Optimised Multiplication Architectures for Accelerating Fully Homomorphic Encryption

Xiaolin Cao, Ciara Moore, *Student Member, IEEE*, Máire O'Neill, *Senior Member, IEEE*, Elizabeth O'Sullivan, Neil Hanley

Abstract—Large integer multiplication is a major performance bottleneck in fully homomorphic encryption (FHE) schemes over the integers. In this paper two optimised multiplier architectures for large integer multiplication are proposed. The first of these is a low-latency hardware architecture of an integer-FFT multiplier. Secondly, the use of low Hamming weight (LHW) parameters is applied to create a novel hardware architecture for large integer multiplication in integer-based FHE schemes. The proposed architectures are implemented, verified and compared on the Xilinx Virtex-7 FPGA platform. Finally, the proposed implementations are employed to evaluate the large multiplication in the encryption step of FHE over the integers. The analysis shows a speed improvement factor of up to 26.2 for the low-latency design compared to the corresponding original integer-based FHE software implementation. When the proposed LHW architecture is combined with the low-latency integer-FFT accelerator to evaluate a single FHE encryption operation, the performance results show that a speed improvement by a factor of approximately 130 is possible.

Index Terms—Fully homomorphic encryption, large integer multiplication, low Hamming weight, FPGA, cryptography

1 Introduction

Fully homomorphic encryption (FHE), introduced in 2009 by Gentry [1], is set to be a key component of cloud-based security and privacy-related applications (such as privacy-preserving search, computation outsourcing and identity-preserving banking) in the near future. However, almost all FHE schemes, [1]–[11], reported to date face severe efficiency and cost challenges, namely, impractical public-key sizes and a very large computational complexity. Existing software implementations highlight this shortcoming; for example, in a software implementation of the original lattice-based FHE scheme by Gentry and Halevi [3], also known as the GH scheme, the public-key sizes range from 70 Megabytes to 2.3 Gigabytes.

There have been several theoretical advancements in the area of FHE; there are various types of FHE schemes, such as the original lattice-based FHE scheme, integer-based FHE and also FHE schemes based on learning with errors (LWE) or ring learning with errors (RLWE). Furthermore, optimisations have been introduced [6], [12], such as modulus switching and leveled FHE, which involves the use of bounded depth circuits.

To improve the performance of FHE schemes, recently there has been research into the hardware acceleration of various FHE schemes [13]–[20]. The lattice-based FHE scheme by Gentry and Halevi [3] has been implemented on a Graphics Processing Unit (GPU) by Wang *et al.* and significant speed improvements are achieved [14], [16]; however, for large security parameter levels, memory is a major bottleneck. Moreover, the use of large hardware multipliers, targeting FPGA and ASIC technology, to enhance the performance and minimise resource usage of implementations of the same FHE scheme [3] has also been explored [15]–[19]. Wang *et al.* [17] present a 768000-bit multiplier using the fast Fourier transform (FFT) on custom hardware. This design offers reduced power consumption and a speed up factor of 29 compared to CPU. Döroz *et al.* proposed a design for a million bit multiplier on custom hardware [18], [19] for use in FHE schemes. The proposed multiplier offers reduced area usage and can multiply two million-bit integers in 7.74 ms.

In this work, we investigate the FHE scheme over the integers, proposed by van Dijk *et al.* [4]. This integer-based scheme has been extended by Coron *et al.* to minimise public-key sizes [9], this extension is referred to in this work as an abbreviation of the authors' names, CNT. Previous research by the authors [21]–[23] has investigated the acceleration of the large integer multiplication required in this FHE scheme. The possibility of implementing the multiplications required in the integer-based FHE scheme using the embedded DSP blocks on a Xilinx Virtex-7 FPGA with the Comba multiplication algorithm was analysed by Moore *et al* [21], [22]. Also, Cao *et al.* [23] presented the first hardware implementation of the encryption step in the CNT FHE scheme, with a significant speed-up of 54.42 compared with the corresponding software reference implementation [9]. However, its hardware cost heavily exceeded the hardware budget of the available Xilinx Virtex-7 FPGA.

The use of low Hamming weight (LHW) parameters has previously been employed to improve the performance of cryptographic algorithms and implementations. The Hamming weight of an integer is used to denote the number of 1 bits in the integer in binary format. Thus, an integer multiplication with LHW operands can be simplified to

X. Cao was previously with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast. He is now with Titan-IC Systems, Belfast. (e-mail: xcao03@qub.ac.uk)
C. Moore, M. O'Neill, E. O'Sullivan and N. Hanley are with the Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Northern Ireland (e-mail: {cmoore50, maire.oneill, e.osullivan, n.hanley}@qub.ac.uk)

a series of additions. For example, LHW exponents have been used to accelerate modular exponentiation in RSA and discrete logarithm cryptography [24]. NTRU cryptography can use LHW polynomials to achieve high performance implementations [25] and McLoone and Robshaw proposed a low-cost GPS authentication protocol implementation for RFID applications using a LHW challenge [26]. Coron *et al.* [9] have suggested that the operand B_i in the encryption step of their FHE scheme, as outlined in Section 2, can be instantiated as a LHW integer with a maximum Hamming weight of 15, while the bit-length of B_i remains unchanged [9]. To the best of the authors' knowledge, no FHE hardware architecture has been reported to date that exploits LHW operands to accelerate the encryption step, which is the objective of this paper.

In this work, optimised hardware architectures for the cryptographic primitives in FHE over the integers are proposed, specifically targeted to the FPGA platform. The reconfigurable FPGA technology lends itself to widespread use in cryptography, as fast prototyping and testing of designs is possible and adjustments in parameters can be made. Moreover, Xilinx Virtex-7 FPGAs contain several DSP slices, which offer dedicated multiplication and accumulation blocks. For these reasons, FPGA technology is chosen as the target platform rather than ASIC technology in this work.

Optimised hardware architectures targeted for the components within FHE schemes are essential to assess and also to enhance the practicality of FHE schemes. FHE is highly unpractical when hardware accelerators are not considered. In addition, prior generic cryptographic designs of the underlying components in FHE schemes, such as multiplication, do not consider such large parameters. Designing architectures to optimally accommodate the large parameters required for sufficient security in actual deployment is widely regarded as a major challenge. This research explores the potential performance that can be achieved for an existing FHE scheme over the integers using FPGA technology utilising such large parameters suggested in current literature for maximum security levels.

More specifically, our contributions are as follows: (i) an optimised integer-FFT multiplier architecture is proposed, which minimises hardware resource usage and latency, in order to accelerate the performance of an FHE scheme over the integers; (ii) a novel hardware architecture of a large integer multiplication using a LHW operand is proposed for the same purpose. The low-latency integer-FFT multiplier architecture and the LHW architecture are both implemented and verified on a Xilinx Virtex-7 FPGA. These architectures are combined, such that the LHW multiplier is used for the multiplications and the FFT multiplier is used for the modular reduction within the FHE encryption step. The encryption step in the CNT FHE scheme is evaluated for these architectures and the analysis results show that a speed-up of approximately 130 is achieved, compared to the reference software implementation [9].

The rest of the paper is organised as follows. FHE over the integers is introduced in Section 2. A brief description of FFT multiplication is given in Section 3. In Section 4, the low-latency FFT multiplier architecture is described. The proposed LHW hardware architecture of the large multiplier is described in Section 5. Section 6 details the implementation, performance and comparison of the architectures. Finally, Section 7 concludes the paper.

TABLE 1: Parameter sizes for encryption step (1)

Param. Sizes	**φ, Bit-length of X_i**	**δ, Bit-length of B_i**	**θ**
Toy	150k	936	158
Small	830k	1476	572
Medium	4.2m	2016	2110
Large	19.35m	2556	7695

2 FHE OVER THE INTEGERS

Of the previously proposed schemes that can achieve FHE, the integer-based FHE scheme, which was originally proposed by van Dijk *et al.* and subsequently extended by Coron *et al.* [9] in 2012, has the advantage of comparatively simpler theory, as well as the employment of a small public-key size of no more than 10.1MB. The FHE scheme over the integers was chosen in this research because of the available parameter sizes for the CNT scheme and the use of similar underlying components in other FHE schemes, allowing the potential future transfer of this research to other FHE schemes. Moreover, batching techniques for FHE over the integers [10] indicate further potential gains in efficiency. Furthermore, the use of previously mentioned optimisations, such as leveled FHE schemes, could also enhance performance. Indeed, there has been further recent research into adaptations and algorithmic optimisations of FHE over the integers to improve performance [27], [28].

