Communications and Network, **2013, 5, 485-492**
http://dx.doi.org/10.4236/cn.2013.53B2089 Published Online September 2013 (http://www.scirp.org/journal/cn)

Efficient Implementations of NTRU in Wireless Network

Xin Zhan, Rui Zhang, Zhilong Xiong, Zhaoxia Zheng, Zhenglin Liu
School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan, China
Email: liuzhenglin@hust.edu.cn

Received April 2013

ABSTRACT

NTRU is a lattice-based public key cryptosystem featuring reasonably short, easily created keys, high speed, and low memory requirements, seems viable for wireless network. This paper presents two optimized designs based on the enhanced NTRU algorithm. One is a light-weight and fast NTRU core, it performs encryption only. This work has a gate-count of 1175 gates and a power consumption of 1.51 μW. It can finish the whole encryption process in 1498 μs at 500 kHz. As such, it is perfect for wireless sensor network. Another high-speed NTRU core is capable of both encryption and decryption, with delays of 16,064 μs and 128,010 μs in encryption and decryption respectively. Moreover, it consists of 25,758 equivalent gates and has a total power consumption of 59.2 μW (it will be reduced greatly if low power methods were adopted). This core is recommended to be used in base stations or servers in wireless network.

Keywords: Wireless Network; Public key Cryptosystem; NTRU; Efficient Implementations

1. Introduction

The wireless network is being expected to be widely used in various fields. Although wireless network offers low-cost deployment and convenience, security implementation is considered essential because of its inherent vulnerability. The public key cryptosystem using two different keys, such as RSA and ECC, has profound consequences in the areas of confidentiality, key distribution and authentication, but its common conception is complex, slow and power hungry [1]. The relatively new lattice-based public key cryptosystem—NTRU that features reasonably short, easily created keys, high speed, and low memory requirements [2], seems viable for wireless network.

O'Rourke first implemented the NTRU core with a gate count of minimum 1483 gates (N = 503) [3]; but it performs star multiplication only. Another design is realized in Kaps's paper (N = 167) [4]; it performs only encryption with an area of 2850 gates and power consumption of 15.1 μW at 500 kHz. The most detailed low-cost implementation of NTRU is realized by AC Atici's [5]. The author presented two compact NTRU architectures (N = 167), one is for encryption only with an area of 2800 gates and a dynamic power consumption of 1.72 μW at 500 kHz. Another is capable of both encryption and decryption, and it consists of 10,500 gates and consumes 6 μW dynamic power. Several other papers focus on the optimization of area and power consumption, but only a few works are related to studies on the optimization of speed.

In Jeffrey Hoffstein's paper [6], an alternative of NTRU is proposed with a new form $f = 1 + pF$ to create private key f that speeds both the key generation and the decryption. Unfortunately, there is little research on the implementation of the enhanced NTRU.

The rest of this paper is organized as follows: Section 2 summarizes the basic of enhanced NTRU algorithm; Section 3 presents the architecture of an encryption-only enhanced NTRU optimized for small area and fast speed; in Section 4, an architecture of enhanced NTRU optimized for high speed is presented, it is capable of both encryption and decryption; in Section 5 we give synthesis results on an ASIC technology library; Section 6 concludes this paper.

2. Algorithm of Enhanced NTRU

NTRU is a parameterized family of lattice-based public key cryptosystems. Its basic operations are realized in a truncated polynomial ring $R = Z[X]/(X^N - 1)$. Polynomials in the ring have a degree of N-1 and all the coefficients are integers. Polynomial $a \in R$ can be presented as:

$$a = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1} = \sum_{i=0}^{N-1} a_i X^i \quad (1)$$

The addition in the ring R is just as the same as general polynomial addition. But the multiplication in R is referred to star multiplication (see [2] for details) and denoted as a * symbol.

Copyright © 2013 SciRes.

The NTRU cryptosystem depends on three parameters, where a large prime N limits the degree of polynomials in the ring and (p, q) satisfies $\gcd(p, q) = 1$. It is also possible to consider p as a polynomial, for example $p = 2 + X$.

