Hardware Design of an NTT-Based Polynomial Multiplier

Conference Paper · November 2014

DOI: 10.1109/SPL.2014.7002209

CITATIONS
12

READS
1,571

2 authors:

Claudia Patricia Renteria
Universidad del Valle (Colombia)
10 PUBLICATIONS **81** CITATIONS
SEE PROFILE

Jaime velasco-medina
Universidad del Valle (Colombia)
110 PUBLICATIONS **569** CITATIONS
SEE PROFILE

Hardware Design of an n-coefficient NTT-Based Polynomial Multiplier

C. P. Rentería-Mejía and J. Velasco-Medina
Bionanoelectronics Research Group
Universidad del Valle. School of Electrical and Electronics Engineering.
Cali, Colombia
E-mail: {claudia.patricia.renteria, jaime.velasco}@correounivalle.edu.co

***Abstract*— This paper presents the design of a parameterizable n-coefficient polynomial multiplier, which performs polynomial multiplication in $Z_p[x]/\langle x^n + 1\rangle$. This multiplier is based on an *n*-point radix-2 NTT-core that is designed using a systolic array. The designed NTT-based polynomial multiplier is described in generic structural VHDL; synthesized on the Stratix EP4SGX230KF40C2 using Quartus II V. 13; verified using Modelsim; and performs the product of two polynomials of degree 4095 in 95.64 µs. The hardware synthesis and performance results show that the designed polynomial multiplier presents a good area-time trade-off and it is suitable for hardware implementations of lattice-based cryptosystems.**

Keywords*— *hardware architectures for cryptography; lattice-based cryptography; NTT; polynomial multiplier; post-quantum cryptography

I. INTRODUCTION

Public key cryptosystems such as RSA and ECC can be broken in a polynomial time using the Shor's algorithm executed in a large scale quantum computer [1]. Taking into account the above fact, it is necessary to develop new public key cryptosystems that resist attacks from quantum computers. These new cryptosystems are known as post-quantum cryptosystems and currently there are four types: code-based, hash-based, lattice-based and multivariate-quadratic-equations [2].

In this context, lattice-based cryptography is one of the most promising candidates for public key cryptosystems in the age of quantum computing, due to its security proofs consider the worst-case hardness, and its software and hardware implementations are relatively efficient [2].

Currently, lattice-based cryptography uses ideal lattices in order to achieve practical hardware implementations, i.e., this cryptography uses less area resources and execution time than hardware implementations of the cryptography based on arbitrary lattices [3]. In the case of ideal lattices, the arithmetic operations are carried out over a polynomial ring, where the polynomial multiplication is the operation that requires the most processing time. The polynomial multiplication can be performed using one of these algorithms: School-book, Karatsuba-Ofman [4] and FFT [5], where the last one is the most efficient algorithm because its computational complexity is $O(n\log n)$. However, it is important to mention that for the case of the polynomial multiplication, the FFT is replaced by its equivalent in the finite field over GF(p), i.e., Number Theoretic Transform (NTT) [6].

This work was supported by Colciencias and Universidad del Valle.

In the literature, the NTT hardware implementations are performed using parallel or conventional sequential processing. In this work, we design an *n*-point radix-2 NTT-core using a systolic array. The *n*-point NTT-core is used to design a parameterizable polynomial multiplier, which performs polynomial multiplication in the ring $Z_p[x]/\langle x^n + 1\rangle$.

This paper is organized as follows. Section II presents previous hardware implementations of polynomial multipliers. Section III describes the basic concepts about NTT and NTT-based polynomial multiplication. Section IV presents the design of the *n*-point NTT-core using one systolic array and the parameterizable *n*-coefficient polynomial multiplier. Section V presents the synthesis and performance results for the proposed design. Section VI presents the conclusions and future work.

