Towards an FPGA Architecture Optimized for Public-Key Algorithms

AJ Elbirt[*], C Paar[**]

Cryptography and Information Security Laboratory, Worcester, MA 01609

Electrical and Computer Engineering Department, Worcester Polytechnic Institute

Presented at: The SPIE's Symposium on Voice, Video, and Communications
September 20, 1999, Boston, MA, USA

ABSTRACT

Cryptographic algorithms are constantly evolving to meet security needs, and modular arithmetic is an integral part of these algorithms, especially in the case of public-key cryptosystems. To achieve optimal system performance while maintaining physical security, it is desirable to implement cryptographic algorithms in hardware. However, many public-key cryptographic algorithms require the implementation of modular arithmetic, specifically modular multiplication, for operands of 1024 bits in length. Additionally, algorithm agility is required to support algorithm independent protocols, a feature of most modern security protocols. Reprogrammability, particularly in-system reprogrammability, is critical in enabling the switching between cryptographic algorithms required for algorithm independent protocols. Field Programmable Gate Arrays (FPGAs) are a viable option for achieving this goal. Ideally, the targeted FPGA will have been designed with the architectural requirements for wide-operand modular arithmetic in mind in an effort to maximize system performance. This contribution investigates existing FPGA architectures with respect to modular multiplication. It also proposes a new FPGA architecture optimized for the wide-operand additions required for modular multiplication.

Keywords: FPGA, public-key cryptography, Montgomery Reduction, modular arithmetic

1. INTRODUCTION

To achieve optimal system performance while maintaining physical security, it is desirable to implement cryptographic algorithms in hardware. Algorithm agility is required to support algorithm independent protocols, a feature of most modern security protocols. In-system reprogrammability is critical in enabling the switching between cryptographic algorithms required for algorithm independent protocols. Additionally, many public-key cryptographic algorithms, such as RSA [1] or schemes based on the discrete logarithm in finite fields, require the implementation of modular multiplication for operands of 1024 bits or more in length. The RSA algorithm takes the form $y = x^a \bmod n$ for encryption/digital signature generation and $x = y^b \bmod n$ for decryption/digital signature verification. These operations are realized by multiple modular multiplication operations based on the value of the exponents. The square-and-multiply algorithm is the conventional algorithm used for computing the exponentiation [2]. Squaring and multiplication may be computed in parallel via the following algorithm [19]:

Algorithm 1: Compute $P = X^E \bmod N$, $E = \sum_{i=0}^{n-1} e_i 2^i$, $e_i \in \{0, 1\}$

1. $P_0 = 1$, $Z_0 = X$
2. For $i = 0$ to $n - 1$ Do
3. $Z_{i+1} = Z_i^2 \bmod N$
4. If $e_i = 1$ then $P_{i+1} = P_i \cdot Z_i \bmod N$

[*] Correspondence: Email: aelbirt@ece.wpi.edu; Telephone: 508 831 5840
[**] Correspondence: Email: christof@ece.wpi.edu; Telephone: 508 831 5061

Algorithm 1 requires *n* squaring operations [3]. The worst case number of multiplication operations is *n*, with the average number of multiplication operations being *0.5* ×*n*. Therefore, we see that the average number of operations required to perform Algorithm 1 is *1.5* ×*n*.

Montgomery has proposed a solution to avoid carry propagation when performing modular multiplication [4]. Montgomery's algorithm increases the speed of modular multiplication independent of the algorithms implementing modular addition and modular subtraction [4] and is described in detail by Koc [5] et al. Modular multiplication may be computed using Montgomery Reduction via the following algorithm [18]:

Algorithm 2: Compute $R_n = (2^{-n} \cdot A \cdot B) \bmod N$, $A = \sum_{i=0}^{n} a_i 2^i$, $a_i \in \{0, 1\}$, $B = \sum_{i=0}^{n} b_i 2^i$, $b_i \in \{0, 1\}$

1. $R_0 = 0$
2. For i = 0 to n + 2 Do
3. $q_i = R_i(0)$
4. $R_{i+1} = (R_i + a_i \cdot B + q_i \cdot N)/2$

Because Algorithm 2 computes $R_n = (2^{-n} \times A \times B) \bmod N$, a second Montgomery multiplication must be executed, multiplying the value of R_n and the constant $2^{2n} \bmod N$. However, if further multiplications are required (as in Algorithm 1), it is more efficient to scale each input by a multiplication factor of $2^{2n} \bmod N$. This results in all intermediate results being scaled by $2^{2n} \bmod N$, requiring one final Montgomery Reduction at the completion of the process, multiplying the final R_n with *1* to eliminate the scaling factor [3].