This research is focused on the encryption step of the CNT integer-based FHE scheme [9]; the encryption step contains two central building blocks: modular reduction and multiplication, which are used throughout other steps and other FHE schemes. The mathematical definition of the encryption step is given in Equation (1).

$$c \leftarrow m + 2r + 2\sum_{i=1}^{\theta} X_i \cdot B_i \; mod \; X_0 \tag{1}$$

where c denotes the ciphertext; $m \in \{0, 1\}$ is a 1-bit plaintext; r is a random signed integer; $X_0 \in [0, 2^{\varphi})$, is part of the public-key; $\{B_i\}$ with $1 \leq i \leq \theta$ is a random integer sequence, and each B_i is a δ-bit integer; $\{X_i\}$ with $1 \leq i \leq \theta$ is the public-key sequence, and each X_i is a φ-bit integer. The four groups of test parameters provided by Coron *et al.* [9] are given in Table 1. For more information on the parameter selection and for further details on the integer-based FHE scheme, see [4], [9].

From Equation (1) and Table 1, it is easy to see that large integer multiplications and modular reduction operations are required in the CNT FHE scheme. The reference software implementation uses the NTL library [29] on the Core-2 Duo E8400 platform and it takes over 7 minutes to complete a single bit encryption operation with large security parameter sizes [9]. This result highlights the need for further optimisations and targeted hardware implementations before this scheme can be considered for practical applications. In this research, novel hardware designs incorporating the use of a LHW parameter in the encryption step are proposed to accelerate performance; two multipliers are presented for this purpose: a low-latency multiplier for general large integer multiplication, used in the modular reduction, and a LHW multiplier for the multiplications required in the encryption step, $X_i \cdot B_i$, which can be seen in Equation (1). The combination of these optimised multipliers contributes to a significant acceleration in performance.

3 FFT Multiplication

FFT multiplication is the most commonly used method for multiplying very large integers. Hence, this is used for the large integer multiplications required in the modular reduction, as mentioned in the previous section. The FFT algorithm has been used in the majority of other hardware architecture designs, such as [16]–[19] to implement Gentry and Halevi's FHE scheme [3]. However, the integration of a LHW multiplier with an FFT multiplier targeted for the application of FHE schemes has not previously been considered. For the application of FHE, the integer multiplication must be exact and therefore a version of FFT multiplication, also known as a number theoretic transform (NTT) is used. The term FFT refers to the NTT throughout the rest of this research.

The FFT multiplication algorithm [30], [31] involves forward FFT conversion, modular pointwise multiplication, inverse FFT conversion and finally the resolving of the carry chain. The modulus, p, used in the FFT algorithm is selected to be the Solinas modulus, $p = 2^{64} - 2^{32} + 1$, and differs from the modulus X_0 required in the encryption step, defined in Equation (1). Modulus reduction and the selection of p is discussed in Section 4.2. The FFT point number is denoted as k, the twiddle factor is denoted as ω and b is the base unit bit length. The adapted algorithm of integer-FFT multiplication used for the proposed low-latency multiplication architecture is given in Algorithm 1.

4 The Low- Latency Multiplication Architecture

The core aim of this architecture is to increase the parallel processing ability of the FFT-multiplier design in order to reduce the latency, with the constraint that the proposed architecture does not exceed the hardware resource budget of the targeted FPGA platform, a Virtex-7 XC7VX980T. This work improves upon the previous work [23], in which the proposed FFT multiplier design did exceed the FPGA resource budget. The proposed low-latency hardware multiplier architecture consists of shared RAMs, an integer-FFT module and a finite state machine (FSM) controller. As the bit lengths involved in the multiplications of the encryption step in the CNT FHE scheme are in the region of a million bits, and on-chip register resource is expensive, it is appropriate to use off-chip RAM to store the operands, x and y, and the final product results, z. We assume that there is sufficient off-chip memory available for the proposed accelerator architecture to store its input operands and final results. We believe that this is a reasonable assumption as the accelerator could be viewed as a powerful coprocessor device, sharing memory with the main workstation (be it a server or PC) over a high speed PCI bus.

Algorithm 1: Proposed parallelised integer-FFT multiplication architecture algorithm

```
Input: $n$-bit integers $x$ and $y$, base bit length $b$,
       FFT-point $k$
Output: $z = x \times y$
 1: $x$ and $y$ are $n$-bit integers. Zero pad $x$ and $y$ to $2n$ bits
    respectively;
 2: The padded $x$ and $y$ are then arranged into $k$-element
    arrays respectively, where each element is of length
    $b$-bits;
    ——————FFT CONVERSION——————
 3: for $i$ in $0$ to $k-1$ do
 4:   $Y_i \leftarrow FFT(y_i)\,(mod\,p)$;
 5:   $X_0 \leftarrow FFT(x_0)\,(mod\,p)$;
 6:   for $j$ in $1$ to $\lceil \frac{k}{2} \rceil - 1$ do
 7:     $X_{2j-1} \leftarrow FFT(x_{2j-1})\,(mod\,p)$;
 8:     $X_{2j} \leftarrow FFT(x_{2j})\,(mod\,p)$;
 9:   end for;
10: end for;
    ————POINTWISE MULTIPLICATION————
11: $Z_0 \leftarrow X_0 \cdot Y_0\,(mod\,p)$;
12: for $i$ in $1$ to $k-1$ do
13:   for $j$ in $1$ to $\lceil \frac{k}{2} \rceil - 1$ do
14:     $Z_{2j-1} \leftarrow X_{2j-1} \cdot Y_i\,(mod\,p)$;
15:     $Z_{2j} \leftarrow X_{2j} \cdot Y_i\,(mod\,p)$;
16:   end for;
17: end for;
    ——————IFFT CONVERSION——————
18: $z_0 \leftarrow IFFT(Z_0)$;
19: for $i$ in $1$ to $\lceil \frac{k}{2} \rceil - 1$ do
20:   $z_{2i-1} \leftarrow IFFT(Z_{2i-1})$;
21:   $z_{2i} \leftarrow IFFT(Z_{2i})$;
22: end for;
    ——————ACCUMULATION——————-
23: for $i$ in $0$ to $k-1$ do
24:   $z = \sum_{i=0}^{k-1} (z_i \ll (i \cdot b))$, where $\ll$ is the left shift
      operation;
25: end for;
    return $z$
```

The integer-FFT module is the core module in this design. It is responsible for generating the multiplication result, and its architecture is illustrated in Fig. 1. Furthermore, Algorithm 1 outlines the core steps in the module, and can be used in conjunction with Fig. 1 to understand the parallelism used within the design. As can be seen in Fig. 1, two FFT modules, two IFFT modules and two point-wise multiplications are used in parallel in this low-latency architecture to reduce the latency of the overall design whilst ensuring the hardware area required for the proposed architecture remains within the limits of the target FPGA platform. An FSM controller is responsible for distributing the control signals to schedule the integer-FFT module, and it also implements an iterative school-book multiplication accumulation logic [30] to accumulate the block products generated by the integer-FFT module.

We adopt an iterative method for the ordering of FFT conversions and point-wise multiplications to maximise resource usage in the proposed low-latency multiplication

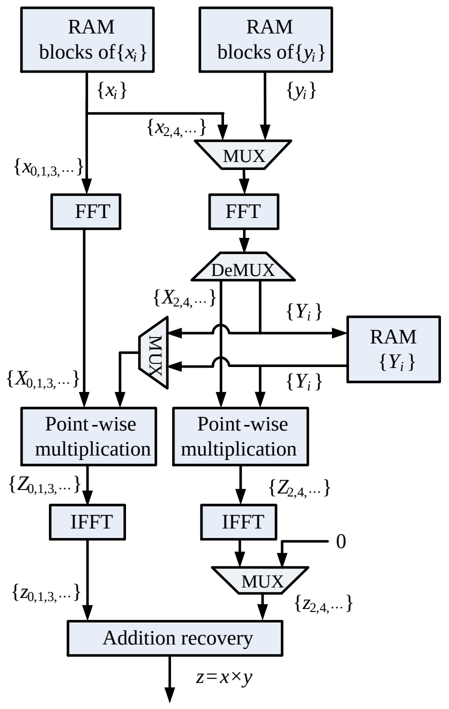


Fig. 1: The integer-FFT module architecture used in the low-latency architecture.

architecture. The iterations are divided into two levels, namely, an inner iteration, used to iterate the data blocks of x, and an outer iteration, used to iterate the data blocks of y. These iterations are outlined in Algorithm 1; for example in the FFT conversion stage, where lines 4 to 5 describe the outer iterations and lines 7 to 8 describe the inner iterations. As we instantiate two FFT and two IFFT modules in the proposed low-latency multiplication architecture, two block products can be processed in parallel in each inner iteration after the initial single inner iteration (see lines 14 to 15 of Algorithm 1). Thus, when the multiplication inputs are very large, the inputs are arranged into arrays of k elements where k is much greater than 1, and the total inner iteration count can be reduced to almost a half using the proposed low-latency architecture with two parallel FFT, point-wise multiplication and IFFT modules compared to when only one FFT, point-wise multiplication and one IFFT module is used and this reduces the total latency of the proposed design.

