




Accelerating Integer Based Fully Homomorphic Encryption Using Frequency Domain Multiplication

Shakirah Hashim^(✉)  and Mohammed Benaissa

University of Sheffield, Sheffield, UK
shashiml@sheffield.ac.uk

Abstract. In this paper, the hardware implementation of Integer based Fully Homomorphic Encryption (FHE) is investigated. A new methodology is proposed to speed up the encryption process by optimizing the very large asymmetric multiplications required. A frequency domain approach is adopted for the multiplication using the Number Theoretic Transforms (NTTs) where the strict relationship between the NTT parameters is relaxed to allow for more optimized hardware implementations on FPGA. This is achieved specifically by relaxing the traditional requirement for a simple transform kernel in favour of optimal transform lengths and moduli in terms of the number of overall iterations, suitable data path, and FPGA architecture. It is shown both analytically and via implementation results that the proposed approach yields faster FHE over the integers implementations. Based on the methodology, a proposed hardware architecture with optimized NTT parameters synthesized on Xilinx Kintex-7 FPGA shows 55% and 76% speed improvement for Medium and Large key sizes respectively.

Keywords: Fully Homomorphic Encryption · Number Theoretic Transform Hardware implementation

1 Introduction

Fully Homomorphic Encryption (FHE) allows a computation to be done on encrypted data (ciphertext) and no decryption is needed prior to any computation, offering thus better privacy [1]. FHE has emerged as a powerful cryptographic tool in recent years as it has been shown to possess both additive and multiplicative homomorphic properties. However, it is still far from practical deployment due to their complexity, mainly due to the huge key size involved. Three variants of FHE: Lattice-Based, Ring Learning with Error (RLWE) and Integer-Based have been an area of active research in recent years to investigate the potentials and limitations of FHE by investigating software [2–5] and hardware [6–9] implementations.

Implementing Lattice-Based FHE in software was initially proposed in [2]; it requires huge key sizes between 17 Megabytes (MB) to 2.3 Gigabytes (GB) with key generation taking from 2.5 s to 2.2 h. Van Dijk *et al.* revised the original FHE scheme and proposed Integer Based FHE [10] where both homomorphic properties are

computed over the integers with the objective of promoting simplicity in its scheme. Later, Coron *et al.* improved this scheme with smaller key sizes of 0.95 Mb to 802 Mb and key generation time between 4.38 s to 43 min [4].

A modulus switching technique was introduced in [5] which allows leveled multiplication on smaller moduli, hence results in smaller public key sizes. In [5], the authors worked on RLWE based FHE, managed to reduce noise growth from quadratic to linear complexity even without modulus switching. Cousins *et al.* introduced the Chinese Remainder Transform (CRT) on Lattice-Based FHE which splits a larger modulus into multiple moduli so that parallelization can be employed on Field-Programmable Gate Array (FPGA) Virtex 6, however extra time is needed for re-conversion from the Montgomery domain to regular integers [11, 12]. Later, Gentry in [13] presented an encryption of 150-bit Advanced Encryption Standard (AES) homomorphically which takes 73.03 s for key generation and 3 Gb memory usage without bootstrapping.

Apart from FHE, recent research also focused on Somewhat Homomorphic Encryption (SHE) [14, 15]. Smart *et al.* in [3] suggested multiple stages of encryption (known as re-encrypt) on larger message sizes rather than single bit proposed originally in [1]; however, key generation still requires more than an hour even for small key size. An improvised version of [3] is done by introducing a Single Instruction Multiple Data (SIMD) implementation in [16], which performs 4.13 times faster re-encryption and 12 times smaller ciphertext than one without SIMD. Also working on SHE, Poppelmann *et al.* [17] showed that Lattice-Based SHE is possible to be deployed on FPGA Spartan-6 with 9063 Number Theoretic Transform (NTT) coefficients multiplication per second, provided NTT parameters are selected appropriately. The recent SHE work is based on Ring-LWE variants and aimed at accelerating the encryption for cloud computing at the FPGA level and also enlarge the NTT coefficients by introducing a 1228-bit modulus [18]. However, the resulting multiplication process was relatively slower than the software implementation with the same NTT size; 26.67 s and 2.98 s respectively. The bottleneck being the memory access.

To accelerate the FHE performance, the authors in [6] exploited the speed of Graphical Processing Units (GPUs) and encrypted 7.68 times faster than standard Central Processing Unit (CPUs). Then, the authors in [19] introduced Integer-Based FHE by batch to reduce the bottleneck on AES encryption. Later, Doroz *et al.* proposed pre-computation of Schönhage Strassen multiplier parameters which allowed FHE encryption to perform better with only 18.1 ms (ms) [20]. Recently, the concept of a re-encryption box was proposed by Roy *et al.* [21] at the hardware level to reduce the effects of growing noise on the ciphertexts. The re-encryption box is also exploited to accelerate the search operation on the encrypted data.

