A Novel Unified Architecture for Public-Key Cryptography

A. Cilardo, A. Mazzeo, N. Mazzocca, L. Romano

A Novel Unified Architecture for Public-Key Cryptography

A. Cilardo, A. Mazzeo, N. Mazzocca, L. Romano
Università degli Studi di Napoli Federico II
Dipartimento di Informatica e Sistemistica
via Claudio 20, 80125 Napoli, Italy
{acilardo, mazzeo, n.mazzocca, lrom}@unina.it

Abstract

In this paper we propose a fully-parallel, bit-sliced unified architecture designed to perform modular multipication/exponentiation and $GF(2^M)$ multiplication as the core operations of RSA and EC cryptography.

The architecture uses radix-2 Montgomery technique for modular arithmetic, and a radix-4 MSD-first approach for $GF(2^M)$ multiplication. To the best of our knowledge, it is the first unified proposal based on such a hybrid approach. The architecture structure is bit-sliced and is highly regular, modular, and scalable, as virtually any datapath length can be obtained at a linear cost in terms of hardware resources and no costs in terms of critical path. Our proposal outperforms all similar unified architectures found in the technical literature in terms of clock count and critical path.

The architecture has been implemented on a Field-Programmable Gate Array (FPGA) device. A highly compact and efficient design was obtained taking advantage of the architectural characteristics.[1]

1. Introduction

The ever-increasing demand for security, along with several crucial requirements like performance and tamper-resistance, has caused in the recent years an increasing deployment of hardware devices to support security services, such as confidentiality, authentication, integrity and non-repudiation.

In particular, asymmetric, or public-key, cryptography plays an important role in all modern security-aware applications, since it has the fundamental property of enabling digital signature and encryption without any sharing of secret information between the interested parties.

Among the existing techniques the Rivest-Shamir-Adleman (RSA) algorithm [12] is at present the most widely adopted public-key cryptography algorithm. However, Elliptic Curve (EC) cryptography [10] has been gaining more and more popularity and importance during the last years, and has been embedded in many security protocols and standards [7].

While hardware-based implementations of public-key algorithms are very computationally efficient and generally offer high levels of tamper-resistance, they are typically inflexible and often yield specific, parameter-dependent solutions. During the last years, many research activities have been attempting to define flexible hardware architectures, that are suitable to perform the fundamental public-key cryptographic operations independent of the parameter values (e.g., the value of the modulus used for modular arithmetic), the operand length, and even the specific class of mathematical operations to compute. In particular, a few proposals have been recently made [14, 15, 6, 13] to provide unified hardware architectures for executing both integer modular arithmetic and $GF(2^M)$ operations, thus covering all the underlying arithmetics supporting RSA cryptography and EC cryptography.

Almost all existing architectures make use of Montgomery multiplication [11]. The Montgomery multiplication algorithm was initially proposed as an efficient method to perform modular multiplication in prime fields. It was then shown in [9] that Montgomery multiplication can also be used in $GF(2^M)$, when elements are represented in the standard basis and the irreducible polynomial for the field is taken arbitrarily.

However, it should be noted that, apart from the possibility of designing unified architectures, no real benefit comes from the Montgomery extension to $GF(2^M)$ arithmetic, since in this case all operations are already intrinsically carry-less and, in principle, the Montgomery technique does not affect the operation speed. On the other hand, there is still the additional pre- and post-processing over-

1 This work was supported in part by the Italian National Research Council (CNR), by Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR), and by Regione Campania, within the framework of following projects: SP1 Sicurezza dei documenti elettronici, Gestione in sicurezza dei flussi documentali associati ad applicazioni di commercio elettronico, Centri Regionali di Competenza ICT, and Telemedicina.

head required by the Montgomery transformation, that turns out to be the price to pay for obtaining a unified architecture.

In this paper we propose a fully-parallel, bit-sliced unified architecture designed to perform modular multipication/exponentiation and $GF(2^M)$ multiplication as the core operations of RSA and EC cryptography. The heart of the architecture is the *dual-mode adder*, a hardware block proposed in [3, 4, 5] that supports both carry-save and carry-propagate addition. In this work the dual-mode adder is extended to become a *dual-field adder* relying on its intrinsic capability of alternating carry and carry-less operation.

