Implementation of an FFT hardware accelerator for security applications

Domenico Argenziano
University of Naples Federico II
domenico.argenziano@unina.it

***Abstract*—This work presents the architecture and implementation of a dedicated hardware accelerator for cryptographic purposes. In particular, the paper describes a highly customized FPGA design targeting the most time consuming operation used by the encryption primitives, long-integer multiplication, based on an efficient implementation of the Schönhage-Strassen algorithm (SSA) exploiting the properties of the Discrete Fourier Transform over finite fields. The paper presents the architecture of the accelerator, the details of its implementation, as well as the main performance results.**

I. Introduction

Cryptographic processing has become increasingly important in current applications, demanding more and more efficient hardware-based support [1]. Reconfigurable hardware, particularly FPGAs, have proved to be an effective platform for cryptographic processing. In fact, FPGAs provide to general applications a number of benefits compared to both programmable systems and traditional ASICs, ranging from high-level design and software-like programming [2], [3], [4], [5] to application-specific testing [8], but they are particularly suitable for cryptographic algorithms because these have peculiar characteristics (e.g. integer computation, often bit-level manipulation, etc.) that make standard platforms like CPUs and GPUs less competitives, while hardware reconfigurability allows the designer to customize the system possibly based on specific parameters, e.g. a cryptographic key. Consequently, the use of FPGAs has been demonstrated in a range of cryptographic applications for processing acceleration [6], [7], [10] as well as for cryptanalytic purposes [11], [12], [13], [14].

This work in particular focused on the FPGA implementation of a dedicated hardware accelerator supporting a very demanding operation, i.e. long-integer multiplication, based on an efficient implementation of the Schönhage-Strassen algorithm (SSA) which exploits the properties of the Discrete Fourier Transform over integers. The accelerator is specifically conceived for use in the context of Homomorphic Encryption [15], a relatively new –and extremely compute-intensive– cryptographic technique allowing computation to take place on encrypted data. The implementation is based on an advanced FPGA platform, i.e. an Altera's Stratix V, a high-end platform that can be reasonably assumed to be available on the server side in cloud settings, where computing on encrypted data might be a highly desirable feature in the future.

The paper is organized as follows. Section II presents the background of this work. Section III describes the implemented algorithm. Section IV provides a few details on the implementation and results. Section V concludes the paper with some final comments.

II. Background

During the last years cloud computing has emerged as an important paradigm shift for a large class of applications [16], [17], [18]. Security is a major concern in current cloud settings, as user data is moved to an external server and processed at a remote site. Homomorphic encryption (HE) is a cryptographic technique which potentially addresses this concern as it allows computation to take place on encrypted data. While the HE concept is known since the early stages of modern cryptography, only in 2009 a seminal work by Gentry [15] showed a practical incarnation of a fully homomorphic encryption scheme. A first implementation of a variant of Gentry's scheme was proposed by Gentry and Halevi [19], while more recent software implementations include [20] and [23]. Available open-source libraries include *hcrypt* [24] and *HElib* [25], reaching a speed-up of 12X over the previous solutions. Concerning the computing platforms, [26] presented an efficient GPU implementation based on NVIDIA GTX 690. [27] compared a solution based on FPGAs vs. GPUs (Altera Stratix V vs. NVIDIA Tesla C2050), with the FPGA version reaching a 2X performance compared to the GPU, with lower power comsumption. [28] proposed a full custom (ASIC) implementation of large operand multiplication. Their design is based on fast multiplication and also includes a 768 Kbit cache to minimize I/O transactions. The same authors also built the first custom hardware implementation [29] of the cryptographic primitives of Gentry-Halevi FHE scheme. The design includes optimizations previously introduced in [26] to reduce the number of FFT computations.

As mentioned above, most existing implementations focus on optimizing the most time consuming operation used by the various encryption schemes, long-operand multiplication. This operation is performed on operands of millions of bits and can benefit from better asymptotic algorithms. The work in [30] proposed an FFT-based large integer multiplier, along with a Barrett reduction module. Their design was based on the FHE scheme presented in [31], [33] and was implemented on a Xilinx Virtex-7 FPGA, resulting in a significant speedup compared to existing software implementations.

