Efficient and Flexible Low-Power NTT for Lattice-Based Cryptography

Tim Fritzmann* and Johanna Sepúlveda*
*Technische Universität München, Munich, Germany, {tim.fritzmann, johanna.sepulveda}@tum.de

***Abstract*—Secure communication is being threatened by the foreseeable breakthrough of quantum computers. When a larger quantum computer is developed, traditional public key cryptography will be broken. Lattice-based cryptography appears as an alternative to protect the communications in the era of quantum computers. However, empowering current electronic devices with these new algorithms poses a challenging problem due to tight performance requirements as well as area and power constraints. Polynomial multiplication is the basic and most computationally intensive operation in lattice-based cryptosystems. The Number Theoretic Transform (NTT) is an attractive technique to perform polynomial multiplication efficiently. So far, previous works have focused on developing fast and compact forward and inverse NTT implementations. However, efficient and low-power NTT design has not been considered before although a low power consumption is crucial for many systems, such as battery-powered Internet of Things (IoT) devices. In this paper, we present the first low-power, fast and secure NTT ASIC design for lattice-based cryptography able to support different NTT parameters. The contribution of this work is three-fold. First, the implementation of a fast NTT through three optimization techniques. Second, utilization of methods for ASIC power minimization in the NTT design. Third, review of previously proposed side-channel attacks and discussion about countermeasures for our design. Our proposed architecture requires only $n \log(n)$ clock cycles for the forward and inverse NTT and can be implemented using a cheap single port RAM. The results of our work show that it is possible to decrease the power dissipation by more than 30 % at nearly no cost.**

***Index Terms*—Lattice-based Cryptography, Number Theoretic Transform, Low-Power Design**

I. Introduction

Public-Key Cryptography (PKC) is the foundation for establishing secure communication between multiple parties. Traditional PKC algorithms such as the *Rivest–Shamir–Adleman* (RSA) cryptosystem [1], which is based on the factorization of large numbers, or *Elliptic Curve Cryptography* (ECC) [2], which is based on the discrete logarithm problem, are considered insecure to attacks performed by a quantum computer.

The foreseeable breakthrough of quantum computers therefore represents a risk for all communication systems. These computers will be able to solve in polynomial time the mathematical problems on which classical PKC (RSA, ECC) relies on by executing Shor's quantum algorithm [3]. The skyrocketing evolution of quantum computers, in particular, poses a significant threat for applications with long lifecycles where deployed devices are hard to update (e.g., cars, airplanes and satellites). As a reaction, the *National Security Agency* (NSA) announced in 2016 the intention to transition towards post-quantum cryptography for governmental usage in the foreseeable future [4]. In the same manner, *National Institute of Standards and Technology* (NIST) started in 2017 the process of post-quantum cryptography standardization [5]. The goal is to select a set of post-quantum algorithms able to meet the security, performance and cost requirements of current and future applications.

Post-quantum cryptography (PQC) appears as an alternative to protect communication channels against the possible quantum threat. PQC refers to a set of cryptographic algorithms based on mathematical problems that are resistant against known attacks using quantum computers, but by themselves can be implemented using classical computing platforms. Among all the types of post-quantum algorithms available today, lattice-based cryptography, based on computationally hard problems in lattices, offers a very good trade-off between security and efficiency [6].

Practical lattice-based cryptographic schemes are instantiated using large polynomial rings defined as $\mathcal{R} = \mathbb{Z}_q/\langle x^n+1\rangle$ with integers n and q. Polynomial addition and multiplication are the two basic operations required to encrypt, decrypt, sign and verify information. While polynomial addition can be performed with $\mathcal{O}(n)$ operations, a naive polynomial multiplication requires $\mathcal{O}(n^2)$ operations. Thus, for large dimensions of n, multiplication becomes very expensive in terms of execution time and power. Currently, polynomial multiplication is still the performance bottleneck of lattice-based cryptography. Previous works have shown that the Number Theoretic Transform (NTT) is an efficient alternative to ease the multiplication of a pair of large polynomials. The NTT is a generalization of the Fast Fourier Transform (FFT) carried out in finite fields. That is, the NTT transforms the pair of polynomials into the spectral domain, where it is possible to point-wise multiply the two vectors. The point-wise multiplication in the spectral domain can be efficiently carried out through a hardware or software solution. Then, the result vector is transformed to the normal domain through the inverse transform (NTT^{-1}). By using the NTT, the multiplication of a pair of polynomials of degree $n-1$ can be carried out through the execution of $\mathcal{O}(n \log n)$ operations.

The NTT design is critical, since it is often the performance bottleneck and the most resource consuming block of the whole lattice-based cryptographic design. While the goal of previous NTT software and hardware implementations was the reduction of the execution time, none of the previous approaches have considered the design of a power-optimized NTT. The optimization of the NTT is mandatory, since it

978-1-5386-8064-3/19/$31.00

is required to make lattice-based post-quantum cryptography feasible and practical.

Contribution. In this work, we propose for the first time a fast, low-power and secure NTT ASIC design. Our NTT architecture is flexible and supports several NTT configuration parameters (n, q, ω_n). In summary, our contributions are:

- Fast NTT architecture that reduces the number of clock cycles through three techniques: i) rescheduling operations into stall cycles; ii) integration of an efficient implementation of the Barrett and Montgomery modulo reduction into the NTT; and iii) avoidance of extra cycles for the inverse NTT post-processing;
- Utilization of clock gating and operand isolation techniques for achieving a low-power NTT ASIC design;
- Review of previously proposed Side-Channel Attacks (SCA) and countermeasures for our NTT ASIC.

The remainder of this article is organized as follows: Section II describes the main concepts of post-quantum cryptography. Section III presents previous works on NTT hardware implementations. Section IV describes the concept of the NTT multiplication. The NTT optimizations on the algorithmic level and our hardware architecture are presented in Section V and Section VI, respectively. Section VII presents the power optimizations and Section VIII discusses the side-channel security analysis. Section IX presents the evaluation results. A conclusion is given in Section X.

II. Post-Quantum Cryptography

Most of the lattice-based post-quantum cryptosystems are built upon the Learning With Errors (LWE) problem and its practical variant—the Ring-Learning With Errors (R-LWE) problem. The R-LWE problem was introduced by Lyubashevsky *et al.* in [7]. Solving the R-LWE problem is equivalent to solving the NP-hard approximate Shortest Vector Problem (SVP) in a regular lattice. Due to its practicality and the difficulty to solve the R-LWE problem—even with a quantum computer—it has been largely employed in modern public-key cryptography.