II. PREVIOUS WORKS

The first hardware implementation of an NTT-based polynomial multiplier is presented in [6], and it carries out the polynomial product over $GF(2^{113})$ using a 16-point radix-4 NTT. In [7] is presented a polynomial multiplier for a lattice-based cryptoprocessor which uses a parallel NTT design that demands a lot of area resources. In [3] is designed a polynomial multiplier that carries out the product over $Z_p[x]/\langle x^n+1\rangle$ using a memory-based radix-2 NTT, which has one butterfly element. In that implementation the twiddle factors and some pre-computations are stored in a ROM. In [8] is designed a multiplier similar of the above one, but the twiddle factors and other values are calculated on-the-fly instead of be pre-computed and stored. Then, that implementation demands less area resources than [3]. In this work, we design an *n*-point radix-2 NTT-core using a systolic array, where each Processing Element (PE) has one radix-2 butterfly block. It is different from the standard implementations in sense that the PE is a modified R2SDF block, which is designed using a modular arithmetic butterfly block B_ω. To the best of our knowledge the *n*-point NTT-core is the first one based on a systolic array. Then, this design allows achieving a core faster than the previous ones presented in the literature.

III. POLYNOMIAL MULTIPLICATION IN $Z_P[X]/\langle X^N+1\rangle$

Lattice-based cryptography that uses ideal lattices speeds up computations and saves storage space for the keys [9]. Those lattices correspond to ideals in the ring $R=Z_p/\langle f\rangle$ for some monic polynomial $f \in Z[x]$ of degree n [10]. For

978-1-4799-6848-0/14/$31.00 ©2014 IEEE

example, the polynomial variant of LWE-based encryption uses anti-cyclic lattices, which correspond to ideals in the ring $R=Z_p/\langle x^n+1\rangle$, where n is a power of two and p is a prime such that $p\equiv 1 \bmod 2n$ [10]. In this case, the polynomial multiplication is the operation that requires the most processing time [2].

Let $A(x)$ and $B(x)$ be two polynomials of degree-(n-1):

$$A(x) = \sum_{i=0}^{n-1} a_i\, x^i,\ B(x) = \sum_{i=0}^{n-1} b_i\, x^i$$

Then, the product $C(x) = A(x)\times B(x)$ can be calculated using the Schoolbook algorithm, which is described by Equation (1):

$$C(x) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i\, b_j x^{i+j} \quad (1)$$

The above algorithm has a computational complexity of $O(n^2)$. However, there are other algorithms that perform the polynomial multiplication using less operations, e.g., the Karatsuba and FFT algorithms have a computational complexity of $O(n^{\log 3})$ and $O(n\log(n))$, respectively [11]. Then, the FFT algorithm is the most efficient to perform the polynomial multiplication, and it is executed using the following procedure:

1. Calculate the FFT of the set of coefficients of each polynomial, i.e., evaluate the two polynomials at a special set of values.
2. Calculate the point-wise product of the above FFTs, i.e., perform the point-wise multiplication of the above point-value forms.
3. Calculate the inverse FFT of the above point-wise product, i.e., interpolate the point-wise product in order to obtain the coefficients of the polynomial product.

A. Number Theoretic Transform

The NTT is the FFT defined in a finite field over GF(p) [2]; then the NTT doesn't use floating point numbers and complex arithmetic. Let $a = (a_0,\ldots,a_{n-1})$ be a vector with elements in Z_p and let ω be a primitive n-th root of unity. Then $\mathrm{NTT}_\omega(a)$ is defined as:

$$A_i = \sum_{j=0}^{n-1} a_j\, \omega^{ij} \bmod p,$$
$$where\ i = 0,1,\ldots,n-1 \quad (2)$$

And the $\mathrm{NTT}_\omega^{-1}(A)$ is defined as:

$$a_i = n^{-1}\sum_{j=0}^{n-1} A_j\, \omega^{-ij} \bmod p,$$
$$where\ i = 0,1,\ldots,n-1 \quad (3)$$

The NTT exists if and only if n divides q-1 for every prime factor q of p, and ω is a primitive n-th root of unity if $\omega^n = 1 \bmod p$ and $\omega^{n/q} - 1 \neq 0 \bmod p$ for any prime factor q of n [2]. For the case where p is prime, n is a power of two and $n \bmod (p-1) \equiv 1$, the NTT algorithm has a computational complexity of $O(n\log n)$.