In order to speed up both the key generation and the decryption process, a new form of enhanced NTRU is suggested [6]. In enhanced NTRU, users choose f to have the form $f = 1 + pF$. Notice that this form has the property of $f_p = f^{-1} \pmod p = 1$, so it is not necessary to compute the inverse modulo p, and the second multiplication in the decryption process disappears. Some other optimizations presented in the rest of this paper are all based on enhanced NTRU.

Before we give an outline of enhanced NTRU algorithm, four sets of binary polynomials F, g, r, m need to be specified first. Some notations are used to specify the sets of polynomials:

R The set of binary polynomials in $R = Z[X]/(X^N - 1)$.

$R(d)$ The set of binary polynomials in R with d ones and N-d zeros.

Then the polynomials F, g, r, m are selected in the specified set: $F \in R(d_F)$, $g \in R(d_g)$, $r \in R(d_r)$, $m \in R(d_m)$.

- **Key Generation:** the user must first choose two random polynomials $F \in R(d_F)$ and $g \in R(d_g)$, private key is computed as:

$$f = 1 + pF. \tag{2}$$

  Besides, $f_q = f^{-1} \pmod q$ must exist. Then the following computation is performed:

$$h \equiv pf_q * g \pmod q. \tag{3}$$

  Now h is the public key and f is the private key.
- **Encryption:** To encrypt a plaintext $m \in R(d_m)$, the user should select a random polynomial $r \in R(d_r)$ at first. Then the cipher e is formed:

$$e \equiv h * r + m \pmod q. \tag{4}$$

- **Decryption:** In order to decrypt the cipher e using private key f, the user first calculates:

$$a \equiv f * e \pmod q. \tag{5}$$

  Then the user chooses $a \in R$ to satisfy this congruence and to lie in a pre-specified subset of R. Finally, a binary polynomial m should be found out to satisfy

$$m(-2) = a(-2) \pmod{2^N + 1}. \tag{6}$$

  It can be proved that m equals the plaintext.

3. Light-Weight and Fast NTRU (Encryption-Only)

In order to meet the requirements in ultra-low cost environments like wireless sensor network, we proposed a light-weight NTRU structure based on the enhanced NTRU algorithm. It performs encryption only and is optimized for both fast speed and small area. The parameter set we have chosen is $(N, q) = (107, 64)$, and $d_r = 5$, which was the lowest security recommended in [2].

Figure 1 shows the architecture of light-weight NTRU engine. It consists of a control engine, a 6 bits result buffer, a multiplication module, a look up table (LUT) and a non-zero coefficients sequence generator (NCSG).

The control engine manages the process of encryption. The result buffer is used to store final result and current sum in star multiplication process. The multiplication module performs star multiplication operation. Public key h is pre-computed and stored in the LUT. NCSG is designed to generate and rotate the degrees of non-zero terms of random polynomial r.

Due to NCSG, one operand in multiplication operation is bounded to 1'b1 (see Section 3.1 below), the multiplier is needless and multiplication module only consists of a 6bits adder and a router.

3.1. Non-Zero Coefficients Sequence Generator (NCSG)

In NTRU encryption process, the computation of $h * r$ is not time-efficient. In fact, r is a quite sparse binary polynomial that has only d_r non-zero coefficients. As a result, most of the multiplications in a star multiplication process are unnecessary.

In this paper, we introduce a structure of non-zero coefficients sequence generator (NCSG), which could record the degrees of non-zero terms in polynomial r when the coefficients of r loaded one by one. Then the non-zero coefficients sequence is rotated during the computation of star multiplication. According to this, the control engine generates the corresponding address of h for LUT. As **Figure 2** shows, NCSG consist of a $7d_r$ bits circular register, a 7 bits counter, a 7 bits adder, a 2-input router, an AND gate and an OR gate.

Input of the right hand 7-bit register is composed of 2 paths and determined by a *ctrl* signal. *Path*1 is from the most significant 7 bits during the computation of star multiplication, the register performs as a general shift register and the degree of current non-zero term is output to control engine by left-hand 7 bits register. During loading polynomial r, the output of the 7 bits counter is as one source of *path*2, it is added to the least significant 7 bits (where the degree of current term is stored). The counter counts the zero from r_in and resets if a one is input. Then the degree of next term is computed by the adder and loaded to the circular register. After N clocks, non-zero coefficients sequence is generated in the circular register.

Copyright © 2013 SciRes.