As an example to illustrate the iterative approach taken in the proposed low-latency multiplication architecture, Fig. 2 demonstrates the proposed block multiplication accumulation logic. As the bit-lengths of multiplication operands, x and y, are too large, which is true of multiplication required in any FHE scheme, the multiplication cannot be completed in a single integer-FFT multiplication step. Thus, inputs are divided into smaller units, and the multiplication is carried out on these smaller units. This smaller multiplication required within the integer-FFT multiplication is described in lines 11-17 in Algorithm 1 and it is demonstrated in the small example given in Fig. 2, where X is divided into five data blocks from LSB to MSB, X_0 to X_4 and Y is divided into two data blocks from LSB to MSB, Y_0 and Y_1. After the first initial iteration, where $X_0 \times Y_0$ is calculated, as defined in line 11 in Algorithm 1, in each subsequent iteration two block products are computed, $X_i \times Y_j$ with $0 \leq i < 5$ and $0 \leq j < 2$, as defined in lines 14-15 in Algorithm 1. This can be seen in inner iteration one in Fig. 2, where $X_1 \times Y_0$ and $X_2 \times Y_0$ are carried out simultaneously. Thus, in this example, the inner iteration count is reduced to 3 rather than 5, as seen in Fig. 2. For larger inputs this reduction in the inner iteration count reduces the overall latency of the proposed design.

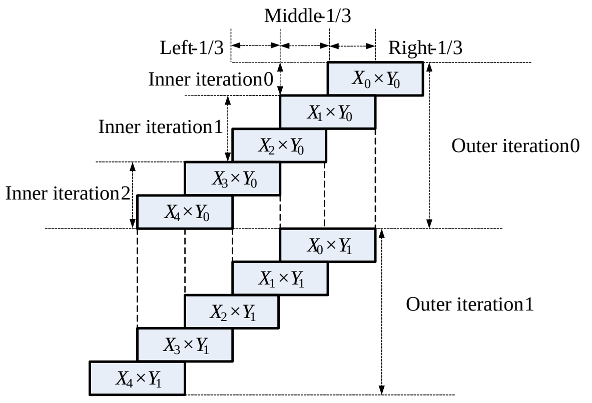


Fig. 2: The proposed block-accumulation logic used in the low-latency architecture.

4.1 The FFT/IFFT Module

A k-point FFT requires $log_2 k$ processing stages, and each processing stage is composed of $\frac{k}{2}$ parallel butterfly modules. The use of a radix-2 fully parallel architecture for FFT and IFFT modules is expensive in terms of hardware resource usage [23]. Therefore, we propose the use of a serial FFT/IFFT architecture, which still requires $log_2 k$ processing stages for a k-point FFT, but only one butterfly module is required in each processing stage. Therefore, the total butterfly module count can be reduced from $\frac{k}{2} \times log_2 k$ to $4 \times log_2 k$.

As there are two FFT modules in the design, and in each clock cycle both read b-bit data blocks from the off-chip memory in parallel, the total bit-width of two FFT RAM data bus bit-widths is equal to $2b$, and the total address bus bit-width of the two operand read ports is $log_2 \frac{n_x}{b} + log_2 \frac{n_y}{b}$, where the input operands are x and y, and their bit-lengths are n_x and n_y respectively.

The FFT processing stage architecture is illustrated in Fig. 3. Each processing stage consists of several buffers and one butterfly module. The buffer count of both up and down branches in each processing stage is the same. The IFFT processing stage architecture is similar to the FFT with the difference that the buffer count of the FFT processing stages decreases from $\frac{k}{2}$ to 1, but the buffer count of the IFFT processing stages increases from 1 to $\frac{k}{2}$. In some literature, this architecture is called radix-2 multi-path delay commutator (R2MDC) [32]. In most applications, only one FFT module is employed for data processing, and the decimation-in-frequency (DIF) R2MDC is commonly used. In this paper, the decimation-in-time (DIT) R2MDC is implemented in both the FFT and the IFFT. The difference between the DIT and DIF in terms of hardware resource usage and performance is minimal, thus DIT is arbitrarily chosen in this research.

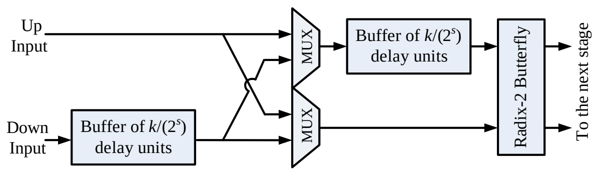

Up Input
Down Input
Buffer of $k/(2^s)$ delay units
MUX
MUX
Buffer of $k/(2^s)$ delay units
Radix-2 Butterfly
To the next stage

Fig. 3: The proposed s-th processing stage used in FFT

The proposed point-wise multiplication module required after the FFT and before the IFFT operations consists of two parallel modular multiplication modules, which can also be observed from Fig. 1. This improves upon a fully parallel FFT architecture, such as previous designs [23], requiring k point-wise multiplication modules for a k-point FFT, this proposed design saves a large amount of hardware resources, as the required number of modular multiplication modules is reduced from k to 4.

4.2 Modular Reduction

Two modular reduction modules are used within the proposed hardware design for the FHE encryption step: first, the modular reduction within the FFT and IFFT modules using the modulus p, and secondly, the final modular reduction step using the modulus X_0. The second modular reduction step uses the traditional Barrett reduction method, which requires two multiplications.

The remainder of this subsection introduces the modular reduction module used after the multiplication operation in FFT/IFFT butterfly and point-wise multiplication modules. The selection of a modulus, p, heavily influences the performance of the integer-FFT multiplier. Experimental results carried out in previous work by the authors [23] demonstrate that the multiplier incorporating the Solinas modulus [33], $2^{64} - 2^{32} + 1$, consumes the shortest multiplication time. The base bit-length, b, determines the valid data processing rate, that is, how many bits of useful data are processed when we perform the 64-bit modular p arithmetic. Therefore, we choose a large value for the base unit bit length in our design, i.e., $b = 28$. Larger values cause overflow problems. Using the Solinas modulus, 128-bit multiplicands can be expressed as $x_i = 2^{96}a + 2^{64}b + 2^{32}c + d$, where a, b, c and d are 32-bit numbers. As $2^{96} \equiv -1(\bmod p)$ and $2^{64} \equiv 2^{32} - 1(\bmod p)$, the Solinas modular reduction can be quickly computed as $x_i \equiv 2^{32}(b+c) - a - b + d(\bmod p)$. Since the result, $2^{32}(b+c) - a - b + d$, is within the range ($-p$, $2p$), only an addition, a subtraction and a $3 \rightarrow 1$ multiplexer are needed for the reduction using the Solinas prime.

4.3 The Addition-Recovery and Product-Accumulation Modules

The addition-recovery module, shown in Fig. 4, converts the IFFT outputs back to an integer by resolving a very long carry chain. The product-accumulation module, shown in Fig. 5, is used to combine the block products to form the final multiplication results to be written to memory. As these modules are tightly coupled together in this design, they are described in the same section. The write/read bus bit-width of RAM access is equal to b, because the base unit bit-length is equal to b. For a pipelined design, each RAM port has independent b-bit read and write buses. So the total bit-width to the RAM read and write data ports is equal to $6b$. As the read/write address range of the three adders is equal to the whole range of the multiplication product, the total bit-width of the read and write address bus is equal to $6log_2\frac{n_x+n_y}{b}$.

The upper, middle and lower parts of Fig. 4 are responsible for computing the $Right_{\frac{1}{3}}$, $Middle_{\frac{1}{3}}$ and $Left_{\frac{1}{3}}$ results respectively, and their positions are illustrated in the example in Fig. 2. When the count of inner iterations is less than or equal to 2, the logic can be simplified. The product is generated with the $2b$-bit advance step; thus, the product-accumulation function in these three adders cannot catch up with the speed of the $2b$-bit advance step. Therefore, the 4^{th} adder is required in the low-latency architecture with $2b$-bit width RAM read and write buses as is illustrated in Fig. 5. It must be noted that this module only functions when the data block count of $x \geq 3$. It starts to work in $\frac{k}{2}$ clocks lagging behind the other 3 adders, due to the fact that its advancing step speed is $2b$-bit and so the correct accumulation pipeline can be formed. Therefore, the scheme guarantees that the advancing distance between the first 3 adders and the 4^{th} adder is equal to half of the block product. Finally, when the data block count of x is even, $k/4$ clocks are needed for the 4^{th} adder to complete the final block product result after the first 3 adders finish their job; when the data block count x is odd, $\frac{3k}{4}$ clocks are needed for the 4^{th} adder to complete the final two block product results after the first 3 adders finish. The 4^{th} adder has separate $2b$-bit read and write buses. The read and write address range is also equal to the whole range of the multiplication product. Thus, the total data and address bus bit-width required by the low-latency architecture is $12b + log_2\frac{n_x}{b} + log_2\frac{n_y}{b} + 6log_2\frac{n_x+n_y}{b} + 2log_2\frac{n_x+n_y}{2b}$.