The first hardware implementation for Integer Based FHE was proposed by Cao *et al.* in [8] with two building blocks of a large NTT multiplier and Barrett reduction to speed up FHE on high-end FPGA technology Virtex 7. Their encryption time is 44.72 times faster than software implementation for 'Large' key size. Comba scheduling is proposed in [22], by utilizing Digital Signal Processing (DSP) slices for uneven operands to shorten the delays during multiplications while reducing 'Write to Memory' operation. Meanwhile, recent research by Cao *et al.* [9] proposed Low Hamming Weight (LHW) design on Virtex 7 to allow simpler multiplications while reducing

hardware usage at the same time. The encryption time of this work outperforming benchmark software implementations by 131 times for ‘Large’ key size, while the encryption time showed by this scheme is between 0.0006 s to 3.317 s, resulting in the best FHE achievement by far with a reasonable speed and small footprint.

Inspired by the significant performance reported with strong potential for improvements, we focused our work on the Integer-Based FHE scheme by Van Dijk *et al.* [10]. The central theme of this scheme is about simplicity. It is easier in terms of parameter selection compared to Lattice-Based while its hardness is based on Greatest Common Divisor (GCD) approximate problem. Furthermore, the sizes of the parameters in Integer-Based FHE are defined clearly in [9], unlike the other variants where only the matrix size is defined rather than bit size.

We propose to accelerate FHE over the integers by adopting frequency domain multiplication using the NTT specifically targeted for FPGAs. The FPGA platform is chosen over custom hardware Application-Specific Integrated Circuits (ASICs) due to the high availability of resources such as DSPs which have dedicated mathematical functions on modern FPGAs.

We followed the seminal work in [2, 5], pronouncing the operands size in four different groups: Toy, Small, Medium and Large as shown in Table 1. At least 150 k to 19 m bits operands are required for the encryption steps which is a large number, hence normal Schoolbook multiplication is no longer efficient. In recent years, there have been many reported ideas by researchers to optimize large number multiplications especially in cryptography; such as Comba [22, 23], Karatsuba [24, 25] and frequency domain conversion methods [14, 26]. The idea of adopting a frequency domain approach on hardware such as in [8, 14, 15, 27, 28] has increasingly gained acceptance as an efficient method to accelerate the multiplication process given its computational complexity being in the order $O(n\log(n))$ for n -bits operand. Researchers in [9, 29, 30] have also shown that NTT hardware implementations outperformed software implementations at certain magnitudes.

Table 1. Test instances for encryption process

Test instances	Bit-length, X_i	Bit-length, B_i	τ
Toy	150 k	936	158
Small	830 k	1476	572
Medium	4.2 m	2016	2110
Large	19.0 m	2556	7659

In this paper, we further advance research in this area by relaxing the strict relationship between the NTT parameters to allow for more optimized hardware implementations on FPGA. To speed up the large integer multiplications required in FHE schemes such as the one proposed in [5], previous research has sought to optimize the multiplication steps within the NTT transform computations by fixing the kernel α to be simply a two or a power of two value [9]. However, such approach tends to impose

restrictions on the possible transform lengths to be deployed, thereby affecting potential optimizations in the overall multiplication process.

In this paper, we propose a different methodology whereby we relax the requirement for a simple kernel in favour of optimal transform lengths and moduli in terms of the number of overall iterations, suitable data path, and FPGA architecture. The kernel multiplications by α 's required for the optimal word lengths and moduli can be easily implemented in the form of Look-Up Tables (LUTs) integral to any FPGA fabrics.

The specific contributions of this paper are summarised as follows:

- A set of NTT parameters that supports large operands for NTT multiplication is proposed.
- Analysis of important hardware design trade-offs; such as the butterfly costs of the NTT building blocks against multiplication iteration for each key group in FHE (Toy, Small, Medium and Large).
- An iterative multiplication method is incorporated to support a small footprint design on hardware while at the same time maximizing the multiplier size to speed-up the overall multiplication process.
- Hardware implementation is validated with results showing improved performance.

The rest of the paper is organised as follow. Section 2 recaps the introduction and mathematical background of FHE over the integers. Our proposed methodology is illustrated in Sect. 3. Section 4 covers the implementation aspects with results given in Sect. 5. The paper concludes with a Conclusion section.

2 Integer Based Fully Homomorphic Encryption

Integer Based FHE needs to perform key generation, encryption and decryption with the additional step of evaluation. Our work in this paper, in line with previously reported implementations [9], is focused solely on the encryption step defined in (1). The work in [9] is workable for binary messages only with message space $\mathcal{Q} = 2\{0, 1\}$; [31] proposed a larger space $\mathcal{Q} > 2$, which means the message can be non-binary with an extended circuit. Their key size is also reduced, although no specific size is reported.

$$c \leftarrow m + 2r + 2 \sum_{i=1}^{\tau} X_i \cdot B_i \text{ mod } X_0 \quad (1)$$

Noted, c is ciphertext; m is a single bit of plaintext binary message with only bit 0 or 1; r is a random signed integer; X_0 is a part of the public key; B_i is a random integer sequence, and X_i is a τ -bits public key sequence with $1 \leq i \leq \theta$. We direct the interested reader to refer to the original work in [5, 10] for details on the parameter selection in (1) and Table 1.