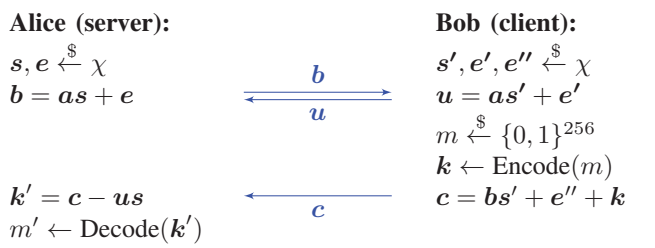
The R-LWE problem uses arithmetic operations with polynomials in the ring $\mathcal{R} = \mathbb{Z}_q/\langle x^n + 1\rangle$. The polynomials of this ring are in the form

$$\boldsymbol{a} = a_0 + a_1 x + a_2 x^2 + \ldots a_{n-1} x^{n-1} \ , \tag{1}$$

where $n-1$ is the degree of the polynomial. Each coefficient a_i, with $i = 0, 1, \ldots, n-1$, is an element of a finite field and reduced by the prime integer q (modulo reduction). The addition of two polynomials is a simple operation, where the coefficients of the polynomials are point-wise added together:

$$\boldsymbol{a}+\boldsymbol{b} = (a_0+b_0)+(a_1+b_1)x+\ldots(a_{n-1}+b_{n-1})x^{n-1} \ . \tag{2}$$

The polynomial multiplication is considerably more complex. After the multiplication, a modulo reduction with $x^n + 1$ is performed in order to remain in the ring $\mathcal{R}$. Thus, the result of the polynomial multiplication has a degree $n-1$.



Protocol 1. Fundamental principles of R-LWE key encapsulation. All polynomials in $\mathcal{R}$ are printed in bold. Non-bold letters, here m and m', are used for usual bit-arrays. Let $\xleftarrow{\$}$ denote the sampling process. Polynomial a is public.

A direct approach to calculate the coefficients c_i of $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b} \mod (x^n + 1)$ is

$$c_i = \sum_{j=0}^{i} a_j b_{(i-j)} - \sum_{j=i+1}^{n-1} a_j b_{(n+i-j)} \ . \tag{3}$$

Due to the reduction with $x^n + 1$, all coefficients with degree greater than $n-1$ are negatively wrapped back, which is represented in the latter part of Eq. 3. This direct approach requires $\mathcal{O}(n^2)$ operations. For this reason, the NTT, which is discussed in Section IV, is used.

A standard R-LWE instance consists of one polynomial multiplication and one polynomial addition. It has the form

$$\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e} \ , \tag{4}$$

where $\boldsymbol{a}$ is a public polynomial, $\boldsymbol{s}$ a secret polynomial and $\boldsymbol{e}$ an error polynomial. The difficulty of the R-LWE problem is to recover the secret $\boldsymbol{s}$ from the equation. Usually, $\boldsymbol{a}$ is sampled from a uniform distribution U_q with outcomes between 0 and $q-1$, and $\boldsymbol{s}$ and $\boldsymbol{e}$ are sampled from an error distribution χ (mostly binomial or Gaussian distribution), where sampling means to take a random value from a set S. With an increasing error term, the difficulty for an attacker to recover the secret increases, too.

Protocol 1 illustrates the fundamental principles of lattice-based cryptography, in this example, by means of a key encapsulation scheme. In the first step, Alice and Bob sample the secret polynomials $\boldsymbol{s}$ and $\boldsymbol{s}'$, and the error polynomials $\boldsymbol{e}$, $\boldsymbol{e}'$ and $\boldsymbol{e}''$. Then, Alice and Bob create the R-LWE instances $\boldsymbol{b}$ and $\boldsymbol{u}$, respectively. The created instances are sent to the other party. Further, Bob samples 256 bits from a random number generator and assigns them to the secret key vector m. The bit-vector is then encoded into the polynomial $\boldsymbol{k}$. Afterwards, another R-LWE instance is created and the secret key polynomial $\boldsymbol{k}$ is hidden in $\boldsymbol{c}$. Now, polynomial $\boldsymbol{c}$ can be securely sent to Alice. The largest noise terms can be removed by subtracting $\boldsymbol{u}\boldsymbol{s}$ from $\boldsymbol{c}$. After the decoding, Alice and Bob agree with a certain probability on the same shared key (m' and m).

III. Related Work

The NTT has been widely exploited in different signal processing applications, such as FIR filters, image filtering or homomorphic image enhancement. In contrast to these applications, the NTT for lattice-based cryptography is used to transform longer vectors and requires an additional modulo reduction (mod $(x^n + 1)$).

Different NTT implementations for lattice-based cryptography have been proposed before. While software NTT implementations offer programming capability, hardware implementations provide a further speedup for cryptographic algorithms. Two goals have driven the NTT hardware design: i) fast NTT, such as the work of [8], which uses a fully parallel design at the expense of high area and power consumption (for large polynomials this method becomes impractical); and ii) compact NTT, as presented in the works of [9], [10], [11], where a memory is used to store three information: i) NTT input, containing the vector that should be transformed; ii) NTT output, which corresponds to the vector in the spectral domain; and iii) partial or complete values of Twiddle factors (powers of ω_n in Eq. 5). The work of [9] stores the complete Twiddle factors together with the NTT input and output in the memory. As an improvement, in [10] the Twiddle factors were calculated during runtime, thus only a set of these factors are stored. This proposal reduces the required memory size for storing the Twiddle factors to $\log(n)$. To speedup this proposal, the work of [11] suggested a smart memory access scheme.

In contrast to previous works, our proposal further optimizes the proposal of [11] by using three techniques: i) reduction of clock cycles by rescheduling operations into stall cycles when single port RAMs are used; ii) integration of efficient modular addition and multiplication reduction techniques (Barrett and Montgomery reduction, respectively); and iii) integration of the post-processing $(\frac{1}{n} \cdot \gamma^{-i})$ of Eq. 8 in the main algorithm, thus further speeding up the computation. Moreover, we present the first NTT ASIC design for lattice-based cryptography. Our ASIC design performs fast NTT transformations and utilizes clock gating and operand isolation low-power design techniques. Different lattice-based algorithms require different NTT configurations. Table I shows the NTT parameters of some lattice-based post-quantum algorithms submitted to NIST. Our hardware design offers high flexibility and simple parameter changes.