Using Algorithm 2 to perform the modular multiplication of steps 2 and 4 of Algorithm 1, the encryption and decryption functions of the RSA algorithm may be realized. In this implementation, the entire computational complexity hinges on the addition of the three *n*-bit operands in step 4 of Algorithm 2 [3]. Therefore, strategies for implementing high performance addition must be analyzed and evaluated for optimal performance of Algorithms 1 and 2.

The remainder of this paper is structured as follows. In Section 2, implementation methodologies that utilize Montgomery Reduction to achieve high performance wide-operand modular multiplication will be reviewed. Current FPGA technologies will be analyzed in Section 3 to determine the key architectural criteria required to implement these methodologies. Finally, in Section 4, an optimal FPGA architecture will be recommended to maximize the performance of modular multiplication implementations and, as a result, cryptographic algorithms.

2. PREVIOUS WORK

It is possible to implement *n*-bit adders in a number of different forms within an FPGA. To maximize the performance of Algorithms 1 and 2, ripple carry adder implementations are not practical, as they are far too slow due to long carry chains. In the case of carry look-ahead adder implementations, the required look-ahead logic resources grow in both size and complexity as the number of addition bits increases. This growth increases past the point of feasibility when used in cryptographic applications, such as the RSA algorithm, that typically require operands of 1024 bits in length [6]. While look-up-table adder implementations have been shown to be feasible for operands of small bit-width [7], they exhibit the same growth in logic as carry look-ahead adder implementations and are therefore not plausible solutions. To increase the speed of the carry structures found in current FPGA architectures, logic cells containing high performance carry chain constructs have been proposed [34]. While these architectures result in a maximum speed increase of 3.8 over current FPGA architectures, they suffer from the same growth in both size and complexity as carry look-ahead adder and look-up-table implementations. Typically, two strategies are used to implement high performance wide-operand modular addition: Redundant Representation and Systolic Arrays [3].

2.1 Redundant Representation Implementations of High Performance Modular Arithmetic

Redundant representation number systems are used in conjunction with Montgomery Reduction to avoid carry propagation and are typically implemented as a redundant radix number system [8] [9] [10] [12]. In a redundant radix number system, the operands are typically in binary representation while the intermediate partial products are in redundant representation. For iterative multiplication, the products of previous multiplications are used as the operands to successive multiplications. This results in the computational speed of iterative multiplication algorithms being decreased due to conversions that require a carry-propagate addition of long numbers at each multiplication stage [8].

Takagi [8] avoids the conversion of the operands to non-redundant form by maintaining all operands in redundant form throughout the multiplication process. However, this results in a multiplier that is nearly twice the length of a multiplier in non-redundant form [9]. To avoid this increase in resources, Shand and Vuillemin [9] propose the use of an asynchronous carry completion circuit. The asynchronous carry completion circuit detects the completion of carry propagation and provides an indication to the multiplier controller that the multiplication has completed. However, the associated cost of the asynchronous carry completion circuit is the increase in the average number of cycles required to implement modular multiplication [9]. Walter proposes another method of minimizing the propagation of carries by subtracting the modulus from intermediate partial products. However, the cost of this implementation is that an evaluation circuit is required to compare the upper bits of the partial product with the upper bits of the modulus, resulting in an increase in resources [10].