4.4 Latency of Proposed Low-latency Architecture

For a k-point integer-FFT algorithm, each FFT/IFFT module has $log_2 k$ processing stages. Each processing stage contains a butterfly module. Let the number of pipeline stages in a FFT butterfly and an IFFT butterfly be N_F and N_{IF} respectively. Then the latency of the FFT and IFFT butterfly modules can be computed as: $(N_F + N_{IF})(log_2 k - 1) + 2$, where we assume addition only takes 1 clock cycle and the total latency of the two butterflies in the 1st processing stage of FFT and IFFT is equal to 2. As the buffer count of the FFT module is $\frac{k}{2^i}$, where i counts to the maximum number of buffers, and the total buffer count of both the FFT and the IFFT is the same, the latency of all of the buffers in the FFT and IFFT modules is computed as $2\sum_{i=2}^{log_2 k}\frac{k}{2^i}$.

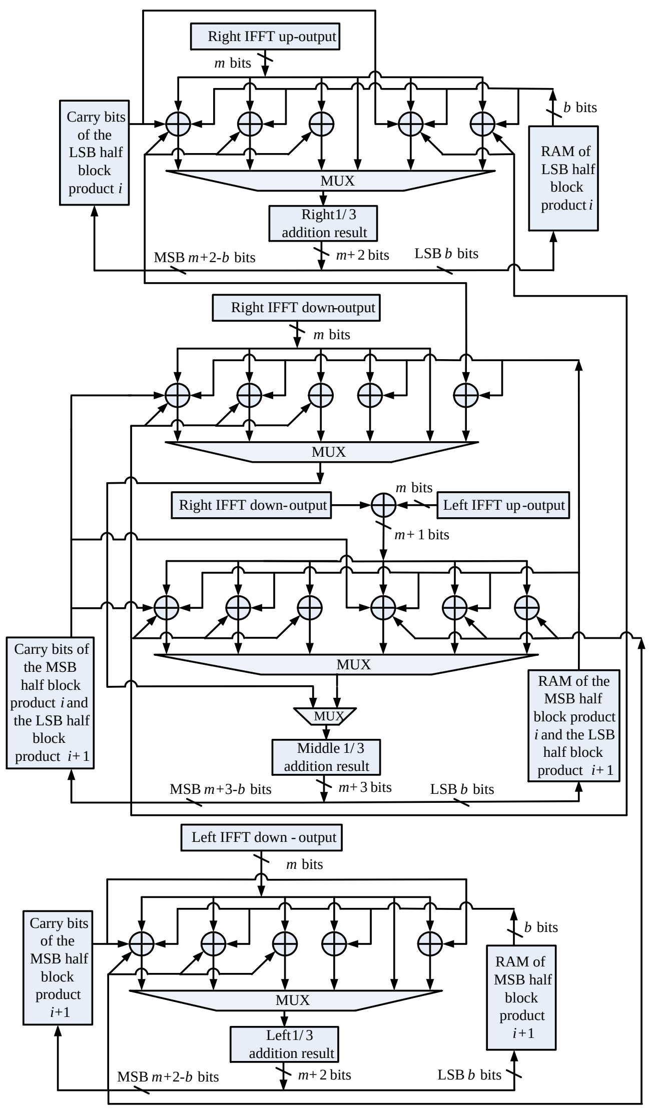

Right IFFT up-output
m bits
b bits
Carry bits of the LSB half block product i
RAM of LSB half block product i
MUX
Right1/3 addition result
MSB m+2-b bits
m+2 bits
LSB b bits
Right IFFT down-output
m bits
MUX
m bits
Right IFFT down-output
Left IFFT up-output
m+1 bits
MUX
Carry bits of the MSB half block product i and the LSB half block product i+1
RAM of the MSB half block product i and the LSB half block product i+1
MUX
Middle 1/3 addition result
MSB m+3-b bits
m+3 bits
LSB b bits
Left IFFT down-output
m bits
b bits
Carry bits of the MSB half block product i+1
RAM of MSB half block product i+1
MUX
Left1/3 addition result
MSB m+2-b bits
m+2 bits
LSB b bits

Fig. 4: The proposed addition-recovery module used in the low-latency architecture

Let the pipeline stage count of the point-wise multiplication module be N_{PW}, then the latency of generating the first $3b$-bit IFFT result (one b-bit is for the $Right_{\frac{1}{3}}$ addition result, the other $2b$-bit is for the $Middle_{\frac{1}{3}}$ and $Left_{\frac{1}{3}}$ addition results) can be computed as:

$$\Delta_0 = 2(\sum_{i=2}^{log_2 k} \frac{k}{2^i}) + (N_F + N_{IF})(log_2 k - 1) + 2 + N_{PW} \quad (2)$$

As our proposed design is a pipelined design, after the first IFFT result is generated, an IFFT result is generated in each subsequent clock. With the k-point IFFT, each inner iteration (i.e. each block product generation) requires $\frac{k}{2}$ clock cycles. Assuming we perform a large integer multiplication, $z = x \times y$, let n_x and n_y be the bit length of the operands x and y respectively. When the data block count is $x \leq 2$, the total inner iteration count in this case is equal to $\lceil \frac{n_x}{\frac{kb}{2}} \rceil \lceil \frac{n_y}{\frac{kb}{2}} \rceil$ where $\frac{kb}{2}$ is equal to the used data bit-length of each operand in a single block product computation. Otherwise, the total count of inner iterations equals

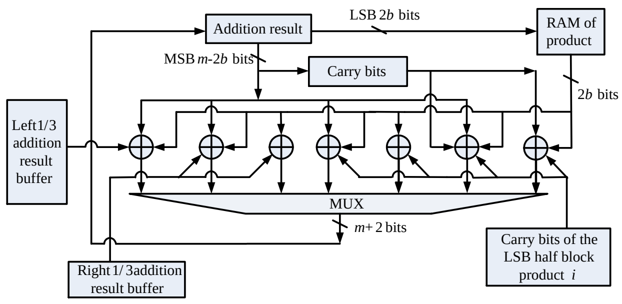

Addition result
LSB 2b bits
RAM of product
MSB m-2b bits
Carry bits
2b bits
Left1/3 addition result buffer
MUX
m+2 bits
Carry bits of the LSB half block product i
Right1/3addition result buffer

Fig. 5: The proposed product-accumulation module used in the low-latency architecture

$$\frac{\lceil \frac{n_x}{\frac{kb}{2}} \rceil - 1}{2} \times \lceil \frac{n_y}{\frac{kb}{2}} \rceil \quad (3)$$

Thus, the total clock cycle count of the first 3 adders is equal to

$$\Delta_1 = \begin{cases} (\lceil \frac{n_x}{\frac{kb}{2}} \rceil \lceil \frac{n_y}{\frac{kb}{2}} \rceil + 1)\frac{k}{2}, & \text{if data block count } x \leq 2 \\ (\lceil \frac{(\lceil \frac{n_x}{\frac{kb}{2}} \rceil - 1)}{2} \rceil \times \lceil \frac{n_y}{\frac{kb}{2}} \rceil)\frac{k}{2}, & \text{otherwise} \end{cases} \quad (4)$$

The latency of the 4th adder for the final product-acumulation is

$$\Delta_2 = \begin{cases} \frac{k}{4}, & \text{when data block count } x \text{ is even} \\ \frac{3k}{4}, & \text{otherwise} \end{cases} \quad (5)$$

Then the total latency of $z = x \times y$ using the proposed low-latency architecture can be estimated as $\Delta_{lowlatency} = \Delta_0 + \Delta_1 + \Delta_2$.

5 The Low Hamming Weight Multiplier Architecture

The low Hamming weight multiplier architecture is used for the multiplication of $X_i \cdot B_i$ required in the encryption step, as defined in Equation (1). The B_i are randomly selected and the Hamming weight (HW) is set to 15 [8]. To the best of the authors' knowledge, currently no such attacks utilising the LHW property are known.