The architecture uses radix-2 Montgomery technique for modular arithmetic, and a radix-4 most-significant-digit (MSD) first approach for $GF(2^M)$ multiplication. To the best of our knowledge, it is the first unified proposal based on such a hybrid approach. The architecture structure is bit-sliced and is highly regular, modular, and scalable, as virtually any datapath length can be obtained at a linear cost in terms of hardware resources and no costs in terms of critical path. Our solution outperforms all similar unified proposals found in the technical literature in terms of clock count and architectural critical path.

The architecture has been implemented using Field-Programmable Gate Array (FPGA) as the target technology. A highly compact and efficient design was obtained taking advantage of the architectural characteristics.

The rest of the paper is organized as follows. Section 2 presents the proposed algorithm for computing the RSA modular exponentiation and $GF(2^M)$ multiplication. Section 3 describes the architecture implementing the RSA modular exponentiation/$GF(2^M)$ multiplication. Section 4 provides details about physical implementation of the proposed architecture and presents the results achieved. Section 5 gives some comparisons against similar architectures and concludes the paper with some final remarks.

2. Algorithms

Algorithm for RSA exponentiation. For implementation of modular multiplication we used a modified form of the standard radix-2 Montgomery algorithm [11]. Our version of the algorithm (referred to as $MonProd$) is based on some optimizations proposed in [16, 3, 4, 5] and uses an intermediate carry save representation of operands to avoid carry propagation in the loop body.

Algorithm 1 - *Carry-Save Montgomery Product MonProd(A,B) radix-2.*

Given $N = \sum_{i=0}^{K-1} N_i \cdot 2^i$, $A = \sum_{i=0}^{K+4} A_i \cdot 2^i < 2N$, $B < 2N$, where $N_0 = 1$, A_i, $N_i \in \{0,1\}$, $A_{K+1} \ldots A_{K+4} = 0$, computes a number falling in [0, $2N$[which is modulo N congruent with $(A \cdot B \cdot 2^{-(K+2)}) \bmod N$

1. $S := 0, \quad C := 0$
2. $V := 0$
3. for $h := 0$ to $K+4$ do
4. $\quad q := q(S_2, S_1, C_1, C_0, N_1)$
5. $\quad V_{NEXT} := A_h \cdot 4B + q \cdot N$
6. $\quad S := S/2, \quad (S, C) := S + C + V$
7. $\quad V := V_{NEXT}$
8. end for
9. return $S/2 + C$

where $(S, C) := S + C + V$ denotes a carry-save addition, i.e. given $S = \sum_{i=0}^{K+3} S_i \cdot 2^i$, $C = \sum_{i=0}^{K+3} C_i \cdot 2^i$, $V = \sum_{i=0}^{K+3} V_i \cdot 2^i$, the updated values of S and C are obtained as $S_i := S_i \oplus C_i \oplus V_i$ and $C_i := S_i C_i + S_i V_i + C_i V_i$.

During the hth iteration, q represents the least significant bit of the partial product U ($U = S/2 + C$) computed during the $(h+1)$th iteration. The least significant bits of the current U are needed to choose which value of V to add during the next operation in the loop of the Montgomery's algorithm. These bits can be easily derived even though numbers are in carry-save form. It is worth emphasizing that in our $MonProd$ algorithm step 6, which performs the current operation, has no dependency upon steps 4-5, which decide on the subsequent operation. Thus, control operations and data operations can be executed concurrently even if the addition is implemented on a fully parallel structure. The quantities $4B$ and $4B + N$ can be computed and stored before executing the $MonProd$ algorithm, and added during the $MonProd$ loop according to the values of A_h and q (step 5).

In the following we report the exponentiation algorithm for computing $X^E \bmod N$ known as Right-To-Left binary method [8], modified in order to take advantage of Montgomery product as defined in Algorithm 1.

Algorithm 2 - *Right-To-Left Modular Exponentiation using Montgomery Product.*

Given $N = \sum_{i=0}^{K-1} N_i \cdot 2^i$, $X < N$, $E = \sum_{i=0}^{H-1} E_i \cdot 2^i < N$, and $W = (2^{K+2})^2 \bmod N$. computes $P = X^E \bmod N$.

1. $Z := MonProd(X, W)$
2. $P := MonProd(1, W)$
3. for $j := 0$ to $H-1$ do
4. $\quad Z := MonProd(Z, Z)$
5. $\quad$ if $(E_j = 1)$ then $P := MonProd(P, Z)$
6. end for
7. return $MonProd(P, 1)$

where $MonProd(A, B)$ is a number falling in [0, $2N$[which is modulo N congruent with $(A \cdot B \cdot 2^{-(K+2)}) \bmod N$.