III. Algorithm

We aim to exploit the potential of a highly-customized FPGA design for accelerating integer multiplication. Since this operation is performed on ultra-large operands, in the order of millions of bits, we relied on asymptotically faster (but inherently more complex) algorithms, particularly the Schönhage-Strassen algorithm (SSA), exploiting the properties of the Discrete Fourier Transform, which turn out to be

advantageous for operands of at least $100,000$ bits. In essence, the algorithm computes $c = a \cdot b$ by regarding operands a and b as polynomials and decomposing them into groups of m bits; performing the integer FFT of a and b, hence getting A and B; then computing $C = A \cdot B$ component-wise; deriving the Inverse FFT of C (call it c'); and last, computing the final result c as the collection of the components of c'.

The computational complexity of SSA is $O(n \cdot \log n \cdot \log\log n)$. The most time consuming operation in the SSA is the computation of the FFT (and IFFT). By using the divide et conquer approach of the Cooley-Tukey scheme, it can be executed in an $O(N \cdot \log N)$ time, where N is the number of points on which the transform is computed. Instead of a classic binary recursive splitting approach relying on a radix-2 transform, we used higher order transforms as the basic block for FFT. This comes from the general FFT splitting approach:

$$F[k] = \sum_{n=0}^{N-1} f[n]\omega^{n \cdot k}, k = 0, 1, \ldots, N-1$$

where N can be represented as $N = N_1 \cdot N_2$ so $F[k]$ and $f[n]$ can be split in N_1 sub-sequences of lenght N_2. Let: $n = N_2 n_1 + n_2$ and $k = N_1 k_2 + k_1$ with $n_1 = 0, 1, , N_1 - 1, n_2 = 0, 1, , N_2 - 1, k_1 = 0, 1, , N_1 - 1$ and $k_2 = 0, 1, , N_2 - 1$.

So ω_N^{n*k} can be expressed as

$$\omega_N^{n*k} = \omega_N^{(N_2 n_1 + n_2)(N_1 k_2 + k_1)} =$$

$$= \omega_N^{N_2 n_1 k_1} \cdot \omega_N^{N_1 k_2 n_2} \cdot \omega_N^{N_2 N_1 n_1 k_2} \cdot \omega_N^{n_2 k_1} =$$

$$= \omega_{N_1}^{n_1 k_1} \cdot \omega_N^{n_2 k_1} \cdot \omega_{N_2}^{n_2 k_2}$$

Therefore, the DFT can be written as:

$$F[N_1 k_2 + k_1] = \sum_{n=0}^{N-1} f[n]\omega_N^{kn} =$$

$$\sum_{n_2=0}^{N_2-1} [(\sum_{n_1=0}^{N_1-1} f[N_2 n_1 + n_2] \cdot \omega_{N_1}^{n_1 k_1}) \cdot \omega_N^{n_2 k_1}] \cdot \omega_{N_2}^{n_2 k_2}$$

We chose to perform the computation in the finite field $\mathbb{Z}/p\mathbb{Z}$, with prime p. The computation of a finite-field FFT requires three operations: modulo addition, modulo subtraction, and multiplication. By selecting a proper prime p, the modular multiplication in the finite field can be computed rapidly. In this paper, we chose the prime $p = 2^{64} - 2^{32} + 1$ from a special family of prime numbers, called Solinas primes. This choice enables us to perform most multiplications as simple shifts.

In this finite field, FFT can be represents as:

$$F[k] = \sum_{n=0}^{N-1} f[n] \cdot \omega_N^{n*k} \pmod{p}$$

In our implementation we set the base b to 24. Therefore, every sample is 24-bit wide and we have a total of 32K samples. We need to apply 64K-point FFT, in order to accommodate the result of a multiplication. By applying the general FFT splitting approach on the 64k-point FFT we have:

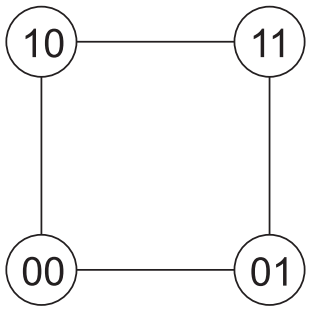

Fig. 1. Hypercube topology with 4 nodes.