B. NTT-Based Polynomial Multiplication

The polynomial multiplication using NTT can be carried out using the Convolution Theorem: Let ω be a primitive $2n$-th root of unity in Z_p; let $a = (a_0,\ldots,a_{n-1})$ and $b = (b_0,\ldots,b_{n-1})$ be vectors of length n in Z_p, where each coefficient of the polynomials $A(x)$ and $B(x)$ correspond to an element of the vectors a and b, respectively; let $\tilde{a} = (a_0,\ldots,a_{n-1},0,\ldots,0)$ and $\tilde{b} = (b_0,\ldots,b_{n-1},0,\ldots 0)$ be the extended vectors of a and b of length $2n$, where the trailing components have been filled with zeros. Then, the product $a\times b = \mathrm{NTT}_\omega^{-1}(\mathrm{NTT}_\omega(\tilde{a})\circ\mathrm{NTT}_\omega(\tilde{b}))$ where $\circ$ meaning component-wise multiplication [3]. In this case, the product $a\times b$ should be reduced to the ring $R = Z_p/\langle x^n+1\rangle$.

Taking into account that the arithmetic operations are performed in the ring $R = Z_p/\langle x^n+1\rangle$, where $x^n + 1$ is the reduction function, the NTT-based polynomial multiplication can be calculated using the Negative Wrapped Convolution for $Z_p/\langle x^n+1\rangle$, such as is shown in Algorithm 1. This convolution is defined as follows: Let ω be a primitive n-th root of unity in Z_p and $\psi^2 = \omega$. Let $a = (a_0,\ldots,a_{n-1})$ and $b = (b_0,\ldots,b_{n-1})$ be vectors of length n with elements in Z_p. Let $c = (c_0,\ldots,c_{n-1})$ be the negative wrapped convolution of a and b. Let a', b' and c' be defined as $(a_0, \psi a_1,\ldots, \psi^{n-1}a_{n-1})$, $(b_0, \psi b_1,\ldots, \psi^{n-1}b_{n-1})$ and $(c_0, \psi c_1,\ldots, \psi^{n-1}c_{n-1})$. Then $c' = \mathrm{NTT}_\omega^{-1}(\mathrm{NTT}_\omega(a')\circ\mathrm{NTT}_\omega(b'))$ [3]. Taking into account the mentioned above, the NTT of each polynomial can be calculated using only n-coefficient of the polynomials. It is important to mention that the n-coefficient of each polynomial are multiplied by powers of ψ and the vector c' is multiplied by powers of ψ^{-1} to obtain the vector c. In this case, the polynomial product $c = a\times b \in R = Z_p/\langle x^n+1\rangle$ is directly obtained, i.e., it is not necessary to reduce c to the ring $R = Z_p/\langle x^n+1\rangle$, as in the Convolution Theorem case.

Algorithm 1: NTT-based polynomial multiplication using Negative Wrapped Convolution for $Z_p/\langle x^n+1\rangle$

Inputs: $a = (a_0,\ldots,a_{n-1})$, $b = (b_0,\ldots,b_{n-1})$
Pre-computations: $p\psi = (1,\psi,\ldots,\psi^{n-1})$, $p\psi^{-1} = (1, \psi^{-1},\ldots, \psi^{-(n-1)})$ where $\psi^2 = \omega$
Temporal variables: $A = (A_0,\ldots,A_{n-1})$, $B = (B_0,\ldots,B_{n-1})$, $C = (C_0,\ldots,C_{n-1})$, $a' = (a'_0,\ldots,a'_{n-1})$, $b' = (b'_0,\ldots,b'_{n-1})$, $c' = (c'_0,\ldots,c'_{n-1})$
Output: $c = (c_0,\ldots,c_{n-1})$

1. $a' := p\psi \circ a$;
2. $b' := p\psi \circ b$;
3. $A := \mathrm{NTT}(a')$;
4. $B := \mathrm{NTT}(b')$;
5. $C := A \circ B$;
6. $c' := \mathrm{NTT}^{-1}(C)$;
7. $c := c' \circ p\psi^{-1}$;
8. **Return** (c);

IV. Design of Polynomial Multiplier Based on NTT

A. Design of the n-point NTT-core

In order to design the n-point NTT-core, we have analyzed two hardware architectures used for FFT design. The first one is a fully parallel architecture based on decimation in frequency. Figure 1 shows an 8-point FFT using a parallel architecture [12].

