Low Complexity and Hardware-friendly Spectral Modular Multiplication

Donald Donglong Chen #1, Gavin Xiaoxu Yao #1, Çetin Kaya Koç *2, Ray C.C. Cheung #3

# *Department of Electronic Engineering, City University of Hong Kong*
Tat Chee Avenue, Kowloon, Hong Kong SAR
1 {dlchen2,gavin.yao}@student.cityu.edu.hk
3 r.cheung@cityu.edu.hk

* *University of California Santa Barbara*
2 koc@cs.ucsb.edu

***Abstract*—The Schönhage-Strassen Algorithm (SSA) is an asymptotically fast multiplication algorithm with the complexity of $O(l \log l \log\log l)$ where l is the operand size. It outperforms other multiplication algorithms when l is large enough. One possible usage of such long integer multiplication is for cryptography. Innovated from SSA, the Interleaved Spectral Montgomery Modular Multiplication (ISM^3) algorithm is proposed to accelerate the modular multiplication. ISM^3 algorithm primarily interleaves the Montgomery modular multiplication algorithm between time and spectral (frequency) domain. We show that the tasks in each step of the proposed algorithm have little data dependency, and hence, extremely suitable for hardware implementation. We present the parallel ISM^3 architecture and implement it on Xilinx Virtex-II and Virtex-6 FPGAs. Experimental results show that our 3838-bit ISM^3 is faster than the previous Montgomery multiplier. Moreover, our design can complete a 7678-bit modular multiplication in 3398 cycles in 17.98 μs on a Virtex-6 device.**

I. Introduction

Cryptography provides a way to protect the information's confidentiality, integrity, and authenticity. In order to provide sufficient security level, the key size of the cryptographic systems has been growing continuously. For instance, for the cryptosystem whose security relies on integer factorization problem (e.g. RSA [1]), the recommended key size is 3248 bits for a long term protection for 30 years [2]. This means to utilize RSA to provide a long term protection, one needs to perform 3248-bit modular multiplications, and consequently, a high-performance long integer multiplier is in demand.

The most popular algorithm to perform modular arithmetics to date is Montgomery Modular Multiplication (MMM) [3]. MMM has been extensively studied, and several variants have been proposed for both hardware and software platforms [4]–[8]. MMM has the same order of complexity as a multiplication (In fact, the original MMM [3] involves 3 multiplications). Therefore, the acceleration of the multiplications can also benefit the performance of MMM.

The multiplication algorithms faster than the schoolbook multiplication are listed in Table I. A good summary of these multiplication algorithms is in [9]. The GMP library [10] shows their efficiency in real implementations. However, both [9] and the GMP library [10] only focus on software realization; the hardware realization of these multiplication algorithms is only limited to the schoolbook [6]–[8] and Karatsuba method [11]–[13] to our knowledge.

Table I
Multiplication algorithms and their complexity

Algorithm	Complexity
Schoolbook	$O(n^2)$
Karatsuba [11]	$O(n^{\log 3/\log 2})$
Toom-Cook [14]	$O(n^{\log(2k-1)/\log k})$
Schönhage-Strassen [15]	$O(n \cdot \log n \cdot \log\log n)$
Fürer [16]	$O(n \cdot \log n \cdot 2^{O(\log *N)})$

On the other hand, the Schönhage-Strassen Algorithm (SSA) basically transforms the presentation from time domain to spectral domain (a.k.a. frequency domain) using Fast Fourier Transform (FFT), and performs the multiplication in the spectral domain. We observe that there are extensive parallelisms in SSA multiplication, which can be explored by a parallel architecture on hardware platforms. The parallelization, and consequently, the acceleration may benefit the modular multiplications which used extensively in cryptography.

In this paper, we analyze the characteristics of the SSA arithmetic, and apply its idea in MMM to perform modular multiplications by interleaving the computation between time and spectral domain. The main contributions of this paper are as follows:

- The primitives of Interleaved Spectral Montgomery Modular Multiplication (ISM^3) algorithm is proposed for efficient modular multiplication computation;
- The suitable parameter sets for different operand sizes are provided for fast computations;
- A fast and area-efficient architecture is designed for the proposed algorithm;
- The prototype implemented on FPGA demonstrates that such architecture can finish one 7678-bit modular multiplication in 17.98 μs.