Eldridge and Walter propose a system to perform modular multiplication using Montgomery Reduction that is based on redundant representation and uses a modified version of Brickell's multiplier [11]. System performance is enhanced by interleaving modular subtractions within the partial product calculations, pre-calculation of multiples of the modulus for use in the modular subtractions, and other speed-up techniques [12]. Note that Marnane proposes a similar system to that of Eldridge and Walter [13] and that Beth and Gollmann provide an overview of Brickell's multiplication algorithm, as well as multiplication algorithms from Vielhaber, Sedlak, and Bucci [14]. Using similar techniques as Eldridge and Walter, Jeong and Burleson propose an implementation where pre-computed modulus complements and the iterative Horner's rule are used to perform modular multiplication [15]. Based on a slightly modified carry-save adder structure, the implementation operates on the most significant bits first and evaluates the intermediate result to determine which multiples of the modulus to subtract.

The major hindrance of all of these implementations is that they require either a large number of clock cycles or a large amount of storage space to compute the modular multiplication [3]. Additionally, if the modulus is to be dynamic, the stored values of the modulus multiples must be updated whenever the modulus is changed. In Jeong and Burleson's implementation [15], the stored values of the modulus complements must also be updated whenever the modulus is changed. However, Eldridge and Walter show evidence of a speed-up factor of two when comparing their implementation to other modular multiplication algorithms [12]. Therefore, with sufficient storage space, the implementation of Eldridge and Walter is a highly attractive solution, especially for systems with a dynamic modulus.

2.2 Systolic Array Implementations of High Performance Modular Arithmetic

Numerous systolic array approaches have been proposed for implementing modular arithmetic in conjunction with Montgomery Reduction. Gai and Chen have presented a VLSI-based architecture to perform modular multiplication using Montgomery Reduction [16] that is physically realizable due to the limited number of processing elements and their simplicity of design. However, the architecture requires approximately four times as many cycles to compute a modular multiplication in comparison to a more customary implementation [3]. Beth and Gollmann provide further information regarding other VLSI implementations of modular arithmetic that are not based on systolic array architectures [14].

Two-dimensional systolic array architectures, whose array size is based on the bit length of the modulus, have been shown to achieve a throughput of one modular multiplication per clock cycle [3] [17] [18] [19] [20]. Note that this throughput is theoretical and is accomplished by performing multiple modular multiplication

operations in parallel with each processing element operating on one bit, requiring n^2 processing elements, where n is the number of bits in the modulus. The number of modular multiplication operations executed in parallel is equal to twice the bit length of the modulus. For cryptographic algorithms such as RSA, the size of the modulus is typically 1024 bits in length, resulting in an extremely large array size which is not practical given the resources currently available in either VLSI or programmable logic technology.

Blum et al. alleviate the resource shortfall of two-dimensional systolic array architectures by using larger processing elements that operate over multiple bits [3]. When applied to the RSA algorithm, Blum's first implementation outperforms all others when the modulus is not fixed and therefore not hard-wired into the logical representation of the circuit. By storing the modulus, exponent, and pre-computation factor in registers and RAM, greater algorithm agility is achieved. Note that Vuillemin's implementation [21] outperforms Blum's first implementation by a factor of 1.35. However, the modulus is hard-wired and the user must update the implementation whenever the modulus is changed [21]. Therefore, Blum's first implementation is more suitable for a system where the modulus is dynamic. Also note that Blum's second implementation outperforms Blum's initial implementation by a factor of 3.40 on average and also outperforms Vuillemin's implementation by a factor of 2.96. Therefore, we see that Blum's second implementation is more suitable for any system regardless of whether or not the modulus is dynamic [35].

3. REPROGRAMMABLE DEVICES AND CRYPTOGRAPHY

In cryptographic applications, it is desirable to implement cryptographic algorithms within reprogrammable devices. Reprogrammable devices such as FPGAs are highly attractive for algorithm independent protocols – they allow for dynamic system evolution as cryptographic algorithms evolve and new algorithms are created to meet security needs. As shown in Section 2, the computational complexity of many public-key algorithms is dependent on the system's ability to perform modular addition. Both Redundant Representation and Systolic Array implementations have been shown to be viable methods for carrying out high performance modular addition. Therefore, for optimal performance, it is critical that the targeted hardware be architected with these methodologies in mind. As it is desirable to utilize reprogrammable devices such as FPGAs for the implementation of cryptographic algorithms, a great deal of knowledge may be gained by examining current FPGA architectures to determine their strengths and weaknesses when used to perform arithmetic operations. The results of this examination will reveal the key architectural features to be used in designing an FPGA for future cryptographic algorithm implementations.