As seen from (1), the FHE encryption step needs two core operational building blocks: (1) Multiplication; and (2) Reduction. These can be designed as individual building block and combined later as a complete process of FHE encryption. Meanwhile, as can be seen from Table 1, both multiplicands X_i and B_i are not symmetrical in size. Multiplicands are also known as operands after this point. Thus, we exploit this

unsymmetrical property to propose a hybrid multiplication approach of Schoolbook and NTT based multiplication. Schoolbook multiplier is employed for the outer iterations whereas the NTT multipliers will be used for the inner multiplications. In fact, employing symmetric multiplication methods for non-symmetric operands leads to significant waste of computational time as well as hardware resources.

2.1 Number Theoretic Transform (NTT) Multiplication

The NTT has been used widely in signal processing for implementing convolution and correlation operations because of its error-free advantages (no rounding or truncation errors) and efficient implementation. Recently there has been a revival of interest in NTTs to be deployed in frequency domain approaches to implementing large operand multiplications required in new offerings in Cryptography. Dai *et al.* in [32] proposed large NTTs of 2^{15} coefficient integrated with CRT in order to accelerate NTRU-based FHE. Meanwhile, diminished-1 NTTs is used for performing SWIFFT hash function in [33] to simplify modular NTTs but is limited to certain modular form such as Fermat primes only. Promising more parallelization, NTT is also widely used in hardware implementation with good performances [8, 34]. The Mathematical representation of an NTT is given in (2). Where $k = 0, \dots, N - 1$ and α is twiddle-factor with the condition of $\alpha^N \equiv 1 \pmod{m}$.

$$X(t) = \sum_{n=0}^{N-1} x(n)\alpha^{nk} \pmod{m} \quad (2)$$

From (2), parameters α, m and N are interdependent. The desirable choice of NTT parameters traditionally involved [35]:

- α to be selected as two or a power of two so that the exponentiation operations required can be implemented as shift operation;
- N to be highly composite, a power of two if possible so that efficient NTT type algorithms can be employed
- m has a special form so that reduction can be a simple operation.

In this paper, we use Classical Modular NTT, with each operation is bounded by ring Z_m where m is moduli. Algorithm 1 describes NTT multiplication steps with 4 underlying steps; Forward Transform, Pointwise Multiplication, Inverse Transform and Carry Accumulation.

Algorithm 1 Number Theoretic Transform (NTT)

```

1: Let  $\alpha$  be a primitive  $n$ -th root of unity in  $m$  and  $b$  is word size
2: Let  $\mathbf{x} = (x_0, \dots, x_{(n/2)-1}, 0 \dots 0)$ ,  $\mathbf{y} = (y_0, \dots, y_{(n/2)-1}, 0 \dots 0)$  and  $\mathbf{z} = (z_0, \dots, z_{n-1})$ 
3: Input:  $x, y, \alpha$ 
4: Output:  $z = x * y$ 
5: Precompute:  $\alpha^i$  where  $i = 0, 1, \dots, n-1$ 
6:   for  $i$  from 0 to  $n-1$ ;
7:      $X = \text{NTT}_{\alpha_i}^m(x_i)$            //Forward Transform
8:      $Y = \text{NTT}_{\alpha_i}^m(y_i)$            //Forward Transform
9:   end for
10:  $Z = X (*) Y \text{ mod } m$            //Pointwise Multiplication
11: for  $i$  from 0 to  $n-1$ ;
12:    $z = \text{INTT}_{\alpha_i^{-1}}^m(Z_i)$      //Inverse Transform
13: end for
14: for  $i$  from 0 to  $n-1$ ;
15:    $z = \sum_{i=0}^{N-1} (z_i \ll (i \cdot b))$  //Carry Accumulation
16: end for
17: Return  $z$ 

```

3 Proposed Methodology

The efficiency of NTT designs as explained before is related closely to the trade-off of its three key inter-related parameters, namely the kernel α , the transform length N and the modulus m . In this paper we stipulate that in the context of FHE where very large multiplications of asymmetrical operands are required, a methodology that allows more flexibility in terms of transform length, offers better scope for improving overall FHE performance on modern FPGA platforms. The proposed methodology is more efficient than traditional methodologies driven by overcoming the complexity of the multiplications by the kernel of the transform at the detriment of the transform length. In this case, the impact of the transform length on overall performance is far more significant than that of the kernel multiplication within the NTT. This is because, the long multiplier unit will be able to cater for larger operand size, thus minimize the number of partial product iterations. As a result, multiplication complexity can be reduced specifically for asymmetric operands. A study of NTT parameters and its optimization is discussed in the next section.