The rest of this paper is organized as follows. Section II introduces the background information. Section III proposes the interleaved spectral Montgomery modular multiplication and analyzes its parameter requirement. In Section IV, the hardware architecture of ISM[3] is designed. Section V provides the FPGA implementation results and the comparison with other works. Section VI concludes this paper.

II. Background

A. Number Theoretic Transform and Spectral Domain

Discrete Fourier Transform (DFT) is a critical operation to transform the signals from time domain to spectral domain for signal analysis and processing. It is usually defined over complex field $\mathbb{C}$. Number Theoretic Transform (NTT) is similar to DFT, but over a finite ring $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ [17]. We still call the transformed domain as spectral domain, and the normal representation as time domain.

In this paper, an integer x in time domain is represented by a polynomial

$$x(t) = x_{s-1}t^{s-1} + \ldots x_1 t + x_0$$

where x_i are the base-b (a.k.a. radix-b) coefficients of x (let $b = 2^{\mu}$, then it is μ-bit coefficient) satisfied $x(b) = \sum_{i=0}^{s-1} x_i 2^{i\mu} \equiv x$. For instance, $x = (\texttt{12345678})_{\mathrm{h}}$ can be represented by a radix-2^8 polynomial as

$$x(t) = (12)_{\mathrm{h}}t^3 + (34)_{\mathrm{h}}t^2 + (56)_{\mathrm{h}}t + (78)_{\mathrm{h}}$$

.

Let $X(k)$ be the polynomial in spectral domain for $x(t)$ after transformation. Let x_i, X_i be their i-th coefficients of $x(t), X(k)$, respectively. The definition of NTT and its inverse (INTT) is given as follows:

Definition 1. *Let ω be a principal d-th root of unity in $\mathbb{Z}_q$. Let $x(t)$ and $X(k)$ be polynomials of degree less than a positive number $d-1$ with coefficient satisfied $0 \leq x_i < b$ for all $i = 0, 1, \ldots, d-1$. The length-d number theoretic transform defined modulo q from $x(t)$ to $X(k)$ and its inverse is defined as*

$$X_i = \mathrm{NTT}_d^{\omega}(x(t))_i := \sum_{j=0}^{d-1} x_j \omega^{ij} \mod q \quad (1)$$

$$x_i = \mathrm{INTT}_d^{\omega}(X(k))_i := d^{-1}\sum_{j=0}^{d-1} X_j \omega^{-ij} \mod q. \quad (2)$$

Pollard [17] proved that the Fast Fourier Transform (FFT) is also available to NTT. The n-th stage decimation-in-time equation for constant geometry FFT [18] is as follows ($0 \leq n \leq \log_2 d - 1$):

$$\begin{cases} X_k = x_{2k} + x_{2k+1}\omega^{P_{nk}} \bmod q \\ X_{k+\frac{d}{2}} = x_{2k} - x_{2k+1}\omega^{P_{nk}} \bmod q \end{cases} \quad (3)$$

where $P_{nk} = \lfloor \frac{k}{2^n} \rfloor \times 2^n$, $k = 0, 1, \cdots, \frac{d}{2} - 1$.

Another characteristic which NTT inherits from DFT is the *cyclic convolution*. Let $X(k) = \mathrm{NTT}_d^{\omega}(x(t))$, $Y(k) = \mathrm{NTT}_d^{\omega}(y(t))$ and $\odot$ be component-wise multiplication, then,

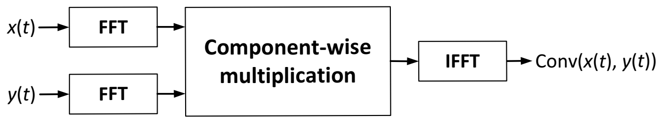

Figure 1. Data flow of Depth-1 Schönhage-Strassen Algorithm.