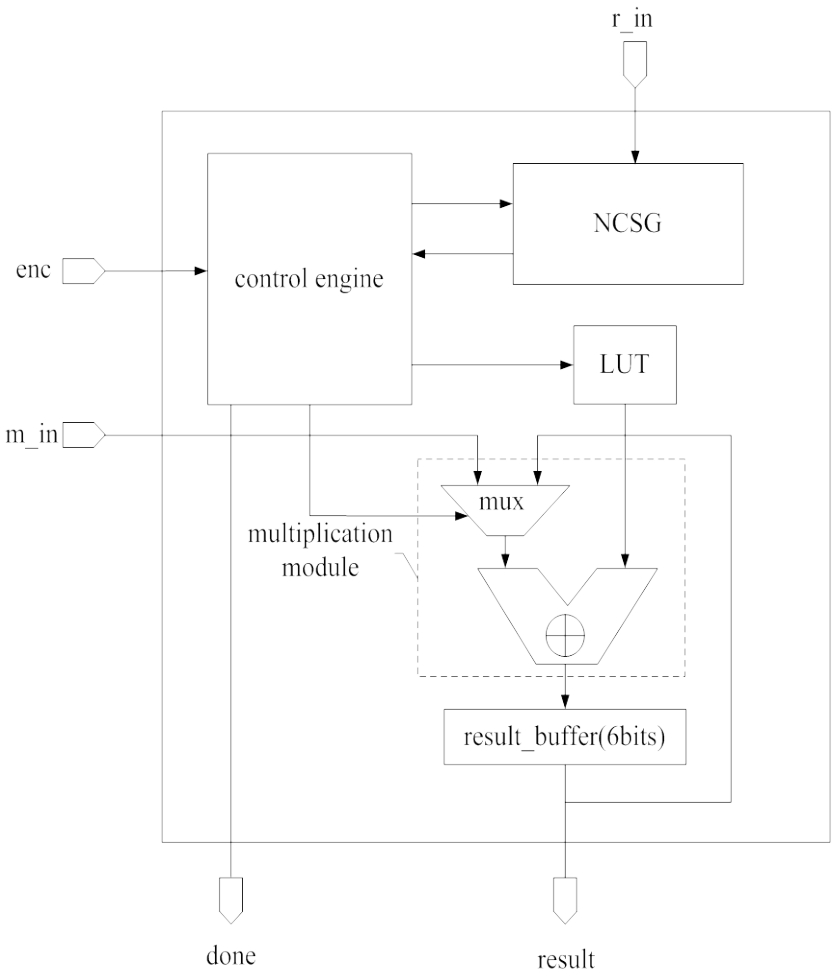

Figure1. light-weight NTRU architecture.