3.1 NTT Parameters Optimization

The central parameter to be optimized is the NTT length as large NTT length can facilitate larger operands, by relaxing the kernel α restriction. The choice of modulus needs a specific consideration, as explained later so that every operation during the NTT over the defined ring is optimized for the targeted hardware. Importantly, the NTT

coefficient must be within the dynamic range b , as expressed in (3) to ensure no overflow error. More details of dynamic range is in [36].

$$\frac{N}{2}(b - 1)^2 < m \tag{3}$$

To illustrate the improvements in operand sizes achieved by the proposed approach we report in Table 2 the comparison between two types of moduli, Solinas and Fermat (F_6); they are 64 bits and 65 bits moduli respectively. Solinas 1 and F_6 1 show the NTT parameter set without optimization, whereas Solinas 2 and F_6 2 show these parameters with our proposed optimization. The optimization is done by enlarging the NTT length as well as relaxing the kernel restriction. As a result, both Solinas 2 and F_6 2 result in much larger multiplier sizes of 1792 bits and 3072 bits which correspond to almost double the length.

Table 2. Comparison between Solinas and Fermat moduli NTT parameters

	Solinas		Fermat	
	Solinas 1 [37]	Solinas 2	Fermat F_6 1 [38]	Fermat F_6 2
N -point	64	128	128	256
Twiddle-factor α	8	$2^{49} - 2^1$	2	$2^{33} - 2^1$
Dynamic range b	28	28	24	24
Multiplier size	896 bits	1792 bits	1536 bits	3072 bits
Modulus m	$2^{64} - 2^{32} + 1$		$2^{64} + 1$	
Reduction cost	1 shift, 2 addition, 2 subtraction		2 addition, 1 subtraction	

Let $y \bmod p$ where $y = 2^{96}a + 2^{64}b + 2^{32}c + d$, a 128 bits integer. The Solinas reduction can be simplified as (4).

$$2^{32}(b + c) - a - b + d \tag{4}$$

Algorithm 2 is used for Special form modulus, of $2^{n-1} \pm 1$ as proposed in [39]. We used this Algorithm for Fermat F_6 1 and Fermat F_6 2 reduction.

Algorithm 2 Special form modulus reduction [39]

- 1: Let $p = 2^m \pm 1$
 - 2: **Input:** x, p
 - 3: **Output:** $y = x \pmod p$
 - 4: $y_1 = x[m - 2: 0] + x[2m - 1: 2m - 2] - x[2m - 3: m - 1]$
 - 5: **if** $y_1 < 0$
 - 6: $y = y_1 + p;$
 - 7: **else**
 - 8: $y = y_1$
 - 9: **end if**
 - 10: **Return** y
-

In terms of reduction's complexity cost, Solinas just needs shift, addition and subtraction. Also, the Solinas form lends itself to efficient FPGA implementation. As the goal of this work is to design a large multiplier on a targeted FPGA, then, Solinas 2 was chosen as the optimal modulus as it covers an acceptable number of operands; 1792 bits and the 64 bits modulus is an optimal fit in terms of a single word. Although $F_6 2$ can cover larger operands of 3072 bits, its 65 bits modulus needs more than a single word operand, which is not optimal for hardware implementation. Even if the diminished-1 number system can be adopted to handle 65 bits modular operation as suggested in [40], the conversion to and from this number system is costly and can become a performance bottleneck in particular for the special case of the zero detection.

Cost Analysis

We first analyzed the operational cost of the NTT block for Solinas 1 and Solinas 2 individually and later we analyzed the cost for overall multiplication during the encryption. For a fair comparison, we presume α for Solinas 1 and Solinas 2 are pre-computed over the Solinas modulus beforehand and stored in LUTs as 64-bits Read-only Memories (ROMs). This was also done before in [27, 41] with the same purpose of speeding-up the kernel multiplication process.

In our work, $64 \times \frac{N}{2}$, pre-computed operands are needed to be stored in the LUTs which is relatively small compared to the available LUTs of the targeted hardware, Kintex 7. Exponentiation by α during the Butterfly operation in (5) can be replaced with a 'Read' operation which is obviously faster than computing exponential α by using an algorithm.

Meanwhile, as NTTs over the ring has a symmetrical root of unity, then it benefits the NTT implementation because the same table also can be used for retrieving α^{-1} for Inverse NTT (INTT) [36]. This way, only N multiplications are required for each transform. As the overall NTT multiplication building block has 2 forward and 1 inverse transforms, then $3N$ multiplications are required. The same goes for the 'Read' operations during the NTT multiplication which is $3 \lfloor \frac{N}{2} \log_2 N \rfloor$.

$$X_i = A_i + \alpha^i B_i, \quad X_{i+\frac{N}{2}} = A_i - \alpha^i B_i \quad (5)$$

The NTT multiplier size n_c can be determined from (6). Division by two is because we use Zero-padded convolution, means only $\frac{N}{2}$ coefficients are employable, and the other $\frac{N}{2}$ appended as zeros.

$$n_c = \frac{N \times b}{2} \quad (6)$$

Table 3 shows the comparison between Solinas 1 and Solinas 2, specifically in terms of operations during the NTT and the space required to store the precomputed operands. As illustrated in Table 3, the Butterfly, 'Read' operation and Addition/Subtraction dominate the cost in Solinas 2, which as expected are higher than Solinas 1, as Solinas 2 caters for larger NTT points. Solinas 2 also requires more LUTs space to store pre-computed α . Crucially though Solinas 2 has the largest NTT points among the similar work done previously in [28, 42].