The proposed architecture of the LHW hardware multiplier consists of shared off-chip RAM and a LHW multiplier. As explained in the previous section, the shared RAMs are assumed to be off-chip, and are used to store the input operands, and intermediate and final results. The proposed LHW multiplier is composed of a finite state machine (FSM) controller and a data processing unit. The FSM controller is responsible for distributing the control signals to interface the LHW multiplier and the shared RAM. The essence of our proposed design for computing $z = x \times y$ is to divide x and y into smaller blocks and to determine which blocks of

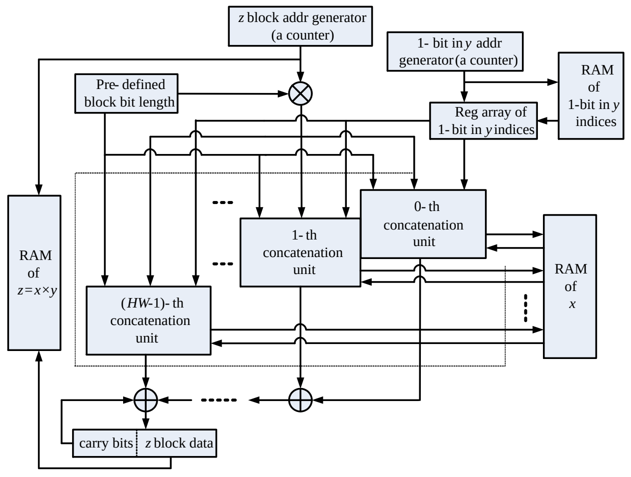

z block addr generator (a counter)
1- bit in y addr generator(a counter)
RAM of 1-bit in y indices
Pre- defined block bit length
Reg array of 1-bit in y indices
RAM of z=x×y
0- th concatenation unit
1- th concatenation unit
(HW-1)- th concatenation unit
RAM of x
carry bits
z block data

Fig. 6: Data Processing Unit in LHW multiplier hardware architecture

x need to be used in the additions, depending on the 1-bit indices of y.

The off-chip RAM is composed of three parts: RAM to store the operand, x, RAM to store the 1-bit indices of the LHW operand, y, and RAM to store the product, z. The operands x and z are stored in normal binary format. However, we propose the use of an encoded data format to store y. More specifically, we only store the set bits in indices of y rather than the entire binary value of y.

We assume that this pre-processing of y is carried out prior to its storage in RAM, which is reasonable as there is significant storage savings. The advantages of using this encoded format for y are that we can easily employ these 1-bit indices of y in our proposed design and it provides savings in terms of memory cost. The latter advantage does not apply for smaller bit lengths of x and y, such as the simple example described later in Section 5.1; however, it will apply when the appropriate FHE parameter bit lengths are used, such as those outlined in Table 1. The bit length of the LHW operand, y, is bounded by 2^{12}. As the required Hamming weight is no more than 15, the required memory cost of an encoded y is bounded by 180 ($= 12 \times 15$) bits, which is much smaller than the original bit length, which ranges from 936 to 2556 bits.

As shown in Fig. 6, the proposed data processing unit requires two address generators, which can be conveniently implemented using binary counters. The first counter is used to address the RAM of 1-bit y indices and it is also used to address the corresponding register array responsible for storing the 1-bit index values received from the RAM. The length of this register array is equal to the Hamming weight of y. The second counter is used to address the RAM that stores the multiplication product, z. For each multiplication it increments from 0 to its maximum value per clock cycle, and then stops. The maximum value of the second counter is the multiplication product bit-length divided by the pre-defined block bit-length.

An array of parallel concatenation units, equal in length to the Hamming weight, is required in our proposed data processing unit to construct the block processing pipeline. Correspondingly, the parallel adder count is equal to the Hamming weight minus 1, and the number of accumulation result registers is equal to the Hamming weight. Each concatenation unit is responsible for generating the required concatenated-value of each inner iteration. The functionality of the concatenation unit is discussed further in Section 5.2.

5.1 Utilising a LHW Operand to Simplify Multiplication

A simple example, as outlined in Fig. 7, is used to illustrate how a LHW operand can be employed to simplify large-integer multiplication.

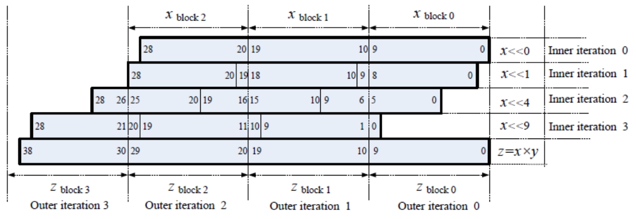


Fig. 7: Utilising a LHW operand to simplify multiplication

In this example, $z = x \times y$, and the bit lengths of the input operands, x and y, are equal to 29 and 10 respectively. Thus, the bit length of the multiplication product, z, is 39. Let the least significant bit (LSB) index of each number be 0. Therefore, the most significant bit (MSB) indices of x and z are 28 and 38 respectively, which can be seen in Fig. 7 (the numbers within the blocks in Fig. 7 are used to denote the bit indices of x and z). Next, we assume that the operand, y, is the LHW operand, and its HW is equal to 4. Let the four 1-bit indices in y be equal to 0, 1, 4 and 9. Therefore, the binary format of y is 1000010011 and the product, z, is equal to the addition of $x \ll 0$, $x \ll 1$, $x \ll 4$ and $x \ll 9$, as shown in Fig. 7, where $\ll$ represents a left shift operation.

In the computation of our proposed design, one block of data is used as the basic processing and computation unit, and we assume that the bit-length of one block is equal to 10. As the bit-length of y is equal to 10, and $log_2 10 \leq 4$, each 1-bit index of y can be represented using a 4-bit binary number. Thus, the encoded data format of y is 1001 0100 0001 0000. Moreover, the block counts of x and z are $3 = \lceil \frac{29}{10} \rceil$ and $4 = \lceil \frac{39}{10} \rceil$ respectively, as depicted in Fig. 2. For example, the bit-ranges of the three blocks of x are [9, 0], [19, 10] and [28, 20] for $x_{block-0,1,2}$.

From Fig. 7, it can be seen that each block of z is equal to the addition of four operands, which are actually four bit-ranges of x, since $x \ll 0, x \ll 1, x \ll 4$ and $x \ll 9$ are easily obtained from x. Assuming the product blocks of z are computed block by block, and the computation is executed serially, the whole multiplication process is composed of four outer iterations and each outer iteration comprises four inner iterations. More specifically, outer iteration-0 (that is, the 0^{th} block of z, z_{block_0}, with bit range [9, 0]) is computed first. The result of this iteration, z_{block_0}, is found by adding the bit-range [9, 0] of x from inner iteration-0, the bit-range [8, 0] of x from inner iteration-1, the bit-range [5, 0] of x from inner iteration-2 and the bit-range [0, 0] of x from inner iteration-3. Using a similar execution flow, the other three blocks of z, $z_{block_{1,2,3}}$, can be obtained. The first resultant block, z_{block_0} does not consider carry bits; however, the addition steps for $z_{block_{1,2,3}}$ need to take account of the carry bits from $z_{block_{0,1,2}}$.

5.2 The Concatenation Unit

The functionality of the concatenation unit involves accurately computing the block addresses and bit-ranges of x that are required in each product block computation. The i-th concatenation unit takes as inputs the i-th 1-bit index of y, the block address of z and the pre-defined block bit-length. Upon receipt of a z block address, the corresponding MSB and LSB indices can be calculated using a multiplication and addition with the block bit-length. Using these indices the required bit-range of x (before shifting left) can be calculated by a subtraction with the i-th 1-bit index of y. Next, the block addresses of two x blocks, denoted as the high and low blocks, can be easily obtained by dividing with the block-bit-length. Then, in order to find the valid bit-ranges of the two x blocks required, the two bit indices of the whole x before shifting and the two bit indices of the required x blocks are subtracted, respectively. Finally, a bit shifting operation, left shifting for the high block and right shifting for the low block, and a concatenation are employed to produce the required concatenated-value.

The computation of z_{block_2} and z_{block_3} in the example defined in Fig. 7 will require concatenation with 0 to the left of the valid bit-ranges of x, that is $0||x[28, 20]$ and $0||x[28, 26]$. It can be observed that the addition operands for z_{block_1} do not need concatenation with 0, as these operands use valid x bit-ranges. This will obviously be the most common setting in the z_{block} computation of the million-bit multiplication used in the CNT FHE encryption step.

To avoid the multiplication and division becoming the performance bottleneck of our proposed architecture, we propose that the pre-defined block-bit-length should be an integer to the power of 2, so that the multiplication and division can be implemented by bit shifting operations.