$$\begin{aligned}\mathrm{conv}(x(t), y(t))_i :=& \sum_{j=0}^{d-1} x_{(i-j) \bmod d} \cdot y_j \\ =& \mathrm{INTT}_d^{\omega}(X(k) \odot Y(k))_i \end{aligned} \quad (4)$$

Note that the existence of NTT and its inverse is not trivial over finite rings. Naussbaumer [19] proved that the cyclic convolution is available to a length-d NTT defined modulo a prime q if and only if $d|(q-1)$. Naussbaumer also stated that if NTT is defined modulo a composite number q, it supports cyclic convolution if and only if the following conditions are met:

- $\omega^d \equiv 1 \mod q$;
- $dd^{-1} \equiv 1 \mod q$;
- ω^e and q are co-prime for every integer e such that d/e is a prime.

B. Schönhage-Strassen Algorithm

The basic idea of SSA is to use FFT and IFFT to transform the representations back and forth between time domain and spectral domain, and to perform the multiplication in spectral domain as shown in Figure 1. Because the complexity of FFT and IFFT is $O(d \log d)$ and that of multiplications in spectral domain is $O(d)$, the entire complexity is sub-quadratic. This technique is used recursively in the component-wise multiplications in SSA.

In order to accelerate the operations involved in SSA, the parameters to construct the ring $\mathbb{Z}_q$ are of special form.

- q is Fermat number of form $2^v + 1$, where v is a power of 2
- ω is a power of 2
- d is a power of 2

Since q is in the form of $2^v + 1$, there exists fast modular reduction arithmetic as shown in Algorithm 1. The choice of ω ensures that a multiplication by ω can be simply achieved by shift operation. The selection of d as a power of 2 makes the FFT structure available.

Algorithm 1 $A \bmod q$ where $q = 2^v + 1$

```
Input: $A, q$ where $q = 2^v + 1$
Output: $A \bmod q$
1: while $A > q$ or $A < 0$ do
2:     $A \leftarrow (A \bmod 2^v) - \lfloor A/2^v \rfloor$
3: end while
4: return $A$
```

Note that for polynomial multiplication, the result is the linear convolution of the coefficients, not the cyclic convolution.

In order to have enough dynamic range such that the cyclic convolution is equivalent to linear convolution, SSA has the following requirements for the operand to avoid overflow [20]:

Theorem 1. *Suppose that $x(t)$ and $y(t)$ are base-b polynomials with degree $s = d/2$. The product $z(t) = x(t)y(t)$ can be computed by SSA without overflow when $q > sb^2$.*

C. Montgomery Modular Multiplication

The modular multiplication is one of the basic arithmetic for public-key cryptography [21], and the efficiency to perform such modular multiplication has a direct impact on the performance of the cryptosystem. There exists fast modular reduction arithmetic when the modulus is of a special form, e.g. Fermat/Mersenne number [22]. However, such fast arithmetic is not always available when the modulus n required by the cryptographic protocols has no specific form. On the other hand, Montgomery modular multiplication [3] can avoid the trivial division to accelerate such modulo-n multiplication. The MMM without conditional subtraction [5] is given in Algorithm 2.

Algorithm 2 Montgomery Modular Multiplication [5]

Copirme integers n and r, $r > 4n$, $x' \equiv xr \bmod n$, $x' < 2n$, $y' \equiv yr \bmod n$, $y' < 2n$, $n' \equiv -n^{-1} \bmod r$.

Input: x', y', n, n', and r
Output: $z' \equiv x'y'r^{-1} = xyr \bmod n$, $z' < 2n$

```
1: $t \leftarrow x' \cdot y'$
2: $m \leftarrow (t \bmod r)n' \bmod r$
3: $z' \leftarrow (t + mn)/r$
4: return $z'$
```

While MMM performs modular multiplication in time domain, Saldamlı and Koç proposed the spectral modular multiplication [23], which performs the multiplication and modular reduction in the spectral domain. Essentially, the Saldamlı and Koç spectral modular multiplication is derived from the digit-serial Montgomery multiplication [4], and as its name indicated, such algorithm is essentially sequential, thus not friendly for massive parallel computation.