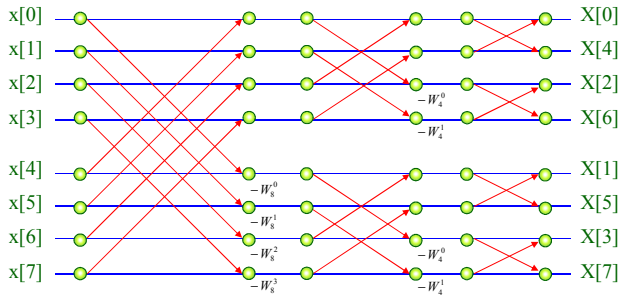

Fig. 1. 8-point FFT based on decimation-in-frequency.

The second one is a pipeline architecture, where each stage has a Radix-2 Single-path Delay Feedback (R2SDF) block, which uses feedback shift registers to store the output of the butterfly block (BF) [12]. The R2SDF block and the pipeline architecture are shown in Figures 2 and 3, respectively.

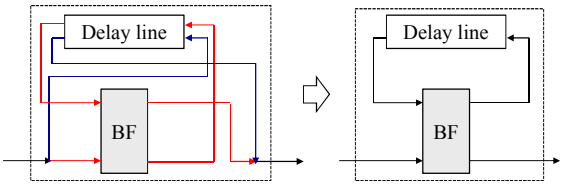

Fig. 2. R2SDF block for FFT.

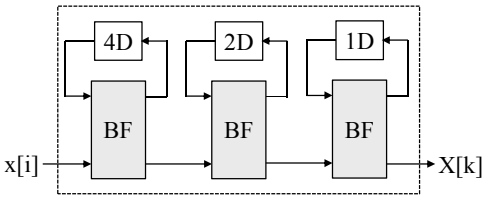

Fig. 3. Pipeline architecture for 8-point FFT.

The parallel architecture is not suitable for the hardware implementation of the n-point NTT-core due to a secure lattice-based cryptosystem uses polynomials of degree greater than or equal to 511. Then, in order to address the above issue, we design an n-point NTT-core mapping Algorithm 2 into a systolic array, where the PE is a modified R2SDF block that has a butterfly block $B\omega$.

The butterfly block $B\omega$ is shown in Figure 4, and it is designed using modular arithmetic and implemented using one integer adder, one integer subtractor, one ROM, one integer multiplier and three modular reduction blocks (MR). The MR block is designed using the algorithm presented in [13] due to the modulus p is a Fermat prime; the ω and ω^{-1} powers are stored in the ROM-Pω; and the results of addition, subtraction, multiplication and modular reduction are all registered.

Figure 5 shows the PE_i of the systolic array which is designed using one butterfly block $B\omega$, two delay registers, one register and one multiplexor. The delay registers *Del1* and *Del2* perform delays of $n/2^{(i+1)}$ -1 and $n/2^{(i+1)}$ clock cycles, respectively. In this case, m is the number of *PEs* and $m = \log_2 n$-1. Figure 6 shows the systolic array for an n-point NTT. The NTT-core has a latency of $n + 2\log_2 n$ clock cycles and it calculates the NTT using $2n + 2\log_2 n$ clock cycles.