5.3 Latency of Proposed LHW Architecture

In this section we analyse the RAM access bit-width and the latency of the proposed LHW multiplier architecture. Let the bit-length of the multiplier inputs, x and y, be n_x and n_y respectively and the block bit-length be n_{block}. The Hamming weight of the LHW operand, y, is denoted as $HW(y)$. Thus, the address bit-width of the RAM storing y is equal to $log_2 HW(y)$. As each bit index of y occupies $log_2 n_y$ bits, the data bus bit-width of the y memory is equal to $log_2 n_y$. Therefore the y memory access bit-width, b_y, is equal to

$$b_y = log_2 HW(y) + log_2 n_y \tag{6}$$

As the z RAM is accessed block by block, its data bus bit-width is equal to n_{block}. Since the block count of z is equal to $\frac{n_x+n_y}{n_{block}}$, its address bus bit-width is equal to $log_2 \frac{n_x+n_y}{n_{block}}$. So the RAM z access bit-width is equal to

$$b_z = n_{block} + log_2 \frac{n_x + n_y}{n_{block}} \tag{7}$$

Each concatenation unit has to access two x blocks of n_{block} bits, and $\frac{n_x}{n_{block}}$ blocks, and the number of concatenation units is $HW(y)$. Since all the units access the x RAM in parallel, the data bus-width of the x memory is equal to $2 \times HW(y) \times n_{block}$, and its address bit-width is equal to $2 \times HW(y) \times log_2 \frac{n_x}{n_{block}}$. Therefore, the access bit-width of the RAM storing x is

$$b_x = 2 \times HW(y) \times (n_{block} + log_2 \frac{n_x}{n_{block}}) \tag{8}$$

Thus, the total RAM access bit-width of the proposed architecture is

$$b_{total} = b_x + b_y + b_z \tag{9}$$

The latency of the proposed architecture can be estimated as follows. Assuming the pipeline stage count of the proposed concatenation unit is N, then the time latency until the pipeline is full is equal to

$$\Delta_3 = N + HW(y) \tag{10}$$

After the pipeline is full, a valid n_{block}-bit z block result is generated on every clock cycle. Therefore, the latency from the beginning of the full pipeline to the final result of z appearing is equal to the number of blocks in the product z, that is

$$\Delta_4 = \frac{n_x + n_y}{n_{block}} \tag{11}$$

Hence, the total time to complete the multiplication of $z = x \times y$ using the proposed LHW architecture can be estimated as

$$\Delta_{LHW_{latency}} = \Delta_3 + \Delta_4 \tag{12}$$

6 Implementation, Performance and Comparison

Both of the proposed architectures are designed and verified using Verilog and implemented on a Xilinx Virtex-7 XC7VX980T FPGA device. Modelsim 6.5a was used as the functional and post-synthesis timing simulation tool. The synthesis tool used was Xilinx ISE Design Suite 14.4.

TABLE 2: Synthesis results of the proposed multiplier using the low-latency architecture

Architecture	FFT -Point k	Frequency (MHz)	Slice Registers	Slice LUTs	DSP -48E1s	RAM access bit width
low-latency arch.	256	161.276	39643	54068	544	622
	512	161.276	44599	61668	608	622
	1024	161.276	49538	71060	672	622
	2048	161.453	54795	83930	736	622
	4096	161.453	59996	103909	800	622
	8192	159.499	64765	138175	864	622

6.1 Implementation and Performance of the Low-latency Architecture

The proposed architecture is instantiated with 256, 512, 1024, 2048, and 8192-point FFT. In our implementations, for the small size 64-bit $\times$ 64-bit multiplication used in FFT butterfly and point-wise multiplication, Xilinx Core Generator is employed to automatically generate a 12 stage pipelined multiplier using the embedded DSP48E1 multipliers. The optimal pipeline stage count was found to be 17 for $N_F = N_{IF}$ and 15 for N_{PW} in all implementations. We also set $n_x = n_y = b \times 2^b$, so the bit-length of both input operands is more than 1 Gigabits, which is sufficient for our simulation and verification. The total bit width of the data and address buses in the low-latency architecture is equal to 622 bits.

The synthesis results for the implementations of the low-latency architecture are displayed in Table 2. The proposed 8192-point implementation using the low latency architecture is within the hardware resource budget of the Virtex-7 XC7VX980T: the Slice Register count is only 5%, Slice LUT usage is no more than 22% and the DSP48E1 utilisation is approximately 24%. The RAM access bit width of 622 bits is non-standard. Hence multiple read and writes from RAM per each 622-bit word may be necessary. Thus, the memory interface controller can run in a separate clock domain and at a higher clock frequency to allow multiple read/write accesses per clock cycle with respect to the underlying clock frequency of the proposed design.

TABLE 3: Multiplication operand bit-length required in Equation (1) for multiplication (Type-I) and modular reduction (Type-II)

Type I: Multiplication			Type II: Modular Reduction		
Group	Operand 1 δ	Operand 2 φ	Group	Operand 1 $\delta+\theta$	Operand 2 φ
Toy I	936	150k	Toy II	1094	150k
Small I	1476	830k	Small II	2048	830k
Medium I	2016	4.2m	Medium II	4126	4.2m
Large I	2556	19.35m	Large II	10251	19.35m

The parameters for the multiplications required in Equation (1) for the multiplication-accumulation operation (Type-I) and also within the modular reduction operation (Type-II) are defined in Table 3. We note the unbalanced input operand bit-lengths in the CNT multiplication for the multiplication-accumulation operation required in Equation (1). Thus, we must carefully compare and choose the suitable FFT-point value. Table 4 gives the latency and simulated running times of the multiplications required in the encryption step, defined in Equation (1) for the integer-based FHE scheme [9]. It can be observed from this table that a larger point FFT does not always represent a better performance. For the Large-II group, 1024-point or larger point FFT are the best choice as they consume the least time; for the Medium-II group, 512-point or larger is recommended and for the other groups, the 256-point or 512-point FFT implementations are the best candidates.

It is assumed that there is only one hardware multiplier, and the accumulation operation $\sum_{i=1}^{\theta} X_i \times B_i$ and the Barrett reduction operation $(\cdot) mod X_0$ serially call the same multiplier. The total running time can be determined by the sum of the accumulation time and Barrett reduction time. The accumulation time is equal to the product of a single multiplication time of Type-I, listed in Table 4, and the accumulation iteration count, θ, listed in Table 1. The Barrett reduction time is equal to twice the multiplication time of Type-II, which is listed in Table 4. For example, when the Toy group is implemented using a 256-point FFT design, a single multiplication time of Type-I equals 0.021 ms and a single multiplication time of Type-II is 0.021 ms. Thus the encryption time is $0.021 \times 158 + 0.021 \times 2 = 3.36$ ms.

6.2 Implementation and Performance of Low Hamming Weight Architecture

In the proposed LHW architecture there are two important parameters, the block bit-length, n_{block} and the Hamming weight, HW(y). In our experiment n_{block} is set to 64, 128 and 256, and HW(y) is set to 15. Three groups of experimental results are obtained and presented in Table 5. The hardware resource usage of the proposed LHW architecture is dominated by the number of concatenation units, which is dependent on the value of HW(y). The frequency of the designs decrease with the increase in block bit-length, since the adder carry chain is the performance bottleneck. The RAM access bit-widths given in Table 5 can be calculated using Equation (9), defined in Section 5.3.

TABLE 5: Synthesis results of the proposed LHW architecture

Block bit length	Hamming Weight Weight	Number of Slice Registers	Number of Slice LUTs	Frequency (MHz)	RAM access bit width
64	15	7553	20330	321.97	2589
128	15	11294	41883	252.27	4542
256	15	18929	82711	175.68	8479

Table 6 illustrates the required latency and simulated run time of the CNT FHE multiplication, $X_i \times B_i$ outlined in Equation (1). From Table 6, it can be observed that the latency trend of the CNT multiplication is dominated by the block bit-length, that is, the latency of the designs with a larger block size (256 and 128) is almost half that of the designs with a smaller block size (64). However, the frequency decreases with the increase in the size of the designs. From a time-cost perspective, the designs with a block size of 256 appear to be the best candidates in our implementations.