TABLE I
NTT parameters of some NIST submissions

Algorithm	**Dimension** (n)	**Modulo** (q)
CRYSTALS-Dilithium	256	8380417
Falcon	512/1024	12289
qTesla	1024/2048	8058881/12681217
CRYSTALS-KYBER	256	7681
Ding Key Exchange	512/1024	120833
NewHope	512/1024	12289
R.EMBLEM	512	12289
Hila5	1024	12289
KCL	256	7681
Titanium	1024/2048	86017/1198081

IV. NTT: Number Theoretic Transform

The polynomial multiplication is the bottleneck of lattice-based post-quantum cryptography. However, this operation becomes very simple in the spectral domain. It corresponds to a point-wise multiplication of the coefficients of a pair of vectors. The Discrete Fourier Transform (DFT) is the general approach to transform a polynomial a from the normal domain into the spectral domain. The coefficients of a are transformed as in Eq. 5

$$\hat{a}_i = \sum_{j=0}^{n-1} \omega_n^{ij} \cdot a_j \ , \tag{5}$$

where $n-1$ is the degree of the polynomial and ω_n the n-th root of unity. In order to transform the result vector from the spectral domain into the normal domain, the inverse must be applied as in Eq. 6

$$a_i = \frac{1}{n} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j \ . \tag{6}$$

The DFT has a complexity of $\mathcal{O}(n^2)$. To speed up the transformation, the Fast Fourier Transform (FFT) can be used. It exploits the symmetric structure of the Fourier transform by splitting one DFT into smaller DFTs.

The NTT is a FTT where ω_n is an element of a finite ring instead of the complex numbers. The use of the NTT has two main advantages. First, the multiplication between a pair of polynomials a and b is faster, by requiring only $\mathcal{O}(n \log n)$ operations. The polynomial multiplication is calculated as $c = \text{NTT}^{-1}(\text{NTT}(a) \circ \text{NTT}(b))$, where $\circ$ denotes a point-wise multiplication and NTT^{-1} is the inverse NTT. Second, the operations in the ring $\mathcal{R} = \mathbb{Z}_q/\langle x^n+1 \rangle$ require a reduction of a, b and c by $x^n + 1$. The reduction in the spectral domain simply corresponds to a scaling of the coefficients a_i and b_i with the factor γ_n^i (before the NTT) and the coefficients of c_i with γ_n^{-i} (after the NTT^{-1}), where $\gamma_n = \sqrt{\omega_n}$. Hence, Eq. 5 and Eq. 6 are modified to Eq. 7 (NTT with pre-processing) and Eq. 8 (NTT^{-1} with post-processing), respectively:

$$\hat{a}_i = \sum_{j=0}^{n-1} \gamma^j \cdot \omega_n^{ij} \cdot a_j \quad \text{and} \tag{7}$$

$$a_i = \frac{1}{n} \cdot \gamma^{-i} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j \ . \tag{8}$$

Algorithm 1 presents the Cooley–Tukey technique to perform the NTT as in [11]. It breaks one DFT of size n into two smaller DFTs of size $n/2$. Equation 5 is split into a sum with *even* indices and a sum with *odd* indices. The core of the algorithm is a small DFT unit, which is often referred to as *butterfly unit*. Note that the Cooley–Tukey algorithm requires a bit reversal at the input to obtain a normal ordered output.

The coefficients of polynomial a are stored in a bit reversed

Algorithm 1: NTT transform

Input: Coefficients a_i, with $i = 0, 1, \ldots, n-1$, pre-calculated values of ω_m
Result: Coefficients $\hat{a}_i$, with $i = 0, 1, \ldots, n-1$

```
1  a ← BitReversal(a)
2  index ← 0
3  for m = 2 to n/2 by m = 2m do
4      ω_m ← Omega_LUT(index)
5      index ← index + 1
6      ω ← Omega_LUT(index) or 1 for NTT^-1
7      for j = 0 to m/2 − 1 by 1 do
8          for k = 0 to n/2 − 1 by m do
9              t_1, u_1 ← a_{k+j+m/2}, a_{k+j}   // MEM_{k+j}
10             t_2, u_2 ← a_{k+m+j+m/2}, a_{k+m+j}   // MEM_{k+j+m/2}
11             t_1 ← t_1 · ω
12             t_2 ← t_2 · ω
13             a_{k+j+m/2}, a_{k+j} ← u_1 − t_1, u_1 + t_1
14             a_{k+m+j+m/2}, a_{k+m+j} ← u_2 − t_2, u_2 + t_2
15             MEM_{k+j} ← a_{k+j+m}, a_{k+j}
16             MEM_{k+j+m/2} ← a_{k+j+3m/2}, a_{k+j+m/2}
17         end
18         ω ← ω · ω_m
19     end
20 end
21 m ← n
22 ω_m ← Omega_LUT(index)
23 ω ← Omega_LUT(index+1) or 1 for NTT^-1
24 for j = 0 to m/2 − 1 by 1 do
25     t_1, u_1 ← a_{j+m/2}, a_j   // MEM_j
26     t_1 ← t_1 · ω
27     a_{j+m/2}, a_j ← u_1 − t_1, u_1 + t_1
28     MEM_j ← a_{j+m/2}, a_j
29     ω ← ω · ω_m
30 end
```

order in the memory (line 1). As the coefficients can be expressed with at most 16-bits, the work of [11] suggested to store two coefficients in a single 32-bit memory line. In our implementation, all even coefficients are stored in the lower 16-bits and all odd coefficients in the higher 16-bits. Then, the value of ω_m is loaded from a small look-up table (LUT) (line 4), which stores only $\log(n)$ pre-calculated values. They are determined by $\omega_m = \omega_n^{n/m}$, where m ranges from 2 to n and doubles at every iteration. The Twiddle factors (ω) are calculated at runtime. This avoids large LUTs and keeps the NTT flexible, as in case of a parameter modification, only $\log(n)$ pre-calculated values must be updated. To avoid pre-processing, ω is initialized with $\gamma_m = \sqrt{\omega_m}$. In fact, the same LUT, as for ω_m, can be used as $\gamma_m =$ Omega_LUT(index+1).

Lines 3 to 20 present the *core of the algorithm* for the first $\log(n) - 1$ rounds. In lines 9 to 10, the coefficients of the memory location $k+j$ and $k+j+m/2$ are loaded and stored in the temporary variables t_1, t_2, u_1 and u_2. After performing the butterfly operation, the results are written back into the memory locations $k+j$ and $k+j+m/2$. The value of ω is always updated in lines 18 and 29. The last round (lines 24 to 30) requires a set of modifications in order to store the results in the correct memory locations.

V. Optimizations of the NTT algorithm

In order to further speed up the execution of the NTT, in this work we propose three different optimizations; i) reduction of clock cycles during the last round; ii) integration of efficient modular reduction techniques into the core algorithm; and iii) avoidance of post-processing for the inverse NTT.