Table 3. Solinas 1 vs Solinas 2

	Solinas 1 [38]	Solinas 2
Butterfly B_u	576	1344
'Read' operation	576	1344
Point multiplication	64	128
Addition/Subtraction	1152	2688
Precomputed operands in LUTs	32 of 64 bits	64 of 64 bits
NTT multiplier size n_c	896 bits	1792 bits

Next, we analyze the entire multiplication, but first we explain how the multiplication building block works during the FHE encryption. As discussed earlier, the NTT multiplier blocks are used for computing the partial products whereas accumulation is completed using a Schoolbook method. In symmetric operands (n -bits) of the Schoolbook method, n^2 multiplication and $2n - 1$ accumulations are needed. However, as in our case asymmetric multiplication is required and the partial products are completed by the NTT multiplier block; then assumption is made that a partial product iteration P_i represents the number of multiplications as determined in (7). Meanwhile, accumulation A_i in (8) represents the number of additions required for accumulating the partial products. Given two operands of asymmetrical size a (n_a bits) and b (n_b bits) with the multiplier size of n_c -bit.

$$P_i = \left\lceil \frac{n_a}{n_c} \right\rceil \times \left\lceil \frac{n_b}{n_c} \right\rceil \quad (7)$$

$$A_i = \left\lceil \frac{n_a}{n_c} \right\rceil + \left\lceil \frac{n_b}{n_c} \right\rceil \quad (8)$$

Figure 1 explains graphically the impact of multiplier size towards partial product iteration and accumulation. Let a and b , the asymmetric operands of 32-bits and 16-bits respectively. Two different multipliers 8-bit and 16-bit are used to show the relationship between the multiplier size and the complexity of multiplication. 32 bits operand is chunked into the multiple blocks depending on multiplier size. The accumulation chain relies on the partial product iteration. For example, an 8-bit multiplier requires 8 partial product iterations and 5 accumulation chains whereas a 16-bit multiplier only consumes 2 partial product iterations and 2 accumulation chains. Essentially, fewer iterations are needed for larger multipliers while long carry accumulation chains also can be minimized.

We analyze the complexity of the multiplication building block, during the FHE encryption with different key sizes Toy, Small, Medium and Large as illustrated in Table 4. P_i and A_i are obtained from Eqs. 7, and 8 respectively. We also include the Butterfly cost B_i in Table 4 which corresponds to the number of butterflies involved during the NTT multiplication to perform FHE encryption as shown in (9). The values of P_i , A_i and B_i in Table 4 represent the overall costs and complexity of the multiplication during the FHE encryption.

$$B_i = B_u \times P_i \tag{9}$$

As can be seen from Table 4, if the multiplier is large enough to cover the operands b_i in a minimum NTT block, then the partial product iterations and accumulation are reduced significantly. For example, Toy operand b_i can fit in a single NTT block of Solinas 2. However, for Solinas 1, operand b_i does not fit a single NTT block, instead 2 NTT block iterations are needed, thus, complicates the multiplication process quadratically.

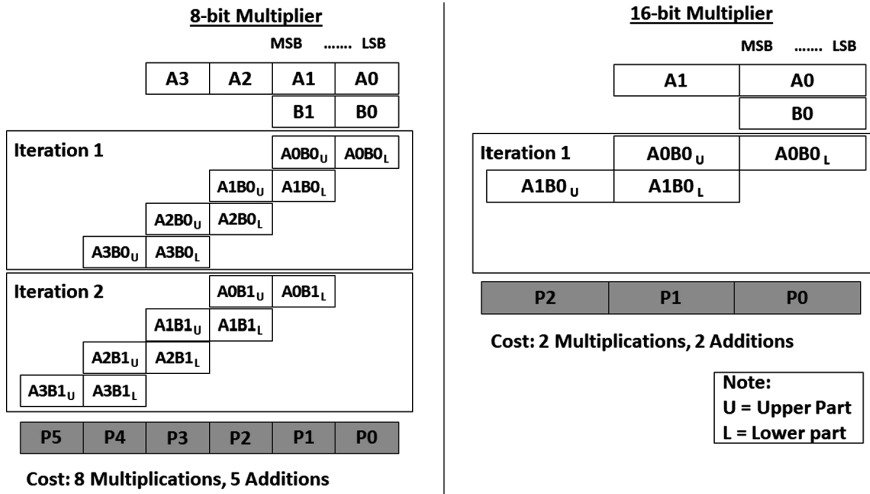


Fig. 1. 8-bit multiplier vs 16-bit multiplier

Overall, the number of partial product iterations (P_i) and accumulations (A_i) in Solinas 2 is reduced drastically compared with Solinas 1. In fact, the butterfly cost in Solinas 2 is also much lower than Solinas 1 despite Solinas 2 incurring a larger butterfly cost than Solinas 1 in a single multiplier block.