Recalling Equation (1), the accumulation, $\sum_{i=1}^{\theta} X_i \times B_i$, has no particular LHW property, and therefore the LHW multiplier cannot be used for the multiplications required within the modular reduction, $mod\ X_0$. Thus we adopt

TABLE 4: The latency (clock cycle count) and timings (milliseconds) of a CNT multiplication with the proposed low-latency multipliers, types (I) and (II)

FFT-point	Toy I		Small I		Medium I		Large I	
k	Latency	Time	Latency	Time	Latency	Time	Latency	Time
256	3451	0.021	15611	0.097	75771	0.470	346299	2.147
512	3997	0.025	16157	0.100	76317	0.473	346909	2.151
1024	5183	0.032	17215	0.107	77375	0.480	347967	2.156
2048	7521	0.047	19297	0.120	79713	0.494	350049	2.168
4096	11651	0.072	23939	0.148	84355	0.522	354691	2.197
8192	20901	0.131	33189	0.208	92581	0.580	362917	2.275
FFT-point	**Toy II**		**Small II**		**Medium II**		**Large II**	
k	Latency	Time	Latency	Time	Latency	Time	Latency	Time
256	3451	0.021	15611	0.097	150779	0.935	1037371	6.432
512	3997	0.025	16157	0.100	76317	0.473	692509	4.294
1024	5183	0.032	17215	0.107	77375	0.480	347967	2.158
2048	7521	0.047	19297	0.120	79713	0.494	350049	2.168
4096	11651	0.072	23939	0.148	84355	0.522	354691	2.197
8192	20901	0.131	33189	0.208	92581	0.580	362917	2.275

TABLE 6: Latency (clock cycle count) and Time (milliseconds) of a CNT multiplication using the LHW architecture

Design		Toy		Small		Medium		Large	
Block bit length	Hamming Weight	Latency	Time	Latency	Time	Latency	Time	Latency	Time
64	15	2386	0.007	13019	0.04	65684	0.204	302411	0.939
128	15	1207	0.005	6523	0.026	32856	0.130	151219	0.599
256	15	617	0.004	3275	0.019	16442	0.094	75623	0.43

the low-latency integer-FFT multiplier, proposed in Section 4, as this work does not exceed the resource budget of a Xilinx Virtex-7 FPGA. The time needed to perform a single CNT FHE encryption operation can be calculated by firstly, summing the time serially spent on calling the LHW multiplier to complete $\sum_{i=1}^{\theta} X_i \times B_i$ and secondly, calling the low-latency integer-FFT multiplier to complete $mod\ X_0$.

Table 7 lists the encryption step performance results using the combination of the LHW multiplier and the low-latency integer-FFT multiplier. Alternative multiplication methods could also be used in place of integer-FFT multiplication for low-area designs in future work. The number of DSP48E1s is equal to that required by the integer-FFT multiplier, as the LHW multiplier does not require DSP48E1 resources. The RAM access bit-width is equal to that needed in the LHW multiplier, as the two multipliers are assumed to be serially scheduled. Therefore, the RAM access interface can be shared between them and the cost of the RAM access bit-width for the LHW multiplier is greater. The total running time can be determined by the sum of the accumulation time and modular reduction time. The accumulation time is equal to the product of a single multiplication time, which is listed in Table 6, and the accumulation iteration count, θ, listed in Table 1. The modular reduction adopts the Barrett reduction solution [34], and its time is equal to twice the time required for the low-latency integer-FFT multiplication. For example, in Table 10, when the Toy group is implemented using the proposed LHW multiplier with a single multiplication time of 0.00357 ms and the 256-point integer-FFT implementation with a single multiplication time of 0.021 ms, the encryption time is $0.00357 \times 158 + 0.021 \times 2 = 0.607$ ms. The timings for the other groups in Table 7 can be similarly determined.

6.3 Comparison of Proposed Architectures

Table 8 compares the performance of a single multiplication in CNT FHE encryption. It can be found that the LHW design with the Hamming weight of 15 runs approximately 5 times faster than the proposed low-latency design.

TABLE 8: The time (milliseconds) comparison of a single multiplication in CNT FHE

Group	Toy	Small	Medium	Large
LHW Design	0.00357	0.0189	0.0943	0.434
Low-Latency Design	0.021	0.100	0.473	2.156

The hardware resource cost of the encryption step in CNT FHE for the large parameter sizes for the known implementations is compared in Table 9. To the best of the authors' knowledge, there are no other implementations of the encryption step of this scheme on hardware, except those mentioned in Table 9. It can be seen that although the proposed implementation combines both the LHW multiplier and the integer-FFT multiplier, its resource cost (i.e. number of Slice registers, Slice LUTs and DSP48E1s) is still much smaller than in prior work [22], [23], and is much smaller than the available resource budget of a Xilinx Virtex-7 FPGA. However, the RAM access bit-width of the proposed implementation does exceed the available input/output pin count of a Xilinx Virtex-7 FPGA. Therefore, in this work the pre-synthesis results rather than the post-synthesis results are presented. This issue can be relieved by adding RAM buffers between off-chip RAMs and the on-chip accelerator, and it will be investigated in future work. As previously mentioned, to ensure sufficiently fast read/write operations, the memory access controller can run in a separate clock domain and at a higher clock speed than the underlying design on the FPGA.

TABLE 7: Hardware cost and time (milliseconds) of CNT encryption with LHW multiplier and Barrett reduction

Params	Implementation	Time	Number of Slice Regs	Number of Slice LUTs	Number of DSP48E1s	RAM access bit width
Toy	LHW + 256-point FFT	0.597	58572	136779	544	8479
Small	LHW + 256-point FFT	10.857	58572	136779	544	8479
Medium	LHW + 512-point FFT	198.422	63528	144379	608	8479
Large	LHW + 1024-point FFT	3316.696	68467	153771	672	8479

TABLE 9: The hardware resource cost comparison for large parameter sizes

Design	Frequency (MHz)	Number of Slice Registers	Number of Slice LUTs	Number of DSP48E1s	RAM access bit width
LHW design	175.68 (mult) 161.276 (mod red)	68467	153771	672	8479
Low-latency	161.276	49538	71060	672	622
Prior design using FFT [23]	166.450	1122826	954737	18496	>896
Prior design using Comba [22]	197.758	542979	365315	1536	>192

A comparison of the running times of the CNT FHE encryption primitive using the proposed architectures is given in Table 10. Again, to the best of the authors' knowledge, all of the implementations of integer-based FHE schemes are included in Table 10. As there are few previous implementations of this scheme on hardware, results are compared with the original benchmark software implementation [9]. Compared to the corresponding CNT software implementation [9] on the Intel Core2 Duo E8400 PC, it can be seen that the proposed FPGA implementation with LHW parameters achieves a speed improvement factor of 131.14 for the large parameter group. This table also shows that our proposed LHW implementation requires significantly less than the running time of prior work [22], [23] whilst requiring fewer hardware resources.

TABLE 10: The average running time comparison of the proposed FHE encryption designs

Group	Toy	Small	Medium	Large
LHW design	0.0006s	0.011s	0.198s	3.317s
Low-latency design	0.00336 s	0.05566s	0.9990s	16.595s
Prior design using FFT [23]	0.000739s	0.0132s	0.4772s	7.994s
Prior design using Comba [22]	0.006s	0.114s	2.018s	32.744s
Benchmark software design [9]	0.05s	1.0s	21s	7min 15s

6.4 Discussion and Summary of Previous Hardware Designs for FHE

As previously mentioned in Section 1, there have been several implementations of the Gentry and Halevi FHE scheme [3] on various platforms and with different optimisation goals; however due to the differences between these schemes and their associated parameter sizes, it is misleading to directly compare the performance of these implementations to our implementation of the integer-based FHE scheme. However, to highlight the developments in this field, Table 11 gives a summary of the hardware designs of FHE schemes to date. Table 11 also presents the timing results for the encryption step for several schemes and thus highlights that the proposed combined LHW and NTT multiplier performs comparably to, or indeed, better than hardware designs of other existing FHE schemes. However, it must be noted that this table does not compare the area resource usage of these designs, as the platforms differ greatly. Other work has also looked at the homomorphic evaluation of the block ciphers, AES and Prince using GPUs [20].

TABLE 11: Summary of hardware designs for FHE encryption

Design	Scheme	Platform	Encrypt - small security level
FHE [14]	GH	C2050 GPU	0.22s
FHE [16]	GH	C2050 GPU	0.0063s
FHE [16]	GH	GTX 690 GPU	0.0062s
FHE [18], [19]	GH	ASIC (90 nm)	2.09s
Our LHW FHE encryption step	CNT	Virtex-7 FPGA	0.011s

TABLE 12: Summary of hardware multipliers for FHE

Design	Platform	Multiplier Size (bits)	Multiplier Timings
FHE [14]	C2050 GPU	16384×16384	12.718ms
FHE [16]	GPU	16384×16384	8.835ms
Multiplier [17]	ASIC (90 nm)	768000×768000	0.206ms
Multiplier [18], [19]	ASIC (90 nm)	1179648×1179648	7.74ms
Our FFT multiplier	Virtex-7 FPGA	19350000×2556	2.156ms
Our LHW multiplier	Virtex-7 FPGA	19350000×2556	0.434ms

Several researchers [15], [17], [18] have taken a similar approach to this research and target the large integer multiplier building block. Table 12 shows existing multiplier designs targeted for FHE schemes. It can be seen that our large multiplier design for FHE over the integers performs comparably to that of other multiplier designs. It must be observed that the multipliers included in Table 12 are mainly targeted towards the GH FHE scheme, except the multipliers proposed in this research. Thus, the multiplier size varies greatly between designs and thus, a direct comparison of these designs is unfair. Other research has included the design of a polynomial multiplier [35] for RLWE and SHE encryption targeting the Spartan-6 FPGA platform, where 2040 polynomial multiplications per second are possible for a 4096-bit multiplier. In general, it can be noticed that, whilst great progress has been made in this area, there is still a need for further research to increase the performance of FHE schemes so that they are practical for real time applications.