A. Reducing number of clock cycles in the last round

The performance bottleneck of the NTT algorithm is the memory access. One possibility to increase the throughput is to use a dual-port RAM. It allows multiple reads and writes at the same time. However, the use of single port RAMs is usually preferred in ASIC designs because they are less costly and do not require large arbitration logic. Therefore, we propose in the following paragraphs optimization methods that are applicable for designs with single port RAMs as well. Suppose that a memory access requires one clock cycle. Then, each inner loop of the first $\log(n) - 1$ rounds (line 9 to 16) requires four clock cycles: two for reading and two for writing. In the last round, the inner loop (line 25 to 29) requires only two clock cycles: one for reading and one for writing. If the same functional blocks are used for the first and the last rounds, two inactive cycles at each iteration of the last round are provoked. To avoid this situation, at each iteration the coefficients of memory location j and $j+1$ are loaded. This technique saves $n/4$ clock cycles.

B. Efficient Modular Reduction

The modulo reduction is usually a complex operation that requires hardware dividers. However, the use of a constant modulo reduction in the NTT algorithm allows to integrate efficient reduction techniques. The Barrett reduction (Algorithm 2) and the Montgomery reduction (Algorithm 3) are used to perform the modulo addition and modulo multiplication, respectively.

Algorithm 2: Barrett modulo reduction

Input: Integer a, constants m and k
Result: Integer $a \mod q$

```
1  u = (a · m) ≫ k
2  u = u · q
3  a = a − u
4  if q ≤ a then
5      a = a − q
6  end
```

In the Barrett reduction (*barrett*), the parameter q guides the selection of the parameters m and k. Due to some approximations, the algorithm has an error $e = 1/q - m/2^k$. As long as the input is smaller than $1/e$ the algorithm works correctly. For the modulo multiplication, the input a in Algorithm 2 is usually too high to allow an error free Barrett reduction.

To realize the modulo multiplication, the Montgomery reduction (*mont*), shown in Algorithm 3, can be used as it allows a larger input range for a. At least one of the input operands

Algorithm 3: Montgomery modulo reduction

Input: Integer a in Montgomery representation, constants R, $\log(R)$ and $q' = -q^{-1} \mod R$
Result: Integer $a \mod q$

1 $u = a \cdot q'$
2 $u = u \wedge ((1 \ll \log(R)) - 1)$
3 $a = a + (u \cdot q)$
4 $a = a \gg \log(R)$
5 **if** $q \leq a$ **then**
6 $\quad a = a - q$
7 **end**

(or the product itself) must be in the Montgomery domain. The Montgomery representation of a is simply aR. The output of the algorithm is scaled with R^{-1}. The constant R is chosen such that the modulo reduction by this constant is easy, e.g., a power of 2. Further, the condition $\text{GCD}(q, R) = 1$ must hold. The input a that has to be reduced is restricted to values in the range of $[0, Rq - 1]$.

In contrast to the work of [12], the values of ω are computed during run-time, avoiding the use of a large LUT. In order to use the Montgomery reduction, ω_m is stored in the Montgomery representation $\omega_n^{n/m} \cdot R \mod q$. Similarly, the values of ω_m^{-1} for the inverse transform are pre-calculated according to $\omega_n^{-n/m} \cdot R \mod q$. The application of the Montgomery and Barrett reduction in the NTT algorithm is illustrated in Algorithm 4. In order to avoid negative inputs at the Barrett reduction, the value q is added during the subtraction. The optimized last round will be discussed separately in Subsection V-C.

Algorithm 4: Optimized NTT transform (first rounds)

Input: Coefficients a_i, with $i = 0, 1, \ldots, n-1$, pre-calculated values of ω_m in Montgomery domain
Result: Coefficients $\hat{a}_i$, with $i = 0, 1, \ldots, n-1$

1 $a \leftarrow \text{BitReversal}(a)$
2 $index \leftarrow 0$
3 **for** $m = 2$ **to** $n/2$ **by** $m = 2m$ **do**
4 $\quad \omega_m \leftarrow$ Omega_LUT_Mont(index)
5 $\quad index \leftarrow index + 1$
6 $\quad \omega \leftarrow$ Omega_LUT_Mont(index) or $R \mod q$ for NTT^{-1}
7 $\quad$ **for** $j = 0$ **to** $m/2 - 1$ **by** 1 **do**
8 $\quad\quad$ **for** $k = 0$ **to** $n/2 - 1$ **by** m **do**
9 $\quad\quad\quad t_1, u_1 \leftarrow a_{k+j+m/2}, a_{k+j}$ // MEM_{k+j}
10 $\quad\quad\quad t_2, u_2 \leftarrow a_{k+m+j+m/2}, a_{k+m+j}$ // $\text{MEM}_{k+j+m/2}$
11 $\quad\quad\quad t_1 \leftarrow \text{mont}(t_1 \cdot \omega)$
12 $\quad\quad\quad t_2 \leftarrow \text{mont}(t_2 \cdot \omega)$
13 $\quad\quad\quad a_{k+j+m/2}, a_{k+j} \leftarrow$ barrett$(u_1 - t_1 + q)$, barrett$(u_1 + t_1)$
14 $\quad\quad\quad a_{k+m+j+m/2}, a_{k+m+j} \leftarrow$ barrett$(u_2 - t_2 + q)$, barrett$(u_2 + t_2)$
15 $\quad\quad\quad \text{MEM}_{k+j} \leftarrow a_{k+j+m}, a_{k+j}$
16 $\quad\quad\quad \text{MEM}_{k+j+m/2} \leftarrow a_{k+j+3m/2}, a_{k+j+m/2}$
17 $\quad\quad$ **end**
18 $\quad\quad \omega \leftarrow \text{mont}(\omega \cdot \omega_m)$
19 $\quad$ **end**
20 **end**

C. Avoiding Post-Processing for the Inverse Transform

The optimized last round is given in Algorithm 5. It avoids the post-processing (multiplications by n^{-1} and negative powers of γ) by integrating these operations into the last NTT^{-1} round. In line 3, ω is initialized with the pre-calculated LUT ($\gamma_m \cdot R \mod q$ for the NTT and $1 \cdot R \mod q$ for the NTT^{-1}). The scaling factors α_1, α_2, α_3 and α_4 are initialized with the pre-calculated values $n^{-1}R \mod q$, $n^{-1}\gamma_n^{-1}R \mod q$, $n^{-1}\gamma_n^{-n/2}R \mod q$ and $n^{-1}\gamma_n^{-n/2-1}R \mod q$, respectively. After loading the coefficients of the memory location j and $j+1$, the butterfly operation is performed in line 11 to 15. The modulo additions and multiplications are done with the Barrett and Montgomery reduction, respectively. To perform the post-processing of the NTT^{-1}, the coefficients of memory location j and $j+1$ are multiplied with α_1, α_2, α_3 and α_4. In line 19 and 20, the values of α_1, α_2, α_3 and α_4 are updated.