For a given modulus value, the factor $W = (2^{K+2})^2 \bmod N$ remains unchanged. It is thus possible to use a precomputed value for such a factor and reduce residue calculation to a $MonProd$. It is worth noting that, due to the absence of dependencies between instructions 4 and 5, these can be executed in parallel.

For further details about our versions of modular multi-

plication and modular exponentiation algorithms, please refer to [3, 4, 5].

Algorithm for $GF(2^M)$ multiplication. We deal with elliptic curves (ECs) defined by the nonsupersingular Weierstraß equation over binary fields $GF(2^M)$. Due to the lack of space, we do not give details about EC cryptography. The interested reader is referred to, for example, [10, 7]. Suffice it to say here that all operations necessary for EC cryptosystems can always be reduced to sequences of addition and multiplication operations in the underlying finite field, namely $GF(2^M)$ in our case, that thus play a fundamental role for implementation of EC cryptography. In turn, the representation of the field elements is crucial for the efficiency of all field operations. For our architecture, we use the *polynomial* or *standard* representation. A field $GF(2^M)$ is isomorphic to $GF(2)[x]/N(x)$, where $N(x) = x^M + \sum_{i=0}^{M-1} N_i \cdot x^i$ is a monic irreducible polynomial of degree M with coefficients $N_i \in \{0, 1\}$. Here each residue class is represented by the polynomial of least degree in its class. A standard basis representation uses the basis defined by the set of elements $\{0, \alpha, \alpha^2, \ldots, \alpha^{M-1}\}$ where α is a root of the irreducible polynomial $N(x)$. In this basis, field elements are represented as polynomials in α of degree less than M with coefficients 0 or 1. Addition/subtraction between elements is simply performed by bitwise XORing their coefficients, and multiplication is performed, conceptually, by performing the usual polynomial multiplication, followed by reduction modulo $N(x)$.

Polynomial representation is particularly efficient for hardware implementation, where field elements are represented as bit strings. Implementation of field addition is straightforward. Multiplication, however, is quite more complex and turns out to be the critical operation for all practical EC implementations.

In the following, we report the algorithm we adopted for our architecture. It is based on a radix-4 representation of field elements and uses a most-significant-digit first approach.

Algorithm 3 - *Most-Significant-Digit first radix-4 algorithm for $GF(2^M)$ multiplication.*
Given $N(x) = \sum_{i=0}^{M} N_i \cdot x^i$, $A(x) = \sum_{i=0}^{M-1} A_i \cdot x^i$, $B(x) = \sum_{i=0}^{M-1} B_i \cdot x^i$ where $N_i, A_i, B_i \in \{0, 1\}$, $N_M = 1$, computes $A(x) \cdot B(x) \bmod N(x)$

1. $X := 0$
2. $V := 0$
3. for $h := \lceil M/2 \rceil$ downto 0 do
4. $\quad g := g(X_{M-1} \ldots X_{M-4}, N_{M-1} \ldots N_{M-3},$
- $\qquad\qquad B_{M-1} \ldots B_{M-3}, A_{2h+1} \ldots A_{2h-2})$
5. $\quad V_{NEXT} := (A_{2h+1} \cdot x + A_{2h}) \cdot B \oplus g \cdot N$
6. $\quad X := 4X \oplus V$
7. $\quad V := V_{NEXT}$
8. end for
9. return X

where $A_{2 \cdot \lceil M/2 \rceil + 1} = \ldots = A_M = A_{-1} = A_{-2} = 0$, $\oplus$ denotes a bit-wise XOR, and $g \in \{0, 1, x, 1 + x\}$.

At each step, a multiple of the irreducible polynomial N is added to the partial result X for zeroing its most significant digit. Note that there is not carry propagation here, so the decision about the next operation to perform can be taken based only on the most significant bits of the operands (by means of the function g). This behaviour is somewhat dual to the previously presented Montgomery algorithm. As for Algorithm 1, step 6, which performs the current operation, has no dependency upon steps 4-5, which decide on the subsequent operation. Thus, control operations and data operations can be executed concurrently.