3.1 FPGA Technologies and Their Suitability for Modular Arithmetic

FPGAs are normally configured based on SRAM, EPROM, EEPROM, or antifuse technology. Antifuse-based FPGAs may only be programmed once while EPROM-based and EEPROM-based FPGAs retain their programming data even after power is removed and may be electrically reprogrammed. However, reprogramming EPROM-based and EEPROM-based FPGAs requires high voltages and therefore is not typically done while the devices are in-system. SRAM-based FPGAs are fully in-system reprogrammable, though they must be reprogrammed each time the system is powered-up, typically from a ROM device containing the FPGA's configuration data [22]. Due to their in-system reprogrammability, SRAM-based FPGAs offer the greatest amount of implementation flexibility and the architectural examination will focus on these types of FPGAs. In particular, focus will be placed on the most popular families of SRAM-based FPGAs – the Xilinx XC4000, the Altera Flex 10K, and the Lucent ORCA 2CA/2TA families.

3.1.1 Xilinx FPGA Architectural Analysis

Xilinx FPGAs are comprised of a two-dimensional array of configurable logic blocks (CLBs) with horizontal and vertical routing channels used to interconnect the CLBs. The largest device of the Xilinx XC4000 family of FPGAs is comprised of a 92x92 two-dimensional array of CLBs [23]. The CLBs of the XC4000 family contain two four-input look-up-tables and two flip-flops. The look-up-tables may be configured as either combinatorial logic or as RAM. Additionally, each CLB contains circuitry to implement fast carry operations for arithmetic circuits [24]. The structure of the CLB allows for the computation of a maximum of two arithmetic bits. Hard-wired carry logic exists within each CLB to both accelerate and condense arithmetic functions [25]. The carry logic and function generators share operand and

control inputs. Dedicated high-speed routing channels are used to route the ripple-carry outputs between CLBs.

Note that a CLB's fast ripple-carry propagates along paths that run left to right for the top and bottom rows and both horizontally and vertically for all columns. This feature is critical for Systolic Array modular multiplication implementations when the size of the processing element is larger than two bits, exceeding the size of a single CLB. Redundant Representation modular multiplication implementations are also capable of taking advantage of the fast ripple-carry propagation feature. Fast ripple-carry propagation minimizes the time required to convert between redundant and non-redundant form and increases the performance of modular subtractions of multiples of the modulus from intermediate partial products.

3.1.2 Altera FPGA Architectural Analysis

Altera's Flex 10K series of FPGAs are comprised of a three-level hierarchy. The basic logic block is termed a logic element, consisting of a four-input look-up-table, a flip-flop, and fast carry-chaining for arithmetic circuits [24]. Eight logic elements are grouped and locally interconnected to form a logic array block (LAB). Finally, the LABs are interconnected via the FastTrack global interconnect. Each wire in the FastTrack global interconnect extends the full length of the device, either vertically or horizontally. Additionally, up to twelve embedded array blocks (EABs) of variable-size SRAM are provided with each row of LABs. Similar to look-up-tables, EABs may be configured to implement logical circuits [24].

Note that fast carry-chains propagate through all of the eight logic elements within a LAB [26]. However, if the carry-chain exceeds the size of a LAB, it must be routed to another LAB via the FastTrack global interconnect. While the FastTrack interconnect delay is predictable due to its uniform length, the delay is far worse than the local interconnect between the eight logic elements within a LAB. Alternatively, EABs may be used to implement large arithmetic circuits such as multipliers and adders. EABs may be configured as look-up-tables ranging from 256x8 to 2Kx1 bit in size, whose performance is independent of the complexity of the implemented logic function [27]. Additionally, EABs may be cascaded together to form either larger blocks of RAM or more complex look-up-tables [24]. One EAB may be configured to implement a single 4x4-bit multiplier [28] and Altera provides macrofunctions to implement arithmetic operations within the EABs. While configuring the EABs as wide-operand multipliers is not feasible given the typical 1024 bit operand lengths in cryptographic applications, configuring the EABs as wide-operand adders would allow for a Redundant Representation implementation for modular multiplication. However, the Altera Flex 10K architecture does not appear to be suited for Systolic Array implementations as its structure is not truly array-like and its logic elements only operate on one arithmetic bit.

