Edmiston Professor of Computer Engineering and Lamson Professor of Computer Science at the Center for Advanced Computer Studies, University of Louisiana at Lafayette, where he has been a faculty member since 1985. He is editor or coeditor of three books in the area of VLSI Signal Processing. His research interests include VLSI design methods and architectures, low-power circuits, and systems, digital signal processing architectures, parallel algorithm design, computer arithmetic, image and video signal processing neural networks, and wideband network architectures. Dr. Bayoumi was Vice President for the technical activities of the IEEE Circuits and Systems Society. Currently, he is Chairman of the Technical Committee (TC) on Circuits and Systems for Communication and the TC on Signal Processing Design and Implementation. He was a founding member and Chairman of the VLSI Systems and Applications Technical Committee. He is also a member of the Neural Network and the Multimedia Technology Technical Committees. He has been on the technical program committee for ISCAS for several years, and he was the publication chair for ISCAS'99. He was the General Chairman of the 1994 MWSCAS and is a member of the Steering Committee of this symposium. He was an Associate Editor of the IEEE CIRCUITS AND DEVICES MAGAZINE, IEEE TRANSACTION ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON NEURAL NETWORKS, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II. He was the cochairman of the Workshop on Computer Architecture for Machine Perception in 1993 and is a member of the Steering Committee of this workshop. He was general chairman for the 8th Great Lake Symposium on VLSI in 1998, and general chairman of the 2000 Workshop on Signal Processing Design and Implementation. He is an Associate Editor of the VLSI Journal INTEGRATION, and the Journal of VLSI Signal Processing Systems. He is a regional editor for the VLSI Design Journal and on the Advisory Board of the Journal on Microelectronics Systems Integration. He served on the Distinguished Visitors Program for the IEEE Computer Society from 1991 to 1994 and is currently on the Distinguished Lecture program of the Circuits and Systems Society. He won the UL Lafayette 1988 Researcher of the Year Award and the 1993 Distinguished Professor Award at UL Lafayette.*