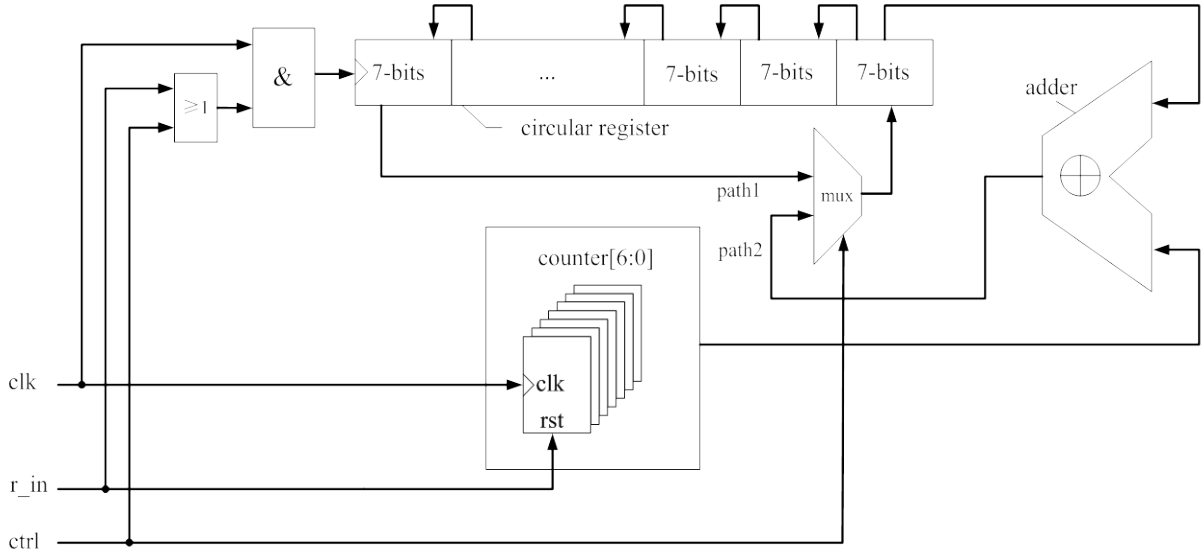

Figure 2. The NCSG architecture.

Copyright © 2013 SciRes.

The clock of circular register is a gated clock based on ADD gate. During a star multiplication process *ctrl* equals 0 and r_in remains high voltage, the clock is always enabled. During the loading stage where *ctrl* equals 1, the clock of circular register is enabled only when a one from r_in is detected.

Due to NCSG, the consumption of time to compute $h*r$ is reduced to $d_r \times N$ clock cycles. In addition, one operand to multiplication module is 1 constantly. As a result, the hardware cost for storing polynomial r is saved.

3.2. Control Engine

Control engine is the controller of the NTRU engine and designed with a 4-state finite state machine (FSM), which initialed with an *idle* state.

When a valid *enc* signal is detected, the encryption process starts and the FSM enters a *load* state, during which the coefficients of polynomial *r* are loaded one by one to NCSG. *N* clock cycles later the degrees of non-zero terms are generated in NCSG. Then FSM transits to *multiplication* state and begins to calculate the first coefficient of $h*r$, this process spends d_r clock cycles. *Multiplication* is followed by an *add* state, where plaintext *m* is added to the current sum. At this time, *e*'s first coefficient is calculated and the control engine outputs a *done* signal. After the addition of the message, the FSM again transmits to *multiplication* to compute the second coefficients of *e*. When the last coefficient of *e* is calculated, the FSM returns to *idle* state and then the encryption process is finished.

4. High-Speed NTRU

The performance gain of enhanced NTRU comes from elimination of an inversion modulo *p* in key generation and a star multiplication in decryption. On this basis, a high-speed NTRU engine is presented to further speed up the encryption process through improving the efficiency of star multiplication. The high-speed NTRU can be used in wireless networks that structured around base stations and centralized servers, which do not have the limitations associated with small portable devices.

Figure 3 shows the architecture of the high-speed NTRU. For decryption, a 7N bits e buffer is used to store