3.1.3 Lucent FPGA Architectural Analysis

Lucent's Optimized Reconfigurable Cell Array (ORCA) FPGAs are similar in structure to Xilinx FPGAs in that they are comprised of a two-dimensional array of programmable function units (PFUs). The largest device of the Lucent ORCA OR2CA/2TA family of FPGAs is comprised of a 30x30 two-dimensional array of PFUs [29]. However, the PFU of the 2CA/2TA family is markedly different from the Xilinx CLB. A PFU is based on four sixteen-bit look-up-tables and may be configured as four four-input look-up-tables, two five-input look-up-tables, or one six-input look-up-table [30] [31]. Four flip-flops are available within each PFU. The structure of the PFU provides greater flexibility in implementing combinatorial logic and twice the number of flip-flops as compared to the Xilinx CLB. When the PFU is configured in the four-input look-up-table mode, several of the look-up-tables' inputs must come from the same PFU. A PFU contains arithmetic circuitry and may be configured as a RAM block [24]. A PFU can implement either a single 16x4 or two 16x2-memory blocks [31]. As in the Xilinx interconnect structure, horizontal and vertical routing channels used to interconnect the PFUs within the ORCA FPGA. In contrast with the Xilinx interconnect structure, the ORCA interconnect matrix is configured in four-bit busses, providing better support for bus applications [24].

The look-up-tables within a PFU may be configured to operate in ripple mode. In ripple mode, the look-up-table performs four-bit ripple functions using high-speed carry logic. Each look-up-table has dedicated carry logic to route to or from an adjacent look-up-table within the PFU. This allows fast arithmetic to be

implemented within one PFU [29]. Additionally, dedicated high-speed routing channels are used to route the ripple-carry outputs between PFUs [32]. In contrast with the Xilinx fast ripple-carry propagation paths between CLBs, the high-speed carry lines can be connected to any of the four sides of the PFU to any adjacent PFU, adding flexibility when performing placement and routing of the design [33]. As shown in Section 3.1.1, fast ripple-carry propagation paths are critical for both Systolic Array and Redundant Representation modular multiplication implementations. Moreover, when implemented in an ORCA FPGA, Systolic Array processing elements may operate on four arithmetic bits without exceeding the size of a single PFU, further enhancing the performance of this implementation.

4. AN FPGA ARCHITECTURE PROPOSAL FOR PUBLIC-KEY ALGORITHM IMPLEMENTATION

Based on the analysis performed in Sections 3.1.1, 3.1.2, and 3.1.3, it is evident that the structural layout of the Xilinx and Lucent FPGAs are desirable when implementing both Systolic Array and Redundant Representation modular multiplication implementations. A two-dimensional array of configurable units interconnected via horizontal and vertical routing channels yields the greatest flexibility when performing design placement and routing. Additionally, fast ripple-carry routing paths have been shown to be critical to achieve high performance addition and allowing these high-speed routing paths to connect to any of the adjacent configurable units yields greater placement flexibility of implementation elements.

When creating the configurable unit, a look-up-table format capable of computing multiple arithmetic bits within one unit will minimize ripple-carry propagation between units and maximize the utilization of each unit. Additionally, the look-up-tables must be interconnected using high-speed carry logic to maximize the performance of arithmetic operations with the configurable unit.

Finally, minimizing the number of ripple-carries between adjacent configurable units is a key factor in increasing the performance of a wide-operand addition. Increasing the number of arithmetic bits that may be computed within one configurable unit results in a decrease in the number of configurable units required for a wide-operand addition. While this will serve to increase system performance, the associated cost is an increase in the unit-delay of the configurable unit. The break-even point occurs when the unit-delay of the configurable unit is equivalent to the delay in routing ripple-carries to adjacent units via the fast ripple-carry routing paths. Therefore, the number of arithmetic bits that may be computed within one configurable unit must be carefully chosen based on the timing associated with the fast ripple-carry routing paths. In terms of current technology, the carry delay of a sixteen-bit adder is approximately equivalent to one CLB delay in a Xilinx XC4000 FPGA [3]. It can therefore be deduced that a configurable unit can compute sixteen arithmetic bits in approximately the same amount of time required to propagate the associated ripple-carry output to an adjacent configurable unit. This architecture results in an n-bit addition requiring $n/16$ configurable units and $2n/16 - 1$ unit delays.