$$F[k] = \sum_{n_1=0}^{15} \omega_{16}^{n_2 \cdot k_2} \cdot (\sum_{n_2=0}^{4095} f[n] \cdot \omega_{4096}^{n_1 \cdot k_1}) \cdot \omega_{64K}^{n_2 \cdot k_1} \pmod{p}$$

The formula can be recursively applied to the inner 4096-point FFT, decomposing it as 64 FFTs, each on 64 points. Each FFT based multiplier will contain one Radix-64 FFT unit and one Radix-16 FFT unit. A 64K-point FFT will be computed in three subsequent stages; the first two requiring 1024 Radix-64 FFTs each, the last using 4096 Radix-16 FFTs. Each stage can be efficiently parallelized, according to the available computing resources.

IV. Implementation and results

For the implementation of a 64K points FFT accelerator, we chose an on-chip distributed approach, based on the use of several processing elements connected in some topology. Specifically we chose a hypercube as in Figure 1.

The distributed approach is more suited for FFT computing, with respect to other approaches, such as shared memory/shared bus, given the peculiar data exchange pattern. The number of communication stages is d, the hypercube dimension; in each stage, each node communicates only with one of its neighbors. The overall communication pattern allows each node to eventually own data which comes from any other node. The architecture of the processing element (or node) is shown in Figure 2.

Our approach is inspired by the work of Huang and Wang [27], where they use a shared memory approach, with local buffers; instead, we adopted a complete distributed approach and introduced our own optimizations. The core computing element is the Radix-64/16 FFT unit, which computes the basic FFT blocks; since in our distributed scheme we need both to compute and communicate at the same time, double buffering is used: a buffer containing current input values is used to feed the FFT unit while the other is written with new values computed by the same node and coming from one of its neighbors. At the end of a computation stage, the roles of the buffers are swapped. Buffers are based on a bank architecture which uses the SRAM primitive blocks of the underlying FPGA architecture. Our memory allows 8 read/write operations in parallel with different access patterns for read and write in order to support the FFT butterfly access pattern. Additionally, we also need a bank of modular multipliers, for twiddle factor multiplications, needed between two consecutive FFT computation stages.

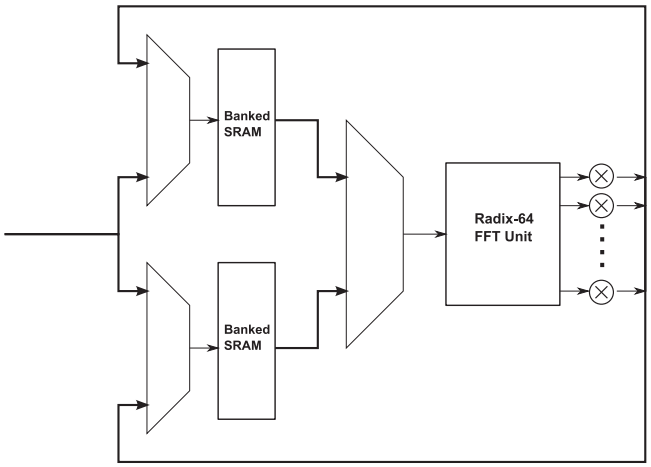

Fig. 2. Architecture of a 64K FFT processing element.

A. Implementation and performance

We aim to implement our hardware accelerator for super-sized integer operations on a high-end FPGA platforms, such as *Altera's Stratix V*. We set the 5SGSMD8N3F45I4 device for simulation and synthesis. Currently most components of the above processing element have been implemented, namely the Radix-64 unit, banked memory, and modular multiplicators. For the efficient implementation of the accelerator, a number of very low-level techniques were evaluated for fast FPGA-based operations, e.g. previous results on efficient arithmetic [22], [34] that might be particularly advantageous for long operands.