Algorithm 2: Simple NTT pseudocode

Inputs: $n = 2^m$, $a = (a_0,\ldots,a_{n-1}) \in F_p^n$
Temporal variables: s and $t \in Z_n$, $w \in Z_p$
Output: $A = (A_0,\ldots,A_{n-1}) \in F_p^n$

1. **for** $i = 0$ **to** n-1 **do**
2. $A_i := a_i$;
3. **end for**
4. **for** $i = 0$ **to** m-1 **do**
5. **for** $j = 0$ **to** 2^i-1 **do**
6. **for** $k = 0$ **to** 2^{m-i-1}-1 **do**
7. $s := j2^{m-i} + k$
8. $t := j2^{m-i} + k + 2^{m-i-1}$
9. $w := \left(\omega^{2^i}\right)^k$
10. $\begin{bmatrix} A_s \\ A_t \end{bmatrix} := \begin{bmatrix} A_s + A_t \\ w(A_s - A_t) \end{bmatrix}$ (butterfly)
11. **end for**
12. **end for**
13. **end for**
14. **Return** (A);

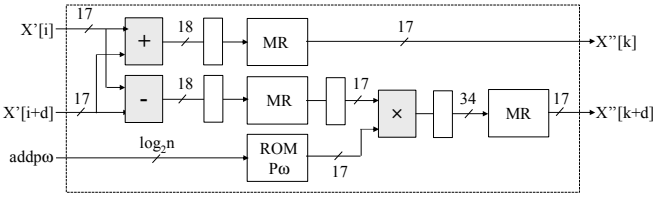

Fig. 4. Block diagram of the butterfly block Bω.

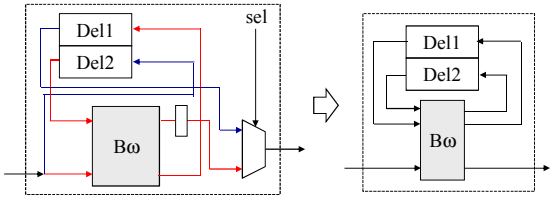

Fig. 5. Block diagram of the PE_i.

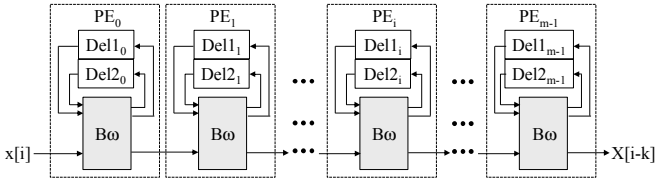

Fig. 6. Systolic array of the n-point NTT-core

B. *Design of NTT-based Polynomial Multiplier*

Figure 7 shows the n-coefficient NTT-based polynomial multiplier, which is designed using one n-point NTT-core, three integer multipliers, three MR blocks, two ROMs, two RAMs and one multiplexer. In this case, the polynomial multiplication is calculated using the following procedure:

1. Calculate the modular product between each element a_i of a and each power of ψ (a'), and calculate the NTT of the above products (NTT(a')). This multiplication is stored in RAM-A.
2. Calculate the modular product between each element b_i of b and each power of ψ (b'), and calculate the NTT of the above products (NTT(b')).

3. Multiply the results obtained in the previous steps, i.e., the point-wise product of NTT(*a'*) and NTT(*b'*). These products are stored in RAM-C.
4. Calculate the partial NTT^{-1} of the above point-wise product, i.e., the multiplication by n^{-1} of the Equation (3) is not carried out.
5. Calculate each element c_i of c by multiplying (modular product) the results of the previous step and the products $\psi^{-i} \times n^{-1}$, which are in the ROM-Pow_iψ.

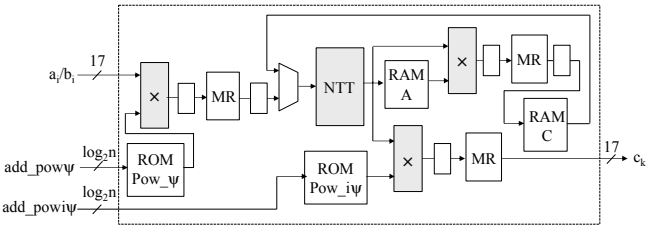

Fig. 7. Block diagram of the NTT-based polynomial multiplier.