A block diagram of the proposed configurable unit is shown in Figure 1 and is comprised of the equivalent of four ORCA PFUs operating in parallel. Each set of four sixteen-bit look-up-tables would be configurable as four four-input look-up-tables, two five-input look-up-tables, or one six-input look-up-table, providing greater flexibility in implementing combinatorial logic. A total of sixteen flip-flops would be available, with each four-bit block having high-speed ripple carry logic both within the block itself as well as between adjacent blocks.

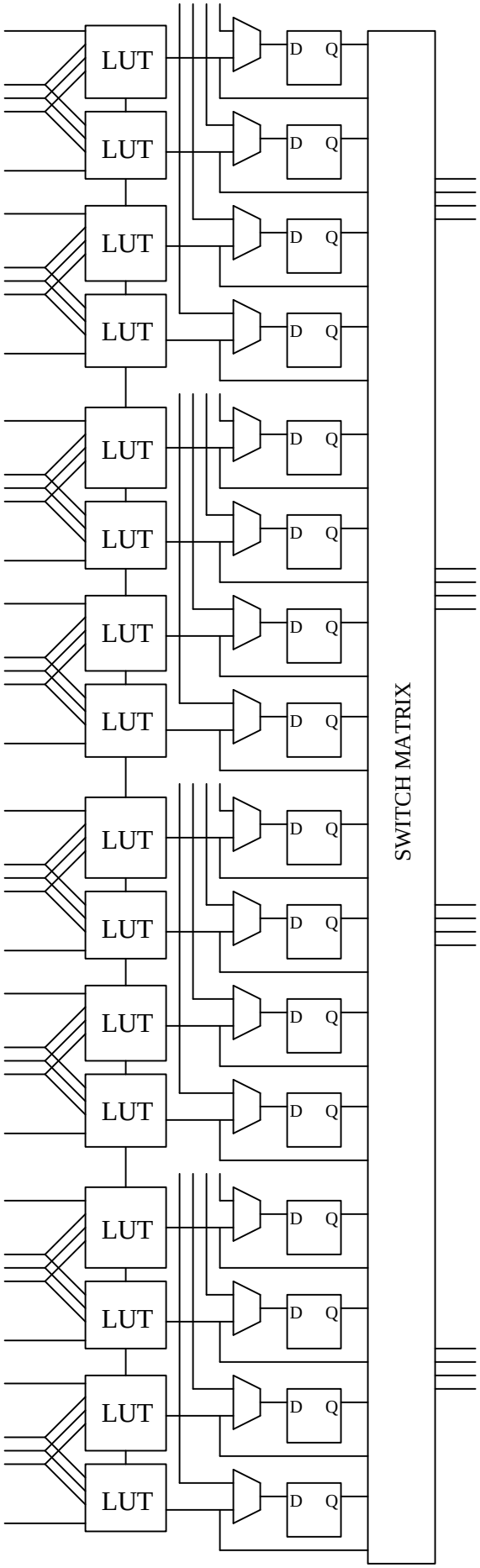

Figure 1 – Configurable Unit Block Diagram

A block diagram of the high-speed routing channels used to route the ripple-carry outputs between configurable units is shown in Figure 2. These high-speed routing channels allow for the propagation of fast ripple-carry paths from a configurable unit to any adjacent configurable unit, adding routing flexibility to the architecture