Based on this analysis, we confirm that choosing appropriate multiplier size can significantly reduce the multiplication building block complexity and therefore by relaxing the kernel restriction to enable longer length NTT, the overall complexity cost of the multiplication building blocks is reduced significantly.

Also, from the complexity analysis in Table 4, our parameter optimization using Solinas 2 shows a significant improvement compared to Solinas 1. For that reason, we conclude that Solinas 2 is more efficient for large asymmetric operands. This is due to the large size of the multiplier which leads to small partial product iterations and short carry chain. In fact, Solinas 2 also costs fewer butterflies, hence reduce entire multiplication complexity.

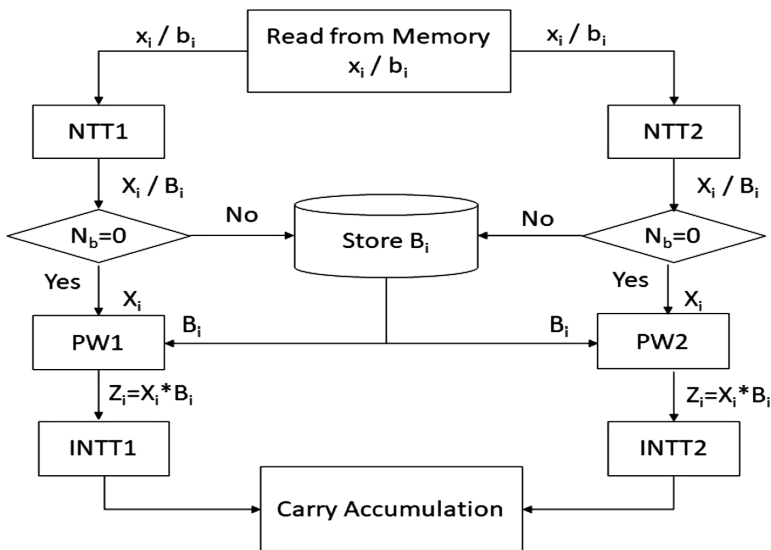
Table 4. Complexity costs of Solinas 1 and Solinas 2

Key size	P_i		A_i		B_i	
	Solinas 1	Solinas 2	Solinas 1	Solinas 2	Solinas 1	Solinas 2
Toy	336	84	170	85	193536	112896
Small	1854	464	929	465	1067904	623616
Medium	14064	4688	4691	2346	8100864	6300672
Large	63618	21206	21209	10605	36643968	28500864

4 The Architecture of NTT Multiplier

Labview FPGA 15 is being used for this hardware implementation, targeted to Xilinx Kintex-7 XC7K160T FPGA device and Xilinx Vivado 2014.4 compiler. Given the size of the operands needed, it is assumed that Block Random Access Memory (BRAMs) is used and sufficient to store X_i and B_i as multiple data chunks where each chunk is b bits size.

The architecture of the NTT Multiplier is depicted in Fig. 2. Initially, both NTT1 and NTT2 are used to transform the B_i operands. After the B_i operands are completely transformed into frequency domain, they are stored in a BRAM B_i . Next, X_i are transformed into frequency domain using NTT1 and NTT2. This also means for each iteration; the NTT block can cover $2n_c$ bits. Then, pointwise (PW) multiplication takes place in parallel by 2 PW units; PW1 and PW2 have 128 points each. During pointwise multiplication, X_i is fed on the fly from both NTT1 and NTT2 outputs, whereas B_i is read from BRAM B_i . The output of PW1 and PW2 then are loaded into INTT1 and

**Fig. 2.** The proposed NTT multiplier architecture

INTT2 respectively. The proposed design is pipelined, so after the INTT takes place, then the following output of INTT is generated at the following clock cycle. The product is then loaded into the accumulation unit for addition and carry management. This unit merely involves shifting and addition.

In the case where B_i does not fit into a single NTT unit, then pointwise multiplication should be done iteratively. For example, operands B_i for Medium and Large exceed the multiplier size as they need two NTT blocks; so pointwise multiplication must undergo 2 iterations to complete the multiplication for both blocks, hence more clock cycles required for this case.

5 Results and Discussion

The synthesis result for our proposed NTT Multiplier is within the available resources of the targeted hardware Kintex-7 as seen in Table 5. As can be seen, registers and LUTs are same for all key sizes Toy, Small, Medium and Large. This has happened because the same NTTs unit is being used for each group. The latency is different due to the number of iteration for each group is different. Meanwhile, BRAMs represent an amount to store the operands X_i and B_i as well as the final results after the reduction.