3. Unified Architecture

In this section we describe the fully-parallel, bit-sliced unified architecture designed to perform modular multipication/exponentiation and $GF(2^M)$ multiplication as the core operations of RSA and EC cryptography.

The architecture extends a previously devised solution implementing RSA modular exponentiation [3, 4, 5], that supported both carry-save and carry-propagate addition within the same hardware block, the *dual-mode adder*. This work shows how such a block can be extended in a straightforward way to become a *dual-field adder*, that is a block supporting both modular multiplication and $GF(2^M)$ multiplication. This extension is accomplished in a natural way and with almost no additional hardware due to the capability of alternating carry and carry-less operation intrinsic to the originally conceived dual-mode adder [3, 4, 5].

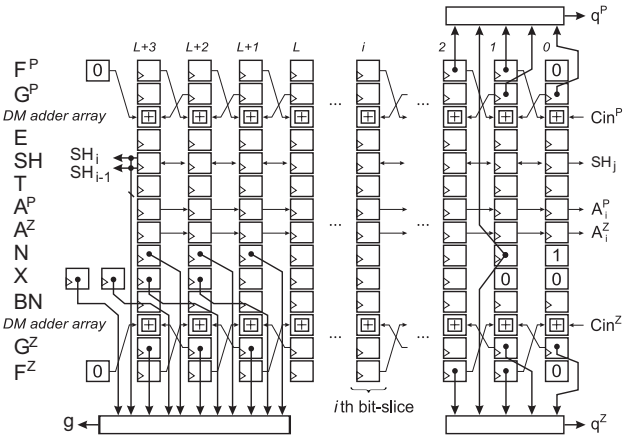

Figure 1. The overall unified architecture.

The overall architecture is depicted in Figure 1, while the bit-slice structure is shown in Figure 2

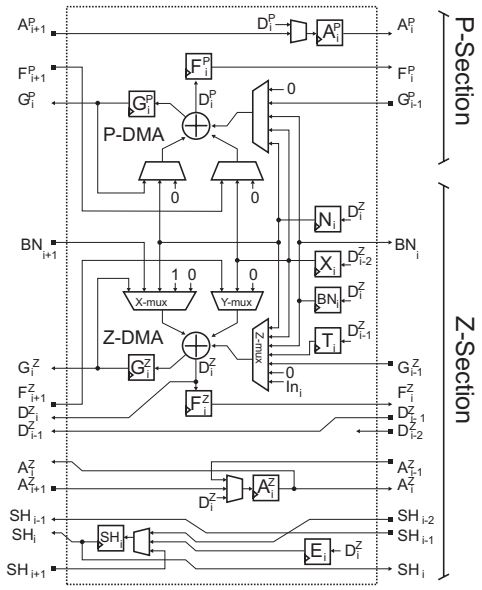

Figure 2. The structure of the ith bit-slice.

The architecture works on up to L-bit moduli for RSA cryptography and up to $L + 3$-bit operands for $GF(2^M)$ multiplication. Its structure is comprised of $L + 4$ bit-slices due to the maximum size of intermediate results during RSA operation.

The architecture is designed to perform in parallel the two coupled multiplications of Algorithm 2 based on the radix-2 Montgomery technique, resorting to the carry-save representation for the main loop of the multiplication process and the carry-propagate addition for the final conversion into the non-redundant form. In addition, the same architecture can compute a $GF(2^M)$ multiplication using a radix-4 most-significant-digit (MSD) first approach, that does not require any transformation on operands while being as efficient as the Montgomery multiplication.

The Dual-Mode (DM) Adder in the upper portion of Figure 1 and Figure 2, along with the associate registers F^p, G^p, and A^p are altogether referred to as P-section in the following, while the remaining part of the architecture is referred to as Z-section. The P-section is used to speed up modular exponentiation execution. As explained in [3, 4], this technique halves the execution time for RSA modular exponentiation, while increasing the hardware costs by a factor less than 2. Note that it would be possible to suppress the P-section and obtain a more compact modular/$GF(2^M)$ multiplier not specifically optimized for RSA exponentiation.

The names of the registers in Figure 1 as well as flip-flops in Figure 2, are kept as general as possible since different registers are used for holding different quantities during modular arithemtic/$GF(2^M)$ operations. In the following subsections we particularize the operation of each component to show how the unified architecture can be used to compute both RSA modualar exponentiation and $GF(2^M)$ multiplication.