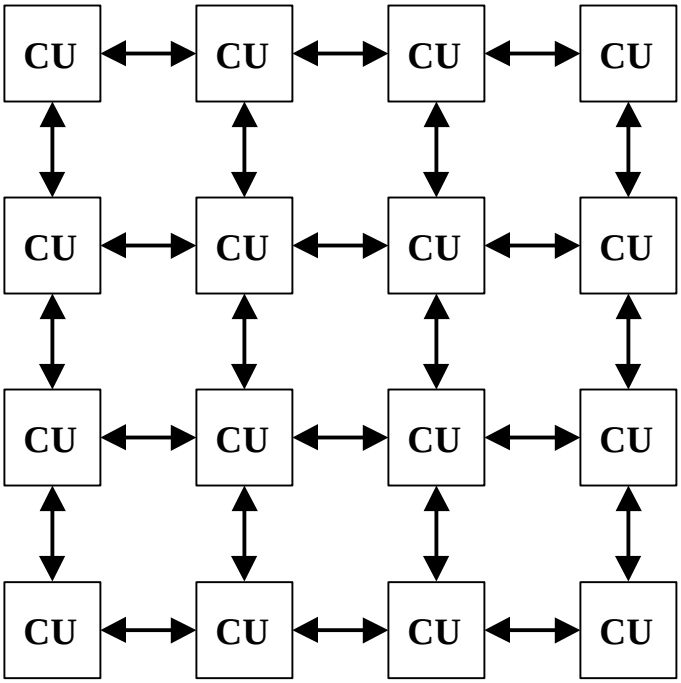

Figure 2 – Fast Carry Propagation Paths for the Proposed FPGA Architecture

Combining all of the aforementioned techniques leads to a theoretical FPGA architecture that will maximize the performance of high-speed addition algorithms. This results in a performance increase for Redundant Representation and Systolic Array implementations of modular multiplication and, as a result, cryptographic algorithms.

5. CONCLUSIONS

The application of modular arithmetic to cryptographic algorithms has been examined and implementation methodologies that utilize Montgomery Reduction to achieve high performance wide-operand modular multiplication have been investigated. This investigation led to the determination that the computational complexity of Montgomery Reduction hinges on the ability to perform high-speed wide-operand addition. Redundant Representation and Systolic Array implementations, the two strategies typically used to perform high-speed wide-operand addition, were evaluated based on their performance. FPGAs that are in-system reprogrammable were investigated to determine the key architectural criteria for implementing high performance wide-operand addition. Finally, an FPGA architecture designed for implementing wide-operand modular arithmetic has been suggested. This architecture will maximize the performance of modular arithmetic implementations, resulting in the implementation of high performance cryptographic algorithms.

6. REFERENCES

[1] Rivest, R., A. Shamir, et al. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM **21**(2): 120-126.

[2] Stinson, D. R. (1995). Cryptography, Theory and Practice. Boca Raton, CRC Press.

[3] Blum, T. and C. Paar (1998). Montgomery Modular Exponentiation on Reconfigurable Hardware. Submitted to 14th IEEE Symposium on Computer Arithmetic.

[4] Montgomery, P. L. (1985). "Modular Multiplication Without Trial Division." Mathematics of Computation **44**(170): 519-521.

[5] Koc, C. K., T. Acar, et al. (1996). "Analyzing and Comparing Montgomery Multiplication Algorithms." IEEE Micro **16**(3): 26-33.

[6] Yu, W. W. H. and S. Xing (1996). Performance Evaluation of FPGA Implementations of High-Speed Addition Algorithms. Proceedings of the SPIE - The-International-Society-for-Optical-Engineering, SPIE.

[7] Ling, H. (1990). "An Approach to Implementing Multiplication with Small Tables." IEEE Transactions on Computers **39**(5): 717-718.

[8] Takagi, N. (1991). A Radix-4 Modular Multiplication Hardware Algorithm Efficient for Iterative Modular Multiplications. 10th IEEE Symposium on Computer Arithmetic, Los Alamitos, CA, IEEE.

[9] Shand, M. and J. Vuillemin (1993). Fast Implementations of RSA Cryptography. 11th IEEE Symposium on Computer Arithmetic, Windsor, Ontario, Canada, IEEE.

[10] Walter, C. (1991). "Fast Modular Multiplication Using Power-2 Radix." International Journal of Computer Mathematics **39**(1-2): 21-28.