Table 5. Synthesis results for proposed NTT multiplier

	Toy	Small	Medium	Large
Registers	18462	18462	18462	18462
LUTs	26328	26328	26328	26328
BRAMs	41	209	526	702
Freq (MHz)	165.21	164.69	161.00	154.44

The latency in Table 6 is calculated using the clock cycles count and the synthesis design frequencies which is generated by the tools. As the timing for both the multiplication and reduction building block are obtained, then the encryption time Enc_t can be computed as (10).

$$Enc_t = (Group\ 1\ timing \times \tau) + (2 \times Group\ 2\ timing) \quad (10)$$

From (10), the first bracket refers to multiplication timing whereas the second bracket refers to reduction timing. Note that we used Barrett reduction which also

Table 6. Latency and timing for proposed NTT multiplier of each group

Key size	Latency	Timing (ms)	Group2	Latency	Timing (ms)
Toys	4542	0.027	Toys2	4542	0.027
Small	4922	0.030	Small2	4922	0.030
Medium	6802	0.042	Medium2	12246	0.076
Large	15256	0.100	Large2	29154	0.0189

utilized the same NTT building blocks with different operands [9]. Multiplication by two for the reduction building block is because the Barrett reduction needs two large multiplications [43]. The Encryption time of each group is presented in Table 6.

We also compared our result with previous research [9] in Table 7. As can be seen, our design outperforms [9] for the Medium and Large groups. This proves that our design manages to reduce multiplication complexity specifically for large operands such as Medium and Large. Although [9] performs better in Toy and Small, but the encryption time of our design shows that it does not increase gradually from Toy to Large. We notice that our design is not efficient for Toy and Small because the operand B_i just utilized 20% and 41% out of full NTT blocks respectively. This can be improved in the future by designing a scalable design which can be flexible depending on size of operands.

Table 7. Encryption Time of our proposed design and previous research [9]

Key size	Encryption time (s)		
	Proposed design	LHW [9]	Low latency [9]
Toy	0.004	0.001	0.003
Small	0.017	0.011	0.056
Medium	0.089	0.198	1.000
Large	0.770	3.317	16.595

6 Conclusion

In this paper, we proposed a new methodology to speed up the large modular multiplications required in FHE schemes in frequency domain using NTTs. The methodology is based on relaxing the strict relationship between the NTT parameters imposed by having a simple transform kernel. In our approach, more emphasis is put on the transform length as it was shown that this parameter has more effect on overall hardware performance. Both Analytical and implementation presented in this paper show that the proposed methodology leads to improved large NTT multiplication. In fact, our Optimized NTT Multiplier is 55% and 76% faster than [9] for Medium and Large group respectively. The results attained illustrate that FHE encryption time is improved. Further enhancements can be carried out by deploying several NTT blocks in parallel.

References

1. Gentry, C.: A fully homomorphic encryption scheme. Ph.D thesis, Stanford University (2009)
2. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_9

3. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_25
4. Coron, J.-S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 487–504. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_28
5. Coron, J.-S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 446–464. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_27
6. Wang, W., Hu, Y., Chen, L., Huang, X., Sunar, B.: Accelerating fully homomorphic encryption using GPU. In: High Performance Extreme Computing (HPEC), pp. 1–5. IEEE (2012)
7. Wang, W., Huang, X.: FPGA implementation of a large-number multiplier for fully homomorphic encryption. In: International Symposium on Circuits and Systems, pp. 2589–2592. IEEE (2013)
8. Cao, X., Moore, C., O’Neill, M., O’Sullivan, E., Hanley, N.: Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction. <http://eprint.iacr.org/2013/616>
9. Cao, X., Moore, C., Oneill, M., Osullivan, E., Hanley, N.: Optimised multiplication architectures for accelerating fully homomorphic encryption. IEEE Trans. Comput. **65**, 2794–2806 (2016)
10. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
11. Cousins, D.B., Rohloff, K., Peikert, C., Schantz, R.: SIPHER: scalable implementation of primitives for homomorphic encryption—FPGA implementation using Simulink. In: High Performance Extreme Computing Conference (2011)
12. Cousins, D.B., Rohloff, K., Peikert, C., Schantz, R.: SIPHER: an update on SIPHER (Scalable Implementation of Primitives for Homomorphic EncRyption) - FPGA implementation using Simulink. In: High Performance Extreme Computing Conference (2012)
13. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_49
14. Öztürk, E., Doröz, Y., Sunar, B., Savaş, E.: Accelerating somewhat Homomorphic Evaluation using FPGAs. IACR Cryptology EPrint Archive, 1–15, <https://eprint.iacr.org/2015/294>
15. Doröz, Y., Öztürk, E., Savaş, E., Sunar, B.: Accelerating LTV based homomorphic encryption in reconfigurable hardware. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 185–204. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_10
16. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Cryptology ePrint Archive, Report 2011/133 (2011), <http://eprint.iacr.org/2011/133>
17. Doröz, Y., Öztürk, E., Savaş, E., Sunar, B.: Accelerating LTV based homomorphic encryption in reconfigurable hardware. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 185–204. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_10