Finally, it is important to take into account that the NTT-based polynomial multiplier concurrently carries out some of the above calculations. Figure 8 shows the number of clock cycles used to carry out the above integer arithmetic operations. From Figure 8, it is possible to deduce that the number of cycles for calculating the polynomial product is given by Equation (4).

$$NC_{prod} = 5n + 4\log_2 n + 5 \qquad (4)$$

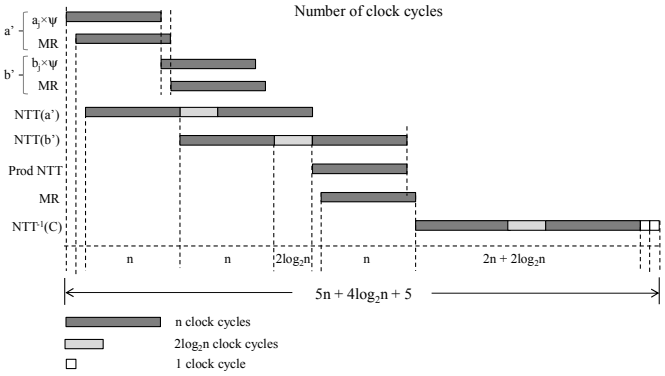

Fig. 8. Number of clock cycles required for the polynomial product.

V. Synthesis and Performance Results

The *n*-coefficient NTT-based polynomial multiplier is described using generic structural VHDL, verified using Modelsim and synthesized on the Stratix EP4SGX230KF40C2. Table I presents the parameters used for designing the *n*-point NTT-core.

TABLE I. NTT Parameters

n	p	ω
512	$2^{16}+1$	628
1024	$2^{16}+1$	1111
2048	$2^{16}+1$	2070
4096	$2^{16}+1$	7105

Table II presents the synthesis results of the *n*-point NTT-core, where *n* = 512, 1024, 2048 and 4096. From these results, it is possible to observe that the NTT-core uses relatively few area resources and has a high operation frequency.

TABLE II. Synthesis Results for NTT-core

n	ALUTs	Memory ALUTs	Block memory bits	Registers	DSP block 18-bit	Fmax (MHz)
512	1921	126	39060	1752	18	249.19
1024	2065	128	85104	1878	18	236.57
2048	2174	110	177228	1967	18	232.45
4096	2263	74	361512	2038	18	232.07

Table III presents the synthesis results for NTT-based polynomial multiplier for processing polynomials of degree 511, 1023, 2047 and 4095. From these results, it is possible to observe that the NTT-based polynomial multiplier uses relatively few area resources and has a high operation frequency.

TABLE III. Synthesis Results for NTT-based Polynomial Multiplier

n	ALUTs	Memory ALUTs	Block memory bits	Registers	DSP block 18-bit	Fmax (MHz)
512	2064	126	66708	1819	22	239.64
1024	2210	128	140400	1949	22	235.68
2048	2323	110	287820	2042	22	228.99
4096	2415	74	582696	2117	22	214.68

Table IV presents the performance results for the NTT-based polynomial multiplier. From these results, it is possible to observe that the polynomial product is carried out in short time, of order of μs.

TABLE IV. Performance Results for NTT-based Polynomial Multiplier

n	Number of cycles	Time (μs)	Mult/s
512	2601	10.85	92166
1024	5165	21.91	45641
2048	10289	44.93	22257
4096	20533	95.64	10455

Table V presents the execution times for the designed *n*-point NTT-core and the hardware implementation presented in [3]. The execution times are calculated from the synthesis results of both polynomial multipliers, i.e., $t_{exec} = \frac{clock\ cycles}{operation\ frequency}$. From these results, it is possible to observe that the execution time of the designed NTT-core is 2.7 times less than the execution time of the other one.

TABLE V. COMPARISON WITH 512-POINT NTT IMPLEMENTATIONS FROM POLYNOMIAL MULTIPLIERS

Reference	Cycles NTT	Cycles INTT	Fmax (MHz)	t_{NTT}(µs)	t_{INTT}(µs)
[3]	2304	2304	196	11.75	11.75
This work	1042	1044	239.64	4.35	4.36

Table VI presents the synthesis and performance results for the designed NTT-based polynomial multiplier and other ones presented in [3] and [8]. From this table, it is possible to observe that the execution time of the designed NTT-based polynomial multiplier is five times less than the execution time of the other ones.