Algorithm 5: Optimized last round

1 $m \leftarrow n$
2 $\omega_m \leftarrow$ Omega_LUT_Mont(index)
3 $\omega \leftarrow$ Omega_LUT_Mont(index+1) or $R \mod q$ for NTT^{-1}
4 $\alpha_1 \leftarrow \gamma_1$ // forward: $\gamma_1 = 0$, inverse: $\gamma_1 = n^{-1}R$
5 $\alpha_2 \leftarrow \gamma_2$ // forward: $\gamma_2 = 0$, inverse: $\gamma_2 = n^{-1}\gamma_n^{-1}R$
6 $\alpha_3 \leftarrow \gamma_3$ // forward: $\gamma_3 = 0$, inverse: $\gamma_3 = \gamma_1\gamma_n^{-n/2}$
7 $\alpha_4 \leftarrow \gamma_4$ // forward: $\gamma_4 = 0$, inverse: $\gamma_4 = \gamma_2\gamma_n^{-n/2}$
8 **for** $j = 0$ **to** $m/2 - 1$ **by** 2 **do**
9 $\quad t_1, u_1 \leftarrow a_{j+m/2}, a_j$ // MEM_j
10 $\quad t_2, u_2 \leftarrow a_{j+m/2+1}, a_{j+1}$ // MEM_{j+1}
11 $\quad t_1 \leftarrow \text{mont}(t_1 \cdot \omega)$
12 $\quad \omega \leftarrow \text{mont}(\omega \cdot \omega_m)$
13 $\quad t_2 \leftarrow \text{mont}(t_2 \cdot \omega)$
14 $\quad a_{j+m/2}, a_j = \text{barrett}(u_1 - t_1 + q), \text{barrett}(u_1 + t_1)$
15 $\quad a_{j+m/2+1}, a_{j+1} = \text{barrett}(u_2 - t_2 + q), \text{barrett}(u_2 + t_2)$
16 $\quad$ **if** *Inverse NTT* **then**
17 $\quad\quad a_{j+m/2}, a_j \leftarrow \text{mont}(\alpha_3 a_{j+m/2}), \text{mont}(\alpha_1 a_j)$
18 $\quad\quad a_{j+m/2+1}, a_{j+1} \leftarrow \text{mont}(\alpha_4 a_{j+m/2+1}), \text{mont}(\alpha_2 a_{j+1})$
19 $\quad\quad \alpha_1, \alpha_2 \leftarrow \text{mont}(\alpha_1 \cdot \beta), \text{mont}(\alpha_2 \cdot \beta)$ // $\beta = \omega_n^{-1}R$
20 $\quad\quad \alpha_3, \alpha_4 \leftarrow \text{mont}(\alpha_3 \cdot \beta), \text{mont}(\alpha_4 \cdot \beta)$ // $\beta = \omega_n^{-1}R$
21 $\quad$ **end**
22 $\quad \text{MEM}_j \leftarrow a_{j+m/2}, a_j$
23 $\quad \text{MEM}_{j+1} \leftarrow a_{j+m/2+1}, a_{j+1}$
24 $\quad \omega \leftarrow \text{mont}(\omega \cdot \omega_m)$
25 **end**

VI. NTT Hardware Architecture

This section describes our proposed NTT hardware architecture. First, we provide a high-level description of the NTT module and then we discuss the different functional blocks of the design.

A. General Description

Figure 1 presents the input and output values of the NTT and NTT^{-1}. The NTT transforms the n coefficients of polynomial a into the spectral domain. The NTT^{-1} transforms the coefficients of the result vector $\hat{a}$ back into the normal domain. To keep the NTT flexible, the pre-calculated values required for the NTT and NTT^{-1} (discussed in Section V) are stored in a RAM. It includes: i) the parameters q, $\log(R)$, q', m

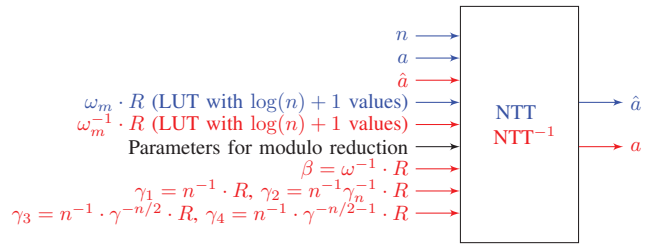


Fig. 1. NTT and NTT^{-1} input/output description. The signals of NTT are in blue and the signals of NTT^{-1} are in red.

and k for the modulo reduction; ii) $\log(n)+1$ values of ω_m (γ_m) or ω_m^{-1} (γ_m^{-1}) in the Montgomery domain; and iii) the pre-calculated scaling parameters $\beta, \gamma_1, \gamma_2, \gamma_3$ and γ_4 for the NTT^{-1}. Our NTT/NTT^{-1} is highly configurable. In order to support new NTT or NTT^{-1} parameters, the RAM is simply loaded with new values. Note that n must be a power of two and the memory large enough to store all coefficients. The bit length of the coefficients is set to be at maximum 16 bits, i.e., q has to be smaller than 2^{16}. Otherwise, a memory with 64 bit width has to be chosen.

B. Functionality

Figure 2 shows the proposed architecture of the forward transform (NTT) and the inverse transform (NTT^{-1}). It is based on the architecture of [11] and enhanced by the optimizations presented in Section IV. It is composed of four main modules: RAM, *Address Unit*, ω/α *Update Unit* and the core operation.

At the beginning of the NTT/NTT^{-1} operation, the RAM is initialized with the coefficients of the polynomial that should be transformed. For simplicity, this step is not shown in the Figure. The lower and higher 16 bits of a memory line are initialized with the even and the odd coefficients, respectively.

The control of the RAM read/write is performed by the *Address Unit*. It configures the read address to load the coefficients from the memory and the write address, together with the write enable signal, to store the result. Note that addresses during the first $\log(n)-1$ and last rounds are different.

The coefficients from the RAM are sequentially stored in registers. More specifically, the 16 lower and 16 higher bits of the memory line are stored in L1 and H1, respectively. First, the coefficients of memory location MEM$_{k+j}$ are loaded and processed (i.e., multiplication, addition and subtraction operations are performed). One clock cycle later, the memory location MEM$_{k+j+m/2}$ is loaded and processed. The first $\log(n)-1$ rounds require a swap of the coefficients (see line 15 and 16 Algorithm 1). Therefore, the registers R1, R2, R3, R4 and R5 are required to store the results and thus allow swapping. The data of R4 and R5 is written back to the memory location MEM$_{k+j}$ during the first write cycle. The data of R2 and R3 is written during the second write cycle into the memory location MEM$_{k+j+m/2}$. During the last round no swap is required and the results of R2 and R5 are written in the first and second write cycle into the memory location MEM$_j$ and MEM$_{j+1}$, respectively.

The additional components that are required for the NTT^{-1} are highlighted in red. It requires additional multipliers for realizing line 17 and 18 of Algorithm 5.