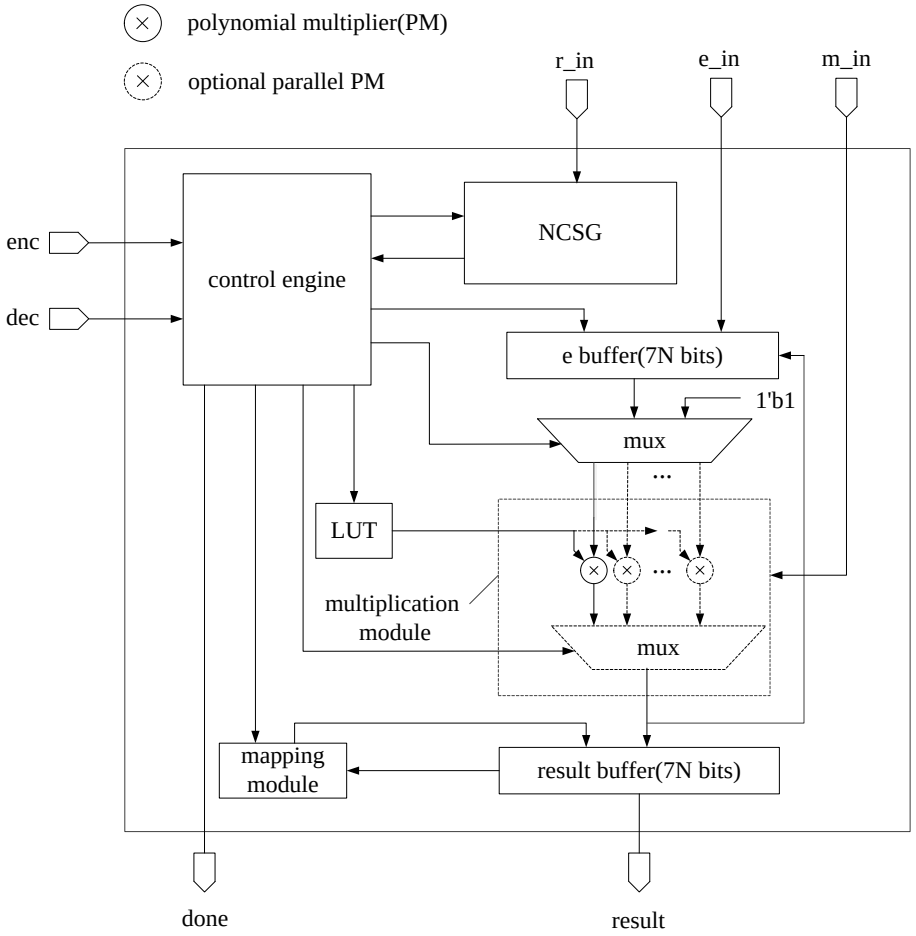

Figure 3. High-speed NTRU architecture.

Copyright © 2013 SciRes.

cipher *e* and another 7N bits result buffer is for the final result. Multiplication module and mapping module perform the main arithmetic function, the former is for star multiplication (parallel polynomial multiplier is optional to further speed up the process) and the latter is used to recover plaintext *m* from the intermediate variable *a*. Moreover, LUT stores both *f* and *h*.

In our design, we have chosen $(N, q) = (251, 128)$, $p = 2 + X$, $d_F = d_g = d_r = 72$ and $d_m = 125$. This parameter set of NTRU is considered with high level of security and extremely low decryption failure probability [8].

4.1. Mapping Module

After we calculated $a = f * e (\mathrm{mod}\, q)$ in the decryption process, we need to recover binary plaintext *m* from *a*. The Binary polynomial *m* satisfies Equation [6].

According to [6], the transformation in the mapping algorithm can be summarized as:

$$(a_i, a_{i+1}, a_{i+2}) \to (u, a_{i+1} + v - 2w, a_{i+2} + v - w), \quad (7)$$

where *u*, *v* and *w* are obtained as:

$$\begin{cases} u = a_i[0]; \\ v = \{1'b0, a_i[6:1]\}; \\ w = \min\{\lfloor (a_{i+1} + v)/2 \rfloor, a_{i+2} + v\}. \end{cases} \quad (8)$$

We denote this transformation as below:

$$(a_i, a_{i+1}, a_{i+2}) \to \mathrm{Algorithm}(a_i, a_{i+1}, a_{i+2}). \quad (9)$$

The mapping module can be implemented in a completely combinational way, it performs the following function:

$$(x_o, y_o, z_o) = \begin{cases} (x_i, y_i, z_i), i \ge N \,\&\, x_i \notin \{0,1\}; \\ \mathrm{Algorithm}(x_i, y_i, z_i), otherwise, \end{cases} \quad (10)$$

where *i* is the value of a 7 bits counter increasing every clock cycle till *i* equals 2*N*.

The connection of the mapping module and the result buffer is shown in **Figure 4**. When the control signal *sel* SASA equals zero, a mapping operation is executed and x_o is connected to the input of the right-hand 7 bits register; when *sel* equals 1, the result of star multiplication is loaded to the result buffer and *data* is as the input to the circular register.

According to the above description, the entire mapping process takes 2*N* clock cycles totally.

4.2. Small Hamming Weight Product

To further decrease the consumption of time in computation of the product $h * r$, an alternative form is suggested that takes advantage of sparse polynomials [6,7]. The suggested form is: $r = r1 * r2 + r3$, where the three sparse binary polynomials $r1$, $r2$ and $r3$ have d_{r1}, d_{r2} and d_{r3} non-zero coefficients (one in this case) respectively. When it satisfies $d_{r1}d_{r2} + d_{r3} = d_r$, the constructed polynomial r has approximately d_r ones (occasional non-binary coefficients are mixed in with the ones and zeros, but don't affect matters very much). Then the computation of $h * r$ is divided to three steps by $h * r1$, $(h * r1) * r2$ and $h * r3$. When combined with NCSG mentioned previously, the operation times of computing $h * r$ are reduced to $(d_{r1} + d_{r2} + d_{r3})N$ clock cycles.