The device could accomodate four processing elements to be implemented on a Stratix V, reaching an operating frequency of 180 MHz. We present below a performance estimation, first with respect to the FFT multiplication and then to cryptographic primitives. The Radix-64 unit is able to output an FFT every 8 clock cycles, while a radix-16 will take only 2 clock cycles; each unit is fully pipelined. A single 64K points FFT will take $T_{FFT} = (T_C \cdot 8 \cdot 1024) \cdot 2/P + (T_C \cdot 2) \cdot 4096/P$, where T_C is the clock period, around 5.6 ns (frequency: 180 MHz), while P is the number of Processing Elements, here 4. The first term refers to the first two stages, with 1024 FFTs on 64 points. The second term refers to the last stage where 4096 FFTs on 16 points are computed. By substituting the clock period and using four Processing Elements, we get $T_{FFT} = 22938ns + 11469ns \approx 34.4\mu s$.

A full SSA multiplication requires three FFTs (two direct FFTs for the inputs and one inverse FFT for the output). Besides, we must perform a component-wise multiplication on two vectors of 64K components and the final carry recovery addition on the inverse FFT components. Our processing element is already able to perform 8 multiplications at a time (with the twiddle factors) and can be refitted to perform the component-wise multiplications, that will take $T_{MUL} = T_C \cdot 65536/(8 \cdot P) \approx 11.5\mu s$

The final carry recovery can be efficiently computed with an *ad hoc* adder tree structure, using carry saving and subsequent accumulation. The resulting time for a complete SSA multiplication is around $150\mu s$.

With respect to a complete cryptographic primitive, e.g. encryption, we must consider that our FPGA accelerator is not intended for a specific scheme, but may support any algorithm which requires operations on very large operands. As an example, we will refer to the scheme presented in [35]. We will suppose to adopt a high level architectural approach similar to that presented in [29], which implemented an ASIC accelerator for such scheme, namely, the cryptographic primitives Encrypt, Decrypt, and Recrypt. The authors adopted various optimizations, such as precomputing and working in the *Fourier domain* as much as possible. Considering their timing performance for FFT and multiplication, we can infer an estimate for our architecture performance, as described in Table I.

TABLE I. PERFORMANCE EXTIMATION

Barrett Modular Reduction	207 μs
Encryption	350 μs
Decryption	310 μs

V. CONCLUSION

This work presented an FPGA implementation of a dedicated hardware accelerator for long-operand multiplication as a basic building block of a homomorphic encryption processor. The paper described the high-level architectural concept and implementation as well as the experimental results collected from hardware synthesis.

ACKNOWLEDGMENT

The presentation of this work is supported by the European Commission in the framework of the H2020-FETHPC-2014 project n. 671668 - *MANGO: exploring Manycore Architectures for Next-GeneratiOn HPC systems*.

REFERENCES

[1] Çetin K. Koç, *Cryptographic Engineering*. Springer Science & Business Medi, 2008.

[2] P. Coussy and A. Morawiec, *High-Level Synthesis from Algorithm to Digital Circuit*. Springer, 2008.

[3] A. Cilardo, L. Gallo, A. Mazzeo, and N. Mazzocca, "Efficient and scalable OpenMP-based system-level design," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 988–991.

[4] A. Cilardo and L. Gallo, "Improving multibank memory access parallelism with lattice-based partitioning," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, pp. 45:1–45:25, Jan. 2015.

[5] A. Cilardo, L. Gallo, and N. Mazzocca, "Design space exploration for high-level synthesis of multi-threaded applications," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1171–1183, 2013.

[6] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," in *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, ser. LNCS, vol. 4727. Springer, 2007, pp. 272–288.

[7] G. de Meulenaer, F. Gosset, M. M. de Dormale, and J.-J. Quisquater, "Integer factorization based on elliptic curve method: Towards better exploitation of reconfigurable hardware," in *Proceedings of the 15th Annual Symposium on Field-Programmable Custom Computing Machines (FCCM) 2007*. IEEE, 2007, pp. 197–206.

[8] A. Cilardo, "New techniques and tools for application-dependent testing of FPGA-based components," *Industrial Informatics, IEEE Transactions on*, vol. 11, no. 1, pp. 94–103, Feb 2015.