The ω/α Update Unit ensures that the variables ω and α_1, α_2, α_3 and α_4 are initialized and updated correctly. The initialization of ω is done as defined in line 6 in Algorithm 4 and line 3 in Algorithm 5. The initialization of α_1, α_2, α_3 and α_4 is done as defined in line 4 to 7 of Algorithm 5. In order to update ω to $\omega = \omega \cdot \omega_m$ the multiplier of the butterfly operation is reused. The scaling factors α_1, α_2, α_3 and α_4 are updated as described in line 19 and 20 of Algorithm 5.

VII. Power Optimization

To reduce the dynamic power consumption of the NTT, in this work we use two methods: clock gating and operand isolation [13].

A. Clock Gating

The clock signal has usually a very high fan-out and thus a high dynamic power consumption. Clock gating is an efficient method for synchronous circuits to reduce this power consumption. Usually, not all functional units in a design are required in each clock cycle. Therefore, the aim of clock gating is to turn on the clock signal only for those units that must be active. When registers are in idle mode, the dynamic power consumption is zero and only static power consumption (due to leakage currents) remains. The principle of clock gating is illustrated in Fig. 3. During the clock gating optimization process, registers that have a common enable signal are grouped together. Then, logic that is originally used to create the enable signal is removed and a common gated clock is created by adding logic to the clock tree. To avoid glitches on the gated clock signal, so called Integrated Clock Gating (ICG) cells instead of usual AND cells and latches are used. Due to the grouping of registers and removal of the enable logic, clock gating can also lead to a decrease of the total circuit size.

B. Operand Isolation

The second technique employed in this work to reduce the switching activity of the circuit is operand isolation. It blocks the signal propagation when it is not required for the functionality. This avoids unnecessary toggles and leads to a lower dynamic power consumption. The signal propagation is usually blocked with latches, AND gates or multiplexers. Although the area and static power consumption is due to additional gates slightly increased, this method can lead to a considerably improvement.

In our design, operand isolation was applied on the inputs of the arithmetic units, i.e., multiplier, adder and subtractor. The results of the first multiplier (H1$\cdot\omega$), the subtractor and the adder in Fig. 2 are required only in two clock cycles per iteration instead of four. As a result, we can block the input of the registers H1 and L1 in two out of four cycles. When the

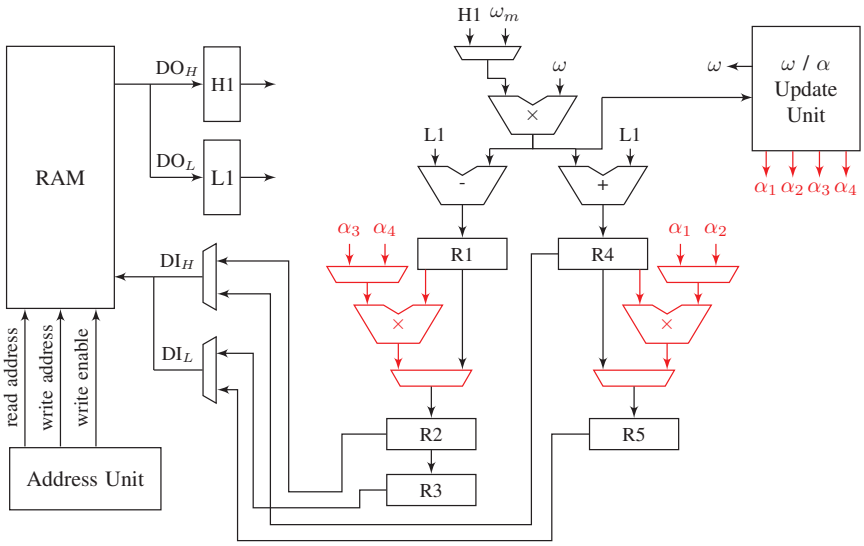


Fig. 2. NTT architecture, in red additional circuit for NTT^{-1}.

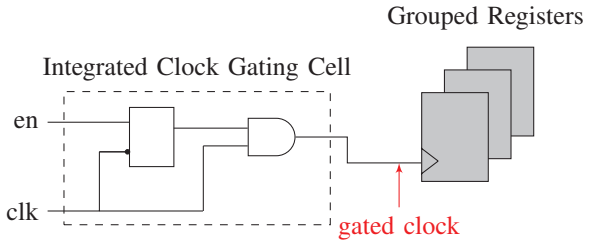


Fig. 3. Clock gating principle

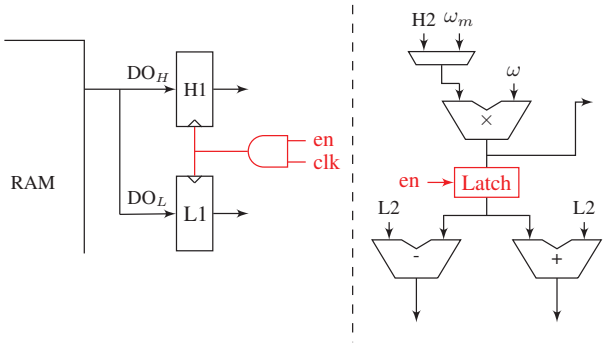


Fig. 4. Operand Isolation at input registers and after first multiplier. Required modifications are highlighted in red

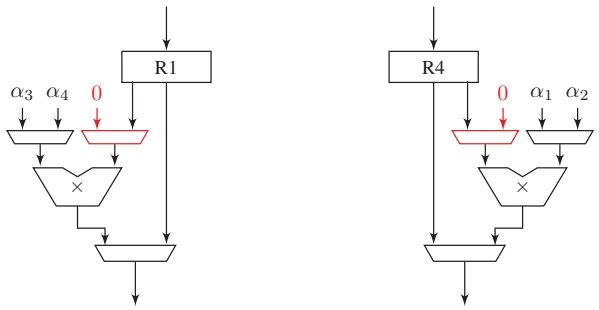


Fig. 5. Operand Isolation at multipliers for NTT^{-1}. Required modifications are highlighted in red

memory is in the write operation, it usually either outputs the value that is currently written or zero. Therefore, the old values of H1 and L1 must be preserved to avoid unnecessary toggles. The left part of Fig. 4 illustrates the required modifications. In order to block the input of H1 and L1 a clock gating cell is inserted. Thus, the registers are only updated when the enable signal is set. Alternatively, a multiplexer in front of the registers can be used that depending on the enable signal selects the new value from the memory or the old value from the register.

The first multiplier is also used to update the value of ω during the idle cycles. But the result of the multiplier during the update of ω is not required for the adder or subtractor. Therefore, a latch is added to the circuit as illustrated in the right part of Fig. 4. This latch forwards the result of the multiplier only when it is required for the subsequent subtractor and adder.