For the case $d_r = 72$, we have chosen $d_{r1} = d_{r2} = d_{r3} = 8$. As the result, the efficiency of star multiplication in NTRU encryption tripled without changing the level of security.

4.3. Control Engine

Control engine in this design has 6 states and begins with an *idle* state.

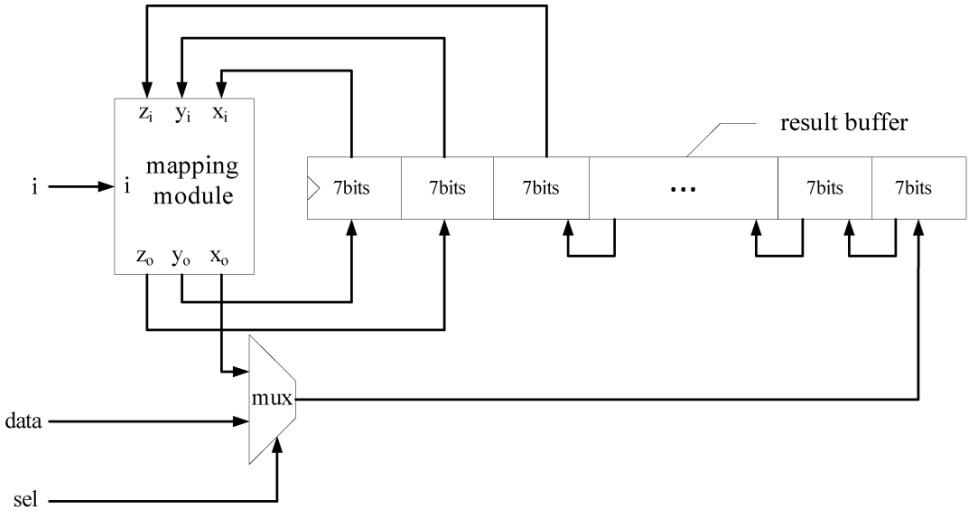

Figure 4. The mapping unit.

Copyright © 2013 SciRes.

On detection of a high signal of *enc*, the encryption process starts and the FSM enters a *load* state. We divide the whole encryption process into three steps, as shown in **Figure 5**.

- **Step1**: computing $\alpha = h * r1$. During *load* state, the coefficients of $r1$ are loaded one by one to NCSG. After *N* clock cycles the degrees of non-zero terms are generated in NCSG. FSM transits to *multiplication* state to calculate the first coefficient of α, this process takes d_{r1} clock cycles, it should be noted that one multiplier is constant "1" during this state. The following state is *result*, during which the first coefficient of α is loaded to the e buffer. Then FSM returns to *multiplication* state to compute α's second coefficient and stores it in *result* state. After repeats this process *N* times, all coefficients of intermediate variable α are calculated and stored in e buffer, then **Step1** is finished and FSM enters **Step2**. **Step1** costs $(d_{r1}+1)N + N$ clock cycles totally.
- **Step2**: computing $\beta = \alpha * r2 + m$. **Step2** starts with *load* state, during which the coefficients of $r2$ are loaded one by one. Then FSM transits to *multiplication* state and begins to calculate the first coefficient of $\alpha * r2$, this process spends d_{r2} clock cycles. *Multiplication* is followed by *add* state, where plaintext *m* is added to current sum. At this time, β's first coefficient is calculated, it is loaded to the result buffer in *result* state. After this, FSM begins to compute and store the second coefficients of β. When the last coefficient of β is stored in the result buffer, **Step3** is followed. **Step2** takes $(d_{r2}+2)N + N$ clock cycles.
- **Step3**: computing $e = h * r3 + \beta$. Like in **Step1** and **Step2**, polynomial $r3$ is loaded in *load* state first. The following state is *multiplication* and the computation of $h * r3$'s first coefficient starts. After this FSM transmits to *add* state, β is added to calculate the first coefficient of cipher *e*. In *result* state, the result is loaded to the result buffer. After repeats *N* times, the whole coefficients of *e* are stored in the result buffer. Finally control engine outputs a *done* signal and the FSM back to *idle* state. **Step3** takes $(d_{r3}+2)N + N$ clock cycles.