RSA Operation. During RSA operation the architecture performs concurrently the two coupled multiplications of Algorithm 2. The superscripts Z and P in the figures indicate that the corresponding signals and components are specifically involved in Z and P product computation of Algorithm 2.

Flip-flops N, X, and BN are used to hold the shared quantities N, $4B$, and $4B + N$, respectively, that are accessed by both the Z and P sections. Sum-Carry pairs are stored into the (F, G) pairs respectively, for each of the two sections. The exponent and the left-operands of Z and P multiplications are handled by registers SH, A^Z, and A^P, respectively, all used as single-bit right shift-registers. The quantity W used in Algorithm 2 is stored into register T. All operands are right-aligned when fed into the architecture.

Each of the two DM adder arrays in Figure 1 uses the register G (G^Z or G^P) to hold the carry part of the carry-save pair during the loop of the $MonProd$ algorithm, while the flip-flops of register G are individually used for carry propagation during a $(K + 4)$-bit carry-propagate addition. More precisely, within the ith bit-slice, each of the two DM adders can add the ith bit of $F/2$ (i.e. F_{i+1}), G_i, and one of $\{0, N_i, (4B)_i, (4B + N)_i\}$ in carry-save mode for executing $F := F/2$, $(F, G) := F+G+V$ in the $MonProd$ loop body taking one clock cycle altogether. In carry-propagate mode the DM adder can add the bits F_{i+1}, G_i together with the carry coming from the $(i - 1)$th bit-slice for executing the $MonProd$ post-processing addition $F/2 + G$, taking $K + 4$ clock cycles altogether. The Z DM adder can also add the bits $(4B)_i$ (held by flip-flop X_i) and N_i in carry-propagate mode for executing the $MonProd$ pre-processing addition $4B + N$ (stored into register BN). It is worth noting that the two carry-propagate additions $F/2+G$ for the P and the Z $MonProd$ (performed concurrently by both the P section and the Z section after the $MonProd$ loop) and the carry-propagate addition $4B+N$ (where B is the previous Z product and the addition is performed before starting the following $MonProd$ loop) can overlap during the modular exponentiation process, thus taking $K+5$ clock cycles altogether.

A complete modular exponentiation process takes $(2K + 14)(H + 2) + K + 7$ clock cycles, where $K \leq L$ is the actual bit size of the modulus and H is the bit size of the exponent. The dominant term $2KH$ is due to $K + 5$ carry-save additions and the overlapped $(K + 4)$-bit carry-propagate additions, to be iterated $H + 2$ times within the modular exponentiation algorithm.

$GF(2^M)$ Operation. During $GF(2^M)$ operation the architecture performs the radix-4, MSD-first multiplication of al-

gorithm 3 within the Z-Section.

Flip-Flops X are used to hold the partial result, while flip-flops N, T, G^Z, and BN are used to hold the quantities N, $2N$, B, and $2B + 2N$, respectively. Thus, multiplexer X-Mux can drive into the adder one of the quantities 0, B, N, or $B + N$, while multiplexer Z-Mux can do the same with quantities 0, $2B$, $2N$, or $2B + 2N$. Therefore, any of the values $\{0, B, N, B+N, B+2N, \ldots, 3B+2N, 2B + 3N, 3B + 3N\}$ can be obtained on the fly and added to the shifted partial result $4X$ through multiplexer Y-Mux. All operations within the main loop of Algorithm 3 are thus accomplished in one clock cycle (note that g at step 5 of Algorithm 3 is a 2-bit number).

The left-operand A of the multiplication is handled by register SH, used here as a two-bit left-shift register. Register F^Z is used for holding temporary data during the initialization phase. The shifts for $F/2$ and $4X$ are wired. Note that all operands are left-aligned when fed into the architecture for $GF(2^M)$ multiplication.

Altogether, a complete $GF(2^M)$ process takes $\lceil M/2 \rceil + 6$ clock cycles, including the initial setup to input operands and compute $2N$ and $2B + 2N$. The factor $\frac{1}{2}$ applied to the dominant term in the above formula derives directly from the radix-4 technique used for the multiplication.