The multipliers used for the inverse transform have a large power dissipation as the input is always updated with new values from R1 and R4. As the output is only required at the last round of the inverse transform, the input can be set to zero in all other rounds, which is illustrated in Fig. 5. Moreover, it must be ensured that no glitches at α_1, α_2, α_3 and α_4 exist.

VIII. Side-Channel Security Analysis

SCA exploit physical leakages gained during the execution of a cryptographic algorithm (e.g., timing information, power consumption and electromagnetic radiation) in order to retrieve information about the secret. The NTT is a critical operation as in lattice-based cryptography secret polynomials are multiplied

with public polynomials. One attack point could be the point-wise multiplication of the transformed polynomials. Another possibility is trying to attack the NTT itself.

Prominent countermeasures that make SCA significantly harder are masking [14] and hiding [15]. In [14], the authors suggested a masking method that splits the secret polynomial into two shares such that $\boldsymbol{a} = \boldsymbol{a}' + \boldsymbol{a}'' \mod q$. One of these shares, e.g. $\boldsymbol{a}'$, is chosen uniformly at random. Now, the NTT is performed with the individual shares. After the point-wise multiplications and the NTT^{-1} the mask can be removed, i.e. the shares are combined together. In [15], the authors suggested to randomize the execution order of the point-wise multiplication. This method is also known as hiding. A successful attack on the NTT was proposed in [16]. The authors gain information about the secret polynomial from the modular arithmetic that is done during the butterfly operation. As implementation platform they used a Cortex-M4F and realized the modulo multiplication by calculating $a \mod q = a - q\lfloor a/q \rfloor$. They found out that the execution time of the integrated hardware divider depends on the bit size of the dividend, i.e. on a, which can be secret.

To avoid timing leakages, our proposal has a constant runtime. We use the Montgomery and Barrett reduction in order to avoid attacks on the divider as done in [16]. Moreover, all power optimizations that we perform are data independent. To harden our design against SCA, we can apply masking and hiding. These methods do not require any changes on the proposed hardware. Only modifications on algorithmic level are necessary.

IX. Experimental Results

The next subsections show the result of our FPGA and ASIC implementation. The FPGA implementation was used as basis and prototype for our ASIC design.

A. FPGA Implementation

Our FPGA hardware architecture was implemented on the Zedboard, which is equipped with a Xilinx Zynq-7000.

Lattice-based post-quantum cryptography is still a young area of research. At the moment, it is not clear which candidate or candidates of the NIST competition will be standardized. Moreover, it is still not clear which parameter sets will prevail. Previous experiences from cryptographic standardization processes have shown that it is also not unlikely that new attacks lead to the necessity of parameter changes. For these reasons, we designed a flexible NTT hardware architecture. We intentionally sacrifice area in order to achieve this flexibility. For example, the dimension n and the remaining parameters of Fig. 1 were designed flexible instead of using simple constants. In contrast to previous works, our implementation can be configured after synthesis.

Table II shows the utilization results of our work and related works. A parallel NTT design, as in [8], is only suitable for extreme high-throughput applications and low values of n. Even for a relatively low value of n, e.g. $n = 256$, the design requires a huge number of LUTs and registers.

The work of [9] has a much smaller utilization although the dimensions with $n = 512$ and $n = 1024$ are larger. However, the disadvantage of their approach is that it requires a lot of space for storing the Twiddle factors (ω^i and inverses) and the scaling factors (γ^i and inverses). In summary, a $3n\lceil \log q \rceil$ bit storage is required: $2n$ values for forward and inverse Twiddle factors and n values for the odd powers of γ and γ^{-1}.

In [10], the memory usage was considerably reduced as the authors suggest to calculate the Twiddle factors during run-time. The authors do not mention the exact utilization results, but a graph shows that for $n = 256$, $n = 512$ and $n = 1024$ the number of slices is between 200 and 300, the number of BRAMs is between one and two, and the number of DSPs is between two and three. The authors of [11] proposed an improved memory access scheme. Their architecture builds the basis of our work. Unfortunately, they only provide implementation results for the complete encryption scheme of [17], which includes the NTT but also a TRNG and a Gaussian sampler. However, we expect that our implementation is slightly larger than in [11] because we decided to increase the flexibility and we require additional multipliers for the inverse transform in order to increase the throughput. The increased utilization of DSP slices comes from the multiplications within the Barrett and Montgomery reduction of Algorithm 2 and Algorithm 3.

Our implementation requires $n \log\lceil q_{max} \rceil$ bits for storing the input/output coefficients and only $2 \log n \lceil q_{max} \rceil + 1$ bits for storing the powers of ω_m, ω_m^{-1} and γ_m ($\gamma_m^2 = \omega_m$), where $\log\lceil q_{max} \rceil$ is set to 16.

All optimization techniques described in this paper can also be applied in a design with fixed parameter set in order to decrease the required area. However, the resource utilization is still very low and it is advisable to keep the possibility of changing parameters.

B. ASIC Design and Power Measurements

The ASIC design was synthesized in the UMC 65 nm technology. In order to have low leakage currents, a low leakage library and a high threshold voltage were chosen. We use a moderate frequency of 25 MHz, a nominal supply voltage of 1.2 V and a temperature of 25°C. In order to get accurate results, we extract the dynamic post-layout power consumption by means of gate level switching activity files instead of RTL level switching activity files.

Table III summarizes the power consumption and Table IV the NTT area of the most common parameter sets of the lattice-based cryptography proposals submitted to NIST. The tables also include measurements with the low power optimization techniques: clock gating and operand isolation. The first test set has the parameters $n = 1024$, $q = 12289$ and $\omega_n = 49$; the second $n = 512$, $q = 12289$ and $\omega_n = 3$; and the third $n = 256$, $q = 7681$ and $\omega_n = 3844$.

The static power consumption does not depend on the parameter set because the same architecture is used for all configurations. Also, the dynamic power consumption does not significantly change for different parameters. However,

TABLE II
AREA USAGE FPGA IMPLEMENTATIONS

	LUTs	Registers	BRAMs	DSPs
Our Implementation	980 (2%)	395 (< 1%)	2 (1%)	26 (12%)
Works of [11] (n=256) [a]	1349	860	2	1
Works of [11] (n=512) [a]	1536	953	3	1
Works of [9] (n=512)	507	563	1.5	1
Works of [9] (n=1024)	529	576	3	1
Works of [8] (n=256) [b]	142481	191449	-	-

[a] Includes area of TRNG (378 LUTs and 9 FFs) and Gaussian sampler (around 164 LUTs).
[b] Parallel Design. No post-quantum application (no pre-/post-processing).