So the whole process of encryption requires a total of $N(d_{r1}+d_{r2}+d_{r3}+8)$ clock cycles.

When *dec* signal is valid, NTRU engine performs decryption operation. Firstly, the FSM transmits to *load* state, coefficients of cipher *e* are loaded to the e buffer one by one. During the following *multiplication* state, the first coefficient of a is calculated; this state takes *N* clock cycles. Then FSM transmits to *result* state and a's first coefficient is loaded into result buffer. After this FSM begins to compute and store the second coefficients of a. When the last coefficient of a is computed, FSM enters *map* state, the plaintext binary polynomial *m* is recovered from a in 2*N* clock cycles. Finally, control engine outputs a *done* signal indicating the end of decryption. The decryption process takes a total of $N(N + 4)$ clock cycles.

5. Implementation Results

In this paper, our designs were implemented with Verilog language and synthesized by Synopsys Design Compiler with a clock frequency of 500 kHz. The targeted ASIC technology library we used is the HJTC 0.18 μm standard cell library. We have also implemented AC Atici's work [5] for comparison.

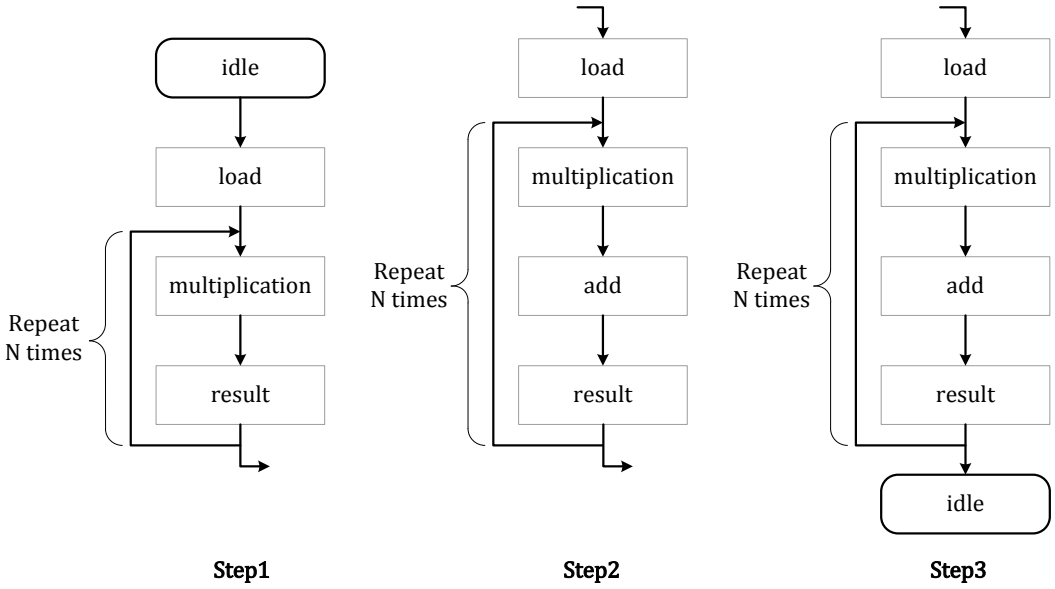

Figure 5. Control sequence in encryption process.

Copyright © 2013 SciRes.

We have first synthesized two encryption-only NTRUs with the same parameters set $(N, q) = (107, 64)$ and $d_r = 5$, one is the implementation of AC Atici's scheme and another is our light-weight NTRU core.

Table 1 reveals that the proposed implementation of the low-weight NTRU has a significant decrease in both encryption delay and area. The encryption delay is found to be 1498 μs, which is only 6.4% of AC Atici's design. This was expected since we took full advantage of sparse polynomial *r* in our design. We can also see that the area is much smaller than the contrast, only with 1175 equivalent gates.

As shown in **Table 2**, the decrease in area is mainly due to the needless of the *r* buffer, which is used to store and rotate polynomial *r* with a consumption of 1243 gates, while the addition module—NCSG consumes only 466 gates. We also got a total power consumption of 1.51 μW from the result of Synopsys Design Compiler.