TABLE VI. COMPARISON WITH 512-POINT NTT-BASED POLYNOMIAL MULTIPLIER IMPLEMENTATIONS

Reference	Area	Registers	Memory	Fmax (MHz)	Cycles	t_{mult} (µs)
[3]	1585 LUTs	1205 FFs	4 BRAMs	196	10014	51.09
[8]	240 Slices	-	0.8 BRAMs	-	12800	51.00
This work	2064 ALUTs	1819 Regs	66708 Block memory bits	239.64	2601	10.85

VI. CONCLUSIONS AND FUTURE WORK

This work presents the design of a parameterizable *n*-coefficient polynomial multiplier based on an *n*-point NTT-core, which is designed using a systolic array. In this case, the polynomial multiplier is designed for multiplying polynomials of degree 511, 1023, 2047 and 4095. The NTT-based polynomial multiplier was described using generic structural VHDL, synthesized on the Stratix EP4SGX230KF40C2 using Quartus II V. 13, and verified using Modelsim. The synthesis results show that the NTT-based polynomial multiplier uses relatively few area resources, and the execution time for calculating the polynomial product is minimal, of order of µs. Therefore, taking into account the synthesis and performance results, we conclude that the designed polynomial multiplier is suitable for hardware implementation of lattice-based cryptosystems, and to the best of our knowledge the *n*-point NTT-core is the first one based on a systolic array. Then, our parameterizable *n*-coefficient polynomial multiplier allows achieving a core faster than the previous ones presented in the literature.

The future work will be oriented to design an *n*-point NTT-core using high-radix butterfly blocks; to design a lattice-based cryptoprocessor; and to design a large integer multiplier using the Strassen algorithm and *n*-point NTT-core for fully-homomorphic encryption.

ACKNOWLEDGMENT

Claudia P. Rentería-Mejía thanks Colciencias for the scholarship.

REFERENCES

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in Annual Symp. on foundations of computer science, Santa Fe, NM, 1994, pp. 124-134.

[2] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-Quantum Cryptography*, Chicago: D. Bernstein, J. Buchmann and E. Dahmen, 2009, pp. 147-191.

[3] T. Pöppelmann and T. Güneysu, "Toward efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in Proc. LATINCRYPT, Santiago de Chile, Chile, 2012, pp. 139-158.

[4] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations", [online] November 15 2013, http://eprint.iacr.org/2006/224.pdf

[5] C. P. Rentería-Mejía, A. López-Parrado and J. Velasco-Medina, "Hardware Design of FFT Polynomial Multipliers," in Proc. IEEE LASCAS, Santiago de Chile, Chile, 2014, pp. 1-4.

[6] L. Sing, A. Mir and T. Hin, "Efficient FPGA implementation of FFT based multiplier," in Canadian conf. on electrical and computer engineering, Saskatoon, 2005, p.p. 1300-1303.

[7] N. Göttert, T. Feller, M. Schneider, J. Buchmann and S. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in CHES, Leuven, Belgium, 2012, pp. 512-529.

[8] A. Aysu, C. Patterson and P. Schaumont, "Low-cost and area-efficient FPGA implementations of lattice-based cryptography," in HOST, 2013, p.p. 81-86.

[9] M. Schneider, "Sieving for shortest vectors in ideal lattices," in Proc. AFRICACRYPT, Cairo, Egypt, 2013, pp. 375-391.

[10] J. van de Pol, "Lattice-based cryptography," M.Sc. Thesis, Dep. Of Math. And Comp. Sc., Eindhoven Univ. of Techn., Eindhoven, Netherlands, 2011.

[11] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations", [online] November 15 2013, http://eprint.iacr.org/2006/224.pdf

[12] S. He and M. Torkelson, "A new approach to pipeline FFT processor", [online] June 11 2014, http://ipdps.cc.gatech.edu/1996/PAPERS/S19/HE/HE.PDF

[13] G. Xiaoxu, R. Cheun, C. Kaya and K. Fung, "Reconfigurable number theoretic transform architectures for cryptographic applications," in Proc. FPT, Beijing, 2010, pp. 308-311.