TABLE III
NTT POWER RESULTS

Variant	n	Component	P_{static} (nW)	P_{dyn} (nW)	P_{total} (nW)
Non Optimized	1024	NTT	19171	4405582	4424753
		NTT^{-1}	19170	4402071	4421241
	512	NTT	19171	4274549	4293720
		NTT^{-1}	19171	4317632	4336803
	256	NTT	19168	4664691	4683859
		NTT^{-1}	19168	4608740	4627908
Clock Gating and Operand Isolation	1024	NTT	19172	2920504	2939676
		NTT^{-1}	19172	2967375	2986547
	512	NTT	19173	2959503	2978676
		NTT^{-1}	19173	2978232	2997405
	256	NTT	19169	3075644	3094813
		NTT^{-1}	19169	3054241	3073410

TABLE IV
NTT AREA RESULTS

Variant	#Cells	$Area_{cell}$ (μm^2)	$Area_{net}$ (μm^2)	$Area_{total}$ (μm^2)
Non Optimized	14259	301459	28039	329497
Clock Gating and Operand Isolation	14352	301546	28052	329598

the energy consumption and the amount of required clock cycles highly depends on the parameter selection, in particular on parameter n. The timing results are discussed in Subsection IX-C.

The design contains in total 550 FFs, where only 171 FFs were suitable for clock gating because 379 FFs have no enable signal and three FFs have a too small register bank width. In our scenario, clock gating does not have a noticeable power reduction because only a relatively small amount of registers is used and most of the registers must be always active during the transform. However, when the whole cryptographic system is considered, clock gating is expected to save a lot of power as the NTT is not always active. The clock signal of the NTT unit can be disabled when the cryptographic system does not require transformations, e.g., during the sampling process of the secret and error terms or during the calculations of hash values. The exact amount of power savings depends on the cryptographic scheme and the time spent for different operations. We leave further analysis of this topic as future work and focus in the following only on the NTT operation itself. Due to the grouping of registers with common enable signal during the clock gating optimization process, the number of total cells were reduced from 14259 to 14207 and the corresponding cell area from 301459 μm^2 to 301294 μm^2.

In our scenario, operand isolation is much more effective. Only a negligible number of additional gates is required to significantly decrease the total power consumption. In com-

parison to the design with clock gating, the number of cells increased for the operand isolation method by 145. However, due to the blockage of unnecessary signal propagation, the switching activity of the system was significantly reduced. When taking both optimization methods into consideration, the total power consumption of the NTT can be decreased with the first, second and third parameter set by 33.56 %, 30.63 % and 33.93 %, respectively. The reductions in power consumption were achieved at nearly no costs.

C. Timing Results

When a single port RAM is used and one clock cycle for each memory access is assumed, the amount of required clock cycles for the NTT and NTT^{-1} is equal to $n\log(n)$ (plus 8 cycles latency). In contrast to previous works, no additional clock cycles are required for the post-processing, i.e. for multiplications of all coefficients with the powers of γ^{-1} and with n^{-1}, as it is integrated into the main algorithm. This makes the NTT computation complete and at least $2n$ clock cycles can be saved. Due to the changes of the last round in Algorithm 1, $n/4$ clock cycles were saved. Additionally, it is expected that with the efficient Barrett and Montgomery reduction a higher clock frequency can be used. As already discussed, a dual port RAM is more costly and has a higher area consumption. However, note that it can be twice as fast and might be preferable for some high-speed applications.

X. Conclusion

In this work, we proposed a fast, flexible and secure NTT ASIC design that was optimized for low power. We speed up the design by rescheduling operations, implementing efficient modulo reduction techniques and integrating post-processing into the core algorithm. The flexible structure of our design allows to use various cryptographic schemes, parameter changes to increase the security level and to ensure long term security, and the possibility to apply SCA countermeasures, such as masking and hiding. Moreover, the use of constant time modulo reduction techniques prevents previously proposed attacks. To decrease the dynamic power consumption, we applied the power optimization methods clock gating and operand isolation. In particular, operand isolation significantly decreased the power consumption of the computing-intensive NTT as unnecessary toggles in the arithmetic units are avoided. If a complete cryptographic design is considered, clock gating is expected to be very effective as well. During the idle cycles of the NTT unit, the clock for the NTT module can be turned off, which reduces the dynamic power consumption. The test results show that with the optimization methods more than 30 % of the overall power dissipation can be saved. Low power design is getting very important as the transistor density in modern chips increases. To cope with the increasing power dissipation, it is not only important to decrease the power consumption on transistor level, but also on the algorithmic and design level.

Acknowledgment

This work has been supported by the German-French Academy for the Industry of the Future (project ASSET).

References

[1] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, Feb. 1978.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[4] "NSA says it "must act now" against the quantum computing threat," https://www.technologyreview.com/s/600715/nsa-says-it-must-act-now-against-the-quantum-computing-threat/, 2016.

[5] National Institute of Standards and Technology, "Announcing request for nominations for public-key post-quantum cryptographic algorithms," 2016, https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms.

[6] O. M. Guillen, T. Pöppelmann, J. M. B. Mera, E. F. Bongenaar, G. Sigl, and J. Sepulveda, "Towards post-quantum security for IoT endpoints with NTRU," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 698–703.

[7] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, ser. Lecture Notes in Computer Science, H. Gilbert, Ed., vol. 6110. Springer, 2010, pp. 1–23. [Online]. Available: https://doi.org/10.1007/978-3-642-13190-5_1

[8] M. Dreschmann, J. Meyer, M. Hübner, R. Schmogrow, D. Hillerkuss, J. Becker, J. Leuthold, and W. Freude, "Implementation of an ultra-high speed 256-point FFT for xilinx virtex-6 devices," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*. IEEE, 2011, pp. 829–834.

[9] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, 2012, pp. 139–158.

[10] A. Aysu, C. Patterson, and P. Schaumont, "Low-cost and area-efficient FPGA implementations of lattice-based cryptography," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2-3, 2013*, 2013, pp. 81–86.

[11] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact Ring-LWE cryptoprocessor," in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, 2014, pp. 371–391.

[12] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange - A new hope," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016, pp. 327–343.

[13] E. Macii, L. Bolzani, A. Calimera, A. Macii, and M. Poncino, "Integrating clock gating and power gating for combined dynamic and leakage power optimization in digital CMOS circuits," in *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, Sept 2008, pp. 298–303.

[14] O. Reparaz, S. S. Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede, "Masking ring-LWE," *J. Cryptographic Engineering*, vol. 6, no. 2, pp. 139–153, 2016.

[15] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical CCA2-secure and masked Ring-LWE implementation," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 1, pp. 142–174, 2018.

[16] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *CHES*, ser. Lecture Notes in Computer Science, vol. 10529. Springer, 2017, pp. 513–533.

[17] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.