One important thing to note for both RSA and GF operation modes is that a look-ahead technique is adopted to govern the multiplication process. This technique can be recognized in the fact that "control" operations necessary to compute V_{NEXT} in Algorithms 1 and 3 are independent of "data-path" operations necessary to update the partial results (the pair (S, C) for RSA or the value X for GF) during the same cycle. From an architectural viewpoint, this allows the controller to know in advance the least/most (for RSA/GF respectively) significant bits of the partial result and issue the control signals one clock cycle before they actually apply. Control signals have thus to be delayed and, importantly, they can be easily broadcast across the structure by means of flip-flops, which break up the critical path and enable very efficient hardware implementations. Note also that the computational effort necessary to compute the control bits in advance is comparable to that of data-path operations, so the controller/datapath pipeline is well balanced. Another important property of our architecture is that each bit-slice communicates only with its neighbours, that is, all data signals are strictly local. This is an advantageous condition for any hardware implementation.

4. Physical Implementation

The architecture presented in the previous section has been implemented on a Field-Programmable Gate Array (FPGA), namely a Xilinx Virtex-E2000 device, for the case L = 1024, which corresponds to 1028 bit-slices to be physically placed on the chip. In order to exploit the regularity of the architecture and to optimize time and area performance, we firstly synthesized the basic building block of the design, i.e. the bit-slice of Figure 2. The synthesized bit-slice requires 24 LUTs, 12 flip-flops, and 3 tristate buffers and has a critical path of 5.25ns. Then, we implemented the overall design. We resorted to a "serpentine" scheme on the chip to preserve the locality of data signals and make sure each bit-slice is close to the two preceding bit-slices and to the next one. A 84×80-CLB area was used to accommodate the complete architecture on the FPGA device. We left a cross-shaped zone within the data-path area (see Figure 3) to allow the synthesis tool to place the controller. This placement constraint facilitate the place-and-route step, and reduce the net delay due to control signal broadcasting, since the controller is embedded in the data-path. We resorted to a number of additional design techniques, such as the manual replication of flip-flops broadcasting control signals, which ensures each replicated flip-flop is wired to a set of bit-slices close to each other to minimize net delay. For furhter details, refer to [3, 4].

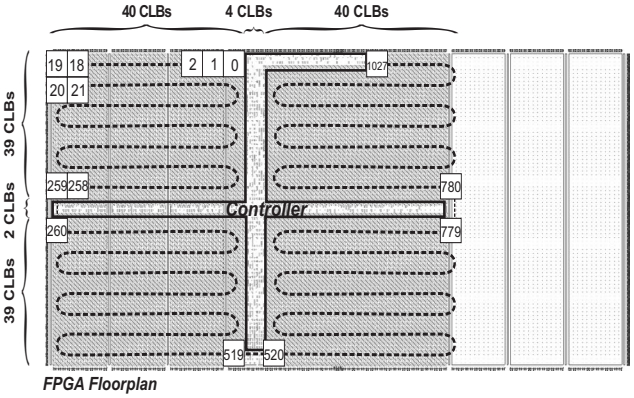

Figure 3. The data-path placement scheme.

Figure 3 shows the whole design implemented on the targeted device after the place-and-route process. Altogether, the design takes 6,709 CLBs (25,633 LUTs, 13,542 flip-flops, and 3084 tristate buffers), i.e. it requires 71% of the target device resources. The minimum clock period is 12.93ns. From the formulas of section 3, a 1024-bit full length exponentiation, i.e. a modular multiplication with an exponent E and a modulus N with a length of 1024 bits, is accomplished in 27.36 ms. A $GF(2^M)$ multiplication with $M = 163$ takes 1.138 μs.

5. Comparison and Conclusions

To the best of our knowledge, there exist no unified architectures fully implementing the RSA modular exponentiation along with $GF(2^M)$ multiplication, thus we compare

our architecture to the fastest FPGA-based implementation of radix-2 Montgomery algorithm [1, 2]. The implementation in [1] takes 40 ms for a 1024-bit modular exponentiation, while our implementation takes 27.36 ms, achieving a 32% reduction. The physical device used in [1] is not the same as ours. This makes an exact comparison difficult in terms of hardware resource requirements and time performance. The architecture of [1] shows a critical path of one 4-bit adder. Our architecture presents a critical path of one single-bit full-adder and some selection logic. Also, it should be noted that, unlike the architecture of [1], our design does not make use of optimized blocks and has considerable device-independent features.

As far as $GF(2^M)$ is concerned, there exist many GF-specific architectures and implementations in the technical literature. Here, we specifically focus on unified architectures proposed in the recent years, that are capable of executing modular and $GF(2^M)$ arithmetic within the same hardware architecture.