18. Roy, S.S., Jarvinen, K., Vliegen, J., Vercauteren, F., Verbauwhede, I.: HEPCloud: an FPGA-based multicore processor for FV somewhat function evaluation. *IEEE Trans. Comput.* (2018)
19. Cheon, J.H., et al.: Batch fully homomorphic encryption over the integers. In: Johansson, T., Nguyen, Phong Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 315–335. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_20
20. Doröz, Y., Öztürk, E., Sunar, B.: Accelerating fully homomorphic encryption in hardware. *IEEE Trans. Comput.* **64**(6), 1509–1521 (2015)
21. Roy, S.S., Vercauteren, F., Vliegen, J., Verbauwhede, I.: Hardware assisted fully homomorphic function evaluation and encrypted search. *IEEE Trans. Comput.* **66**(9), 1562–1572 (2017)
22. Moore, C., O’Neill, M., Hanley, N., O’Sullivan, E.: Accelerating integer-based fully homomorphic encryption using Comba multiplication: In: *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*. IEEE (2014)
23. Großschädl, J., Avanzi, R.M., Savas, E., Tillich, S.: Energy-efficient software implementation of long integer modular arithmetic. *Cryptograph. Hardw. Embedded Syst.* **04**(104), 75–90 (2005)
24. Bos, J.W.: High-performance modular multiplication on the cell processor. In: Hasan, M.A., Hellesteth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 7–24. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13797-6_2
25. Basu Roy, D., Mukhopadhyay, D.: An efficient high speed implementation of flexible characteristic-2 multipliers on FPGAs. In: Rahaman, H., Chattopadhyay, S., Chattopadhyay, S. (eds.) VDAT 2012. LNCS, vol. 7373, pp. 99–110. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31494-0_12
26. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_8
27. Liu, Z., Seo, H., Sinha Roy, S., Großschädl, J., Kim, H., Verbauwhede, I.: Efficient ring-LWE encryption on 8-bit AVR processors. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 663–682. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_33
28. Emmart, N., Weems, C.: High precision integer multiplication with a GPU. In: *IEEE International Symposium on Parallel and Distributed Processing Workshops and Ph.D Forum*, pp. 1781–1787 (2011)
29. Jayet-Griffon, C., Cornélie, M. A., Maistri, P., Elbaz-Vincent, P., Leveugle, R.: Polynomial multipliers for fully homomorphic encryption on FPGA. In: *2015 International Conference on ReConfigurable Computing and FPGAs*. IEEE (2016)
30. Chen, D.D., Yao, G.X., Cheung, R.C.C., Pao, D., Koç, Ç.K.: Parameter space for the architecture of FFT-Based montgomery modular multiplication. *IEEE Trans. Comput.* **65**(1), 147–160 (2016)
31. Nuida, K., Kurosawa, K.: (Batch) fully homomorphic encryption over integers for non-binary message spaces. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 537–555. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_21
32. Dai, W., Doroz, Y., Sunar, B.: Accelerating NTRU based homomorphic encryption using GPUs. In: *2014 IEEE High Performance Extreme Computing Conference, HPEC 2014*. IEEE (2014)
33. Györfi, T., Creț, O., Borsos, Z.: Implementing modular FFTs in FPGAs - a basic block for lattice-based cryptography. In: *16th Euromicro Conference on Digital System Design*, pp. 305–308 (2013)

34. Reddy, N., Amarnath, D., Srinivasa, Rao, J., Suman, V.: Design and simulation of FFT processor using radix-4 algorithm using FPGA. *Int. J. Adv. Sci. Technol.* **61**, 53–62 (2013)
35. Burrus, C., Eschenbacher, P.: An in-place, in-order prime factor FFT algorithm. *IEEE Trans. Acoust. Speech Sign. Process.* **29**(4), 806–817 (1981)
36. Brassard, G., Paul, B.: *Algorithmics: Theory and Practice*. Prentice Hall, Upper Saddle River (1988)
37. Solinas, J.A.: *Generalized mersenne numbers*. Faculty of Mathematics, University of Waterloo (1999)
38. Kalach, K., David, J.P.: Hardware implementation of large number multiplication by FFT with modular arithmetic. In: *The 3rd International Conference on IEEE-NEWCAS 2005*, pp. 267–270. IEEE (2005)
39. Zimmermann, R.: Efficient VLSI implementation of modulo $(2^{\sup n} \text{ spl plusmn } 1)$ addition and multiplication. In: *Computer Arithmetic Proceedings 14th IEEE Symposium*, pp. 158–167. IEEE (1999)
40. Leibowitz, L.: A simplified binary arithmetic for the fermat number transform. *IEEE Trans. Acoust. Speech Sign. Process.* **24**(5), 356–359 (1976)
41. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: *USENIX Security Symposium*, vol. 2016 (2016)
42. Cao, X., Moore, C.: New integer-FFT multiplication architectures and implementations for accelerating fully homomorphic encryption. *IACR Cryptology EPrint Archive* 2013/624 (2013)
43. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) *CRYPTO 1986*. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_24