III. Interleaved Spectral Montgomery Modular Multiplication and its Parameter Selection

A. Interleaved Spectral Montgomery Modular Multiplication

The SSA is only preferred for software implementation when the operands are long enough. For instance, on a 64-bit Intel Core 2 processor, the SSA is not faster than other algorithms until the operand is 303,104 bits [10]. This is because the advantage of $O(d \log d)$ is not obvious compared to $O(d^2)$ when d is small, but SSA introduces extra additions and shifts.

On the other hand, the SSA's idea is more friendly to hardware implementation on FPGA or ASIC; the overhead of modulo-q operation can be hidden in pipeline, and the shift on hardware platform is just routing, which is negligible. More importantly, the component-wise multiplications and the operations in each stage of FFT/IFFT have no data dependency, and consequently, parallelization can accelerate the computation.

Algorithm 3 Interleaved Spectral Montgomery Modular Multiplication

Suppose that there exists a length-d NTT for some principal root of unity ω in $\mathbb{Z}_q$. Let $s = d/2$, $r = 2^{s\mu}$, $b > 0$, $gcd(b, n) = 1$, $x, y < 2n$ and $n' = -n^{-1} \bmod r$. Let $x(t)$, $y(t)$, $n(t)$, and $n'(t)$ be the time polynomial of x, y, n, and n', which satisfied $x(b) = x$, $y(b) = y$, $n(b) = n$, and $n'(b) = n'$. $X(k)$, $Y(k)$, $N(k)$, and $N'(k)$ are the spectral polynomials of $x(t)$, $y(t)$, $n(t)$, and $n'(t)$, respectively.

Input: $X(k)$, $Y(k)$, $N(k)$, $N'(k)$, q and r
Output: $Z(k) = \mathrm{FFT}(z(t))$ where $z = xyr \bmod n$

```
1: $T(k) \leftarrow X(k) \odot Y(k) \bmod q$
2: $t \leftarrow \mathrm{IFFT}(T(k))$
3: $g \leftarrow t \bmod r$
4: $G(k) \leftarrow \mathrm{FFT}(g)$
5: $M(k) \leftarrow G(k) \odot N'(k) \bmod q$
6: $m \leftarrow \mathrm{IFFT}(M(k))$
7: $m \leftarrow m \bmod r$
8: $M(k) \leftarrow \mathrm{FFT}(m)$
9: $K(k) \leftarrow M(k) \odot N(k) \bmod q$
10: $Z(k) \leftarrow T(k) + K(k) \bmod q$
11: $z \leftarrow \mathrm{IFFT}(Z(k))$
12: $z \leftarrow z/r$
13: $Z(k) \leftarrow \mathrm{FFT}(z)$
14: return $Z(k)$
```

However, the modulo-r reduction and division-r operation in Algorithm 2 are not trivial in spectral domain. But such operations can be simply performed in time domain by bit selection because r is chosen as a power of 2. Therefore, in order to optimize the performance, we propose to interleave the long integer multiplication in spectral domain, and the modular reduction and division in time domain. The Interleaved Spectral Montgomery Modular Multiplication (ISM^3) algorithm is shown in Algorithm 3. In Algorithm 3, the FFT and IFFT are in demand to transform the representation back and forth between time and spectral domain.

The ISM^3 is less complex than 3 SSA multiplications when the modulus n is fixed. This is because $N'(k)$ and $N(k)$ only need to be computed once. Also, for exponentiation using the MSB-first square-and-multiply algorithm, it is either multiplication by a fixed number or squaring, and only one multiplicand needs FFT. Therefore, there are 6 FFT/IFFT in ISM^3 instead of 9 in 3 SSA multiplications.

B. Parameter Selection

Before we choose parameters, we first identify the boundaries for the parameters to yield correct results in ISM^3. The following corollary is derived from Theorem 1 and [5], [23] . The detailed proof is skipped due to the limit of page length.

Table II
PARAMETER SELECTION FOR SPECTRAL MODULAR EXPONENTIATION BY USING ISM^3

Bits l	Ring $\mathbb{Z}_q$	NTT length d	Root ω	Wordsize μ	Words s
206	$