We have also synthesized our high-speed NTRU core, which can perform both encryption and decryption function. We chose $(N, q) = (251, 128)$ and $d_F = d_g = d_r = 72$ as the parameters. For comparison, the delay and area results of AC Atici's encryption-decryption NTRU are also listed.

Table 3 shows that the high-speed NTRU can finish the process of encryption and decryption in 16,064 μs and 128,010 μs, respectively, which gains much in speed performance, especially in encryption process. However, the area is 25,758 gates and the total power consumption is 59.2 μW. The high-speed NTRU could be used in base stations and servers of wireless network.

Table 1. Delay and area results of encryption-only NTRUs.

Work	Enc. delay (*μs*)	Area (*eqv.gates*)
AC Atici's design	23,336	1949
Our Low-weight NTRU	1498	1175

Table 2. Area consumption of encryption-only NTRUs.

Block	Area (*eqv.gates*)	
	AC Atici's design	Low-weight NTRU
control logic	382	429
lut	236	236
multiplication module	88	44
r buffer	1243	-
NCSG	-	466
total	1949	1175

Table 3. Delay and area results of encryption-decryption NTRUs.

Work	Enc. delay (*μs*)	Dec. delay (*μs*)	Area (*eqv.gates*)
AC Atici's design	127,508	263,550	14,441
Our High-speed NTRU	16,064	128,010	25,758

By the way, as power consumption is not the target we optimized for in this paper, low power methods such as clock gating and operand isolation are not used in our designs. If these methods are adopted, the power consumption will be greatly reduced.

6. Conclusions

In this paper we presented two hardware architectures of NTRU. The first one capable of encryption only is optimized for small area and fast speed, has a gate-count of 1,175 gates and a total power consumption of 1.51 μW. This NTRU core can finish the encryption process in 1498 μs at 500 kHz. It is very suitable for use in ultra-low cost environment such as wireless portable devices. Another one is designed for high speed with delays of 16,064 μs and 128,010 μs in encryption and decryption respectively, obtaining significant gains when compared with the original NTRU that provides the same level of security. This circuit consists of 25,758 equivalent gates and has a total power consumption of 59.2 μW. So the high-speed encryption-decryption NTRU core is recommended to be used in wireless base stations and servers.

Besides, the designs can be sped up by using parallel polynomial multiplier units.

7. Acknowledgements

This work was supported in part by National Natural Science Foundation of China (Grant No. 60973034, No. 61006020 and No. 61176026).

REFERENCES

[1] G. Gaubatz, J. P. Kaps and B. Sunar, "Public Key Cryptography in Sensor Networks—Revisited," 1*st European Workshop on Security in Ad-Hoc and Sensor Networks*, 2005, pp. 2-18

[2] J. Hoffstein, J. Pipher and J. H. Silverman, "NTRU: A Ring-Base Public Key Cryptosystem," In: J. P. Buhler, Ed., *Algorithmic Number Theory* (*ANTS III*), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1998, pp. 267-288,.

[3] C. M. O'Rourke, "Efficient NTRU Implementations," Master Thesis, Worcester Polytechnic Institute, Worcester, 2002.

[4] J. Kaps, "Cryptography for Ultra-Low Power Devices," Ph.D. Thesis, Worcester Polytechnic Institute, Worcester, 2006.

[5] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede and S. B. O. Yalcin, "Low-Cost Implementations of NTRU for Pervasive Security," *International Conference on Application-Specific Systems, Architectures and Processors*, 2008. pp. 79-84.

[6] J. Hoffstein and J. Silverman, "Optimizations for NTRU," *Public-Key Cryptography and Computational Number*

Copyright © 2013 SciRes.

Theory: *Proceedings of the International Conference organized by the Stefan Banach International Mathematical Center Warsaw*, Poland, September 11-15, 2000, p. 77.

[7] J. Hoffstein and J. H. Silverman, "Random Small Hamming Weight Products with Applications to Cryptography," *Discrete Applied Mathematics*, Vol. 130, No. 1, 2003, pp. 37-49.

[8] J. H. Silverman and W. Whyte, "Estimating Decryption Failure Probabilities for NTRUencrypt," *Technical Report, NTRU Cryptosystems*, 2005, Technical Report No. 18, Version 1.

Copyright © 2013 SciRes.