[11] Brickell, E. F. (1983). A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography. Advances in Cryptology - CRYPTO '82, New York, Plenum Press.

[12] Eldridge, S. E. and C. D. Walter (1993). "Hardware Implementation of Montgomery's Modular Multiplication Algorithm." IEEE Transactions on Computers **42**(6): 693-699.

[13] Marnane, W. P. (1998). "Optimised Bit Serial Modular Multiplier for Implementation on Field Programmable Gate Arrays." Electronics Letters **34**(8): 738-739.

[14] Beth, T. and D. Gollmann (1989). "Algorithm Engineering for Public Key Algorithms." IEEE Journal on Selected Areas in Communications **7**(4): 458-465.

[15] Jeong, Y. and W. Burleson (1997). "VLSI Array Algorithms and Architectures for RSA Modular Multiplication." IEEE Transactions on Very Large Scale Integration (VLSI) Systems **5**(2): 211-217.

[16] Gai, W. and H. Chen (1996). A Systolic Linear Array for Modular Multiplication. 2nd International Conference on ASIC, Shanghai, China, IEEE.

[17] Iwamura, K., T. Matsumoto, et al. (1994). "Montgomery Modular-Multiplication Method and Systolic Arrays Suitable for Modular Exponentiation." Electronics and Communications in Japan, Part 3 **77**(3): 40-50.

[18] Walter, C. D. (1993). "Systolic Modular Multiplication." IEEE Transactions on Computers **42**(3): 376-378.

[19] Wang, P. A., W. Tsai, et al. (1997). New VLSI Architectures of RSA Public-Key Cryptosystem. 1997 IEEE International Symposium on Circuits and Systems, Hong Kong, IEEE.

[20] Tiountchik, A. A. (1998). "Systolic Modular Exponentiation Via Montgomery Algorithm." Electronics Letters **34**(9): 874-875.

[21] Vuillemin, J., P. Bertin, et al. (1996). "Programmable Active Memories: Reconfigurable Systems Come of Age." IEEE Transactions on Very Large Scale Integration (VLSI) Systems **4**(1): 56-69.

[22] Hauck, S. (1998). "The Roles of FPGAs in Reprogrammable Systems." Proceedings of the IEEE **86**(4): 615-638.

[23] Xilinx Inc. (1996). The Programmable Logic Data Book. San Jose, Xilinx Inc.

[24] Brown, S. and J. Rose (1996). "FPGA and CPLD Architectures: A Tutorial." IEEE Design & Test fo Computers **13**(2): 42-57.

[25] New, B. (1996). Using the Dedicated Carry Logic in XC4000E. San Jose, Xilinx, Inc.**:** 10.

[26] Altera Corporation (1996). "Advantages of Carry Chains in FLEX 8000 Devices." News & Views(4): 20-21.

[27] Altera Corporation (1996). "Using the FLEX 10K Embedded Array for Logic." News & Views(1).

[28] Altera Corporation (1995). "FLEX 10K: 100,000-Gate Embedded Array Programmable Logic Family." News & Views(2).

[29] Lucent Technologies Inc. (1998). Lucent Technologies Field-Programmable Gate Arrays Data Book. Allentown, Lucent Technologies Inc.

[30] Synario Design Automation (1996). "Optimized Architectures Using Distributed On-Chip SRAM in FPGAs." App Review(June).

[31] Lucent Technologies Microelectronics Group (1997). VHDL Coding for ORCA FPGAs Part 1: Synchronous Logic and Multiplexors. Allentown, Lucent Technologies Microelectronics Group**:** 28.

[32] Synario Design Automation (1996). "Create Multiply-Accumulate Functions in ORCA FPGAs." App Review(September).

[33] AT&T Microelectronics (1995). Designing a Data-Path Circuit in ORCA FPGAs. Allentown, AT&T Microelectronics**:** 8.

[34] Hauck, S., M. Hosler, et al. "High-Performance Carry Chains for FPGAs." to appear in IEEE Transactions on VLSI Systems.

[35] Blum, T. (1999). "Modular Exponentiation on Reconfigurable Hardware." Master's Thesis, Worcester Polytechnic Institute(April).