7 Conclusion

In this paper, novel large integer multiplier hardware architectures using both FFT and low Hamming weight designs are proposed. Firstly, a serial integer-FFT multiplier architecture is proposed with the features of lower hardware cost and reduced latency. Secondly, a novel low Hamming weight (LHW) multiplier hardware architecture is proposed. Both architectures are implemented on a Xilinx Virtex-7 FPGA platform to accelerate the encryption primitive in a fully homomorphic encryption (FHE) scheme over the integers. Experimental results show that the proposed LHW design is approximately 5 times faster than the integer-FFT hardware accelerator on the Xilinx FPGA for a single multiplication required in the FHE scheme. Moreover, when the proposed LHW design is combined with the low-latency integer-FFT design to perform he encryption step, a significant speed up factor of up to 131.14 is achieved, compared to the corresponding benchmark software implementation on a Core-2 Duo E8400 PC. It is clear that FHE over the integers is not yet practical but this research highlights the importance of considering hardware acceleration optimisation techniques in helping to advance the research towards real-time practical performance.

References

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

[2] ——, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178.

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

[4] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *EUROCRYPT*, 2010, pp. 24–43.

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

[6] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *FOCS*, 2011, pp. 97–106.

[7] ——, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *CRYPTO*, 2011, pp. 505–524.

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

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

[10] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *EUROCRYPT*, 2013, pp. 315–335.

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

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 18, p. 111, 2011.

[13] D. Cousins, K. Rohloff, C. Peikert, and R. E. Schantz, "An update on SIPHER (scalable implementation of primitives for homomorphic encRyption)," in *HPEC*, 2012, pp. 1–5.

[14] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Accelerating fully homomorphic encryption using GPU," in *HPEC*, 2012, pp. 1–5.

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

[16] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar, "Exploring the feasibility of fully homomorphic encryption," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[17] W. Wang, X. Huang, N. Emmart, and C. C. Weems, "VLSI design of a large-number multiplier for fully homomorphic encryption," *IEEE Trans. VLSI Syst.*, vol. 22, no. 9, pp. 1879–1887, 2014.

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

[19] ——, "A million-bit multiplier architecture for fully homomorphic encryption," *Microprocessors and Microsystems*, 2014.

[20] W. Dai, Y. Dorz, and B. Sunar, "Accelerating NTRU based homomorphic encryption using GPUs," in *HPEC*, 2014, pp. 1–6.

[21] C. Moore, N. Hanley, J. McAllister, M. O'Neill, E. O'Sullivan, and X. Cao, "Targeting FPGA DSP slices for a large integer multiplier for integer based FHE," in *Financial Cryptography and Data Security - FC 2013 Workshops, USEC and WAHC 2013, Okinawa, Japan, April 1, 2013, Revised Selected Papers*, 2013, pp. 226–237.

[22] C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "Accelerating integer-based fully homomorphic encryption using comba multiplication," in *2014 IEEE Workshop on Signal Processing Systems, SiPS 2014, Belfast, United Kingdom, October 20-22, 2014*. IEEE, 2014, pp. 62–67. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6973465

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

[24] J. Hoffstein and J. H. Silverman, "Random small hamming weight products with applications to cryptography," *Discrete Applied Mathematics*, vol. 130, no. 1, pp. 37–49, 2003.

[25] ——, "Optimizations for NTRU," in *Public-Key Cryptography and Computational Number Theory. Proceedings of the International Conference organized by the Stefan Banach International Mathematical Center Warsaw*, K. Alster, J. Urbanowicz, and H. C. Williams, Eds. De Gruyter, 2000, pp. 77 – 88.

[26] M. McLoone and M. J. B. Robshaw, "New architectures for low-cost public key cryptography on RFID tags," in *ISCAS*, 2007, pp. 1827–1830.

[27] J. H. Cheon and D. Stehlé, "Fully homomophic encryption over the integers revisited," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, 2015, pp. 513–536.

[28] K. Nuida and K. Kurosawa, "(batch) fully homomorphic encryption over integers for non-binary message spaces," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, 2015, pp. 537–555.

[29] V. Shoup. (2014) A Library for doing Number Theory. [Online]. Available: www.shoup.net/ntl/

[30] A. Schönhage and V. Strassen, "Schnelle multiplikation großer zahlen," *Computing*, vol. 7, no. 3-4, pp. 281–292, 1971.

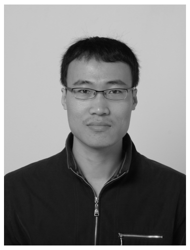
[31] N. Emmart and C. C. Weems, "High precision integer multiplication with a GPU using Strassen's algorithm with multiple FFT sizes," *Parallel Processing Letters*, vol. 21, no. 3, pp. 359–375, 2011.

[32] C. Cheng and K. K. Parhi, "High-throughput VLSI architecture for FFT computation," *IEEE Trans. on Circuits and Systems*, vol. 54-II, no. 10, pp. 863–867, 2007.

[33] J. A. Solinas, "Generalized Mersenne numbers," University of Waterloo, Faculty of Mathematics, Tech. Rep., CORR99-39.

[34] P. Barrett, "Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *CRYPTO*, 1986, pp. 311–323.

[35] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *LATINCRYPT*, 2012, pp. 139–158.



Xiaolin Cao received his Master's degree in Electronics Information and Engineering in Institute of Microelectronics of Chinese Science Academy in 2009, Beijing, China. He received the Ph.D. degree from the School of Electronics, Electronic Engineering and Computer Science at Queen's University Belfast in 2012. He contributed to this work during his post-doctorate research period at CSIT at Queen's. His research interests include cryptographic protocol designs and hardware implementations for RFID and homomorphic encryption. He currently works at Titan-IC Systems in Belfast.

Ciara Moore received first-class honours in the BSc. degree in Mathematics with Extended Studies in Germany at Queen's University Belfast in 2011. She is currently a Ph.D. student in the School of Electronics, Electronic Engineering and Computer Science at Queen's University Belfast. Her research interests include hardware cryptographic designs for homomorphic encryption and lattice-based cryptography.

Máire O'Neill (M'03-SM'11) received the M.Eng. degree with distinction and the Ph.D. degree in electrical and electronic engineering from Queen's University Belfast, Belfast, U.K., in 1999 and 2002, respectively. She is currently a Chair of Information Security at Queen's and previously held an EPSRC Leadership fellowship from 2008 to 2015. and a UK Royal Academy of Engineering research fellowship from 2003 to 2008. She has authored two research books and has more than 115 peer-reviewed conference and journal publications. Her research interests include hardware cryptographic architectures, lightweight cryptography, side channel analysis, physical unclonable functions, post-quantum cryptography and quantum-dot cellular automata circuit design. She is an IEEE Circuits and Systems for Communications Technical committee member and was Treasurer of the Executive Committee of the IEEE UKRI Section, 2008 to 2009. She has received numerous awards for her research and in 2014 she was awarded a Royal Academy of Engineering Silver Medal, which recognises outstanding personal contribution by an early or mid-career engineer that has resulted in successful market exploitation.

Elizabeth O'Sullivan is a Lecturer in CSIT's Data Security Systems group in Queen's University Belfast. She leads research into software security architectures. She holds a PhD in Theoretical and Computational Physics (QUB). She has spent almost 10 years working with industry. She designed embedded software security architectures and protocols, which were licensed to LG-CNS for Electric Vehicle charging infrastructures. She gained extensive experience with Latens Systems Ltd., (Pace UK) in designing and implementing large scale key management systems, secure server infrastructures, security protocols and cryptographic algorithms for embedded platforms. She was the SAP Research UK lead in an FP7 European project on next generation platforms for cloud based infrastructures. She has developed Intellectual Property for Digital Theatre Systems Inc., in the area of signal processing. She is co-investigator of a number of ongoing cyber-security related grants in the UK and collaborative projects with South Korean researchers.

Neil Hanley received first-class honours in the BEng. degree, and the Ph.D. degree in electrical and electronic Engineering from University College Cork, Cork, Ireland, in 2006 and 2014 respectively. He is currently a Research Fellow in Queen's University Belfast. His research interests include secure hardware architectures for post-quantum cryptography, physically unclonable functions and their applications, and securing embedded systems from side-channel attacks.