[9] ——, "Efficient bit-parallel GF(2^m) multiplier for a large class of irreducible pentanomials," *Computers, IEEE Transactions on*, vol. 58, no. 7, pp. 1001–1008, July 2009.

[10] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "High speed flexible pairing cryptoprocessor on FPGA platform," in *Proceedings of the 4th international conference on Pairing-based cryptography*. Springer-Verlag, 2010, pp. 450–466.

[11] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking ciphers with COPACOBANA - a cost-optimized parallel code breaker," in *Proceedings of the 8th international conference on Cryptographic Hardware and Embedded Systems (CHES)*. Springer-Verlag, 2006, pp. 101–118.

[12] T. Güneysu, T. Kasper, M. Novotný, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1498–1513, 2008.

[13] (2011, Jan.) Rivyera S3-5000. [Online]. Available: http://www.sciengines.com/

[14] A. Cilardo and N. Mazzocca, "Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware," *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 5, pp. 810–820, May 2013.

[15] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[16] Y. K. Sinjilawi, M. Q. AL-Nabhan, and E. A. Abu-Shanab, "Addressing security and privacy issues in cloud computing," *Journal of Emerging Technologies in Web Intelligence*, vol. 5, no. 2, pp. 192–199, 2014.

[17] F. Amato, A. Mazzeo, V. Moscato, and A. Picariello, "A framework for semantic interoperability over the cloud," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. IEEE, 2013, pp. 1259–1264.

[18] F. Amato, A. Chianese, V. Moscato, A. Picariello, and G. Sperli, "Snops: a smart environment for cultural heritage applications," in *Proceedings of the twelfth international workshop on Web information and data management*. ACM, 2012, pp. 49–56.

[19] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," *Proc. Advances in Cryptology-EUROCRYPT 2011*, pp. 129–148, 2011.

[20] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the aes circuit," *IACR Cryptology ePrint Archive*, vol. 2012, p. 99, 2012.

[21] A. Cilardo, "Exploring the potential of threshold logic for cryptography-related operations," *Computers, IEEE Transactions on*, vol. 60, no. 4, pp. 452–462, April 2011.

[22] ——, "Modular inversion based on digit-level speculative addition," *Electronics Letters*, vol. 49, no. 25, pp. 1609–1610, December 2013.

[23] J. Coron, T. Lepoint, and M. Tibouchi, "Batch fully homomorphic encryption over the integers," *IACR Cryptology ePrint Archive*, vol. 2013, p. 36, 2013.

[24] H. Perl, M. Brenner, and M. Smith. (2011) hcrypt. [Online]. Available: http://www.hcrypt.com/scarab-library/

[25] S. Halevi and V. Shoup. (2012) Helib, homomorphic encryption library. [Online]. Available: https://github.com/shaih/HElib

[26] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," vol. 99, p. 1, 2013.

[27] W. Wang and X. Huang, "Fpga implementation of a large-number multiplier for fully homomorphic encryption," *ISCAS*, pp. 2589–2592, 2013.

[28] Y. Doröz, E. Öztürk, and B. Sunar, "Evaluating the hardware performance of a million-bit multiplier," *Digital System Design (DSD),16th Euromicro Conference on*, 2013.

[29] Y. Doröz, E. öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *draft, Under Review*, 2013.

[30] X. Cao, C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "High speed fully homomorphic encryption over the integers," *Workshop on Applied Homomorphic Cryptography, to appear*, 2014.

[31] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," *CRYPTO*, pp. 487–504, 2011.

[32] A. Cilardo, E. Fusella, L. Gallo, and A. Mazzeo, "Exploiting concurrency for the automated synthesis of MPSoC interconnects," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 3, p. 57, 2015.

[33] J. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," *EUROCRYPT*, pp. 446–464, 2012.

[34] A. Cilardo, D. De Caro, N. Petra, F. Caserta, N. Mazzocca, E. Napoli, and A. Strollo, "High speed speculative multipliers based on speculative carry-save tree," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 12, pp. 3426–3435, Dec 2014.

[35] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," *Public Key Cryptography*, pp. 420–443, 2010.