While we provide an FPGA-based implementation, all unified architectures considered for comparison report ASIC-based implementations, so it is not possible to give an exact evaluation in terms of area resources and absolute execution time. Therefore, we use the number of clock cycles and an estimation of the critical path in terms of basic hardware blocks, for each of the compared architectures.

In [14], extensive results are provided with respect to the dual-radix (radix-2 for $GF(p)$ and radix-4 for $GF(2^M)$) particularized to the word-length of w=32 and optimized for operands of up to 256 bits. For this architecture, a critical path of one multi-bit full-adder and some selection logic (multiplexers) is obtained, and, taking as a reference a $GF(2^{160})$ multiplication, the clock count is equal to 166. In [6], a MSD-first approach is taken for both $GF(2^M)$ and $GF(p)$. The critical path of this architecture is made of two single-bit full-adders and some selection logic, while the clock count for a 160-bit multiplication is 160 cycles. [13] proposes an arithmetic core block and a unified multiplier architecture built on it. Three different word lengths (i.e. w = 8, 16, 32) are considered. Time complexity is poor compared to the other two architectures as even the multiplier with the shortest word length (w = 8) has seven single-bit full adders and one w-bit carry propagation adder (CPA) in its critical path. For $w = 8$, a 160-bit multiplication takes 882 cycles.

In our architecture, the critical path includes only one single-bit full-adder and some extra logic necessary to select the operands. In addition, the clock count is much lower than the previous architectures: 86 clock cycles for a 160-bit multiplication. We also emphasize that our architecture is the first unified proposal based on a dual-radix, LSb/MSD-first hybrid approach, as it uses radix-2 Montgomery technique for modular arithmetic, and radix-4 MSD-first technique for $GF(2^M)$. This choice is based on the observation that, for $GF(2^M)$ operations, Montgomery technique has basically no benefit while still requiring the domain transformation overhead.

As a future work, we plan to explore how this technique could apply to different, non fully-parallel architectures.

References

[1] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. *Proc. 14th Symp. Computer Arithmetic*, pages 70–77, 1999.

[2] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. on Computers*, 50(7):759–764, July 2001.

[3] A. Cilardo. Modular exponentiation on reconfigurable hardware. Master's thesis, Università degli Studi Federico II, Napoli, January 2003. Available at http://cds.unina.it/∼acilardo.

[4] A. Cilardo, A. Mazzeo, L. Romano, and G. Saggese. Carry-save Montgomery modular exponentiation on reconfigurable hardware. *IEEE Proc. of the Design, Automation and Test Europe (DATE) Conference*, 3:206–211, February 2004.

[5] A. Cilardo, A. Mazzeo, L. Romano, and G. Saggese. Exploring the design-space for FPGA-based implementation of RSA. *Elsevier Microprocessors and Microsystems*, 28(4):183–191, May 2004.

[6] J. Großschädl. A bit-serial multiplier architecture for finite fields $GF(p)$ and $GF(2^k)$. In *Proc. of Cryptographic Hardware and Embedded Systems - Lecture Notes in Computer Science n. 2162*. Springer-Verlag, Berlin, 2001.

[7] IEEE-P1363. *Standard Specifications For Public-Key Cryptography*. 2000.

[8] D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1981.

[9] C. K. Koç and T. Acar. Montgomery multiplication in $GF(2^k)$. *Designs, Codes and Cryptography*, 14(1):57–69, April 1998.

[10] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA, 1993.

[11] P. L. Montgomery. Modular multiplication without trial division. *Math. of Computation*, 170(44):519–521, April 1985.

[12] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures. *Commun. ACM*, 21:120–126, 1978.

[13] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 4:449–460, April 2003.

[14] E. Savaş, A. F. Tenca, M. E. Çiftçibasi, and Ç. K. Koç. Novel multiplier architectures for $GF(p)$ and $GF(2^n)$. *IEE Proceedings - Computers and Digital Techniques*, 151(2):147–160, March 2004.

[15] A. F. Tenca, E. Savaş, and Ç. K. Koç. A design framework for scalable and unified multipliers in $GF(p)$ and $GF(2^m)$. *International Journal of Computer Research*, 2004.

[16] C. D. Walter. Systolic modular multiplication. *IEEE Trans. on Computers*, 42(3):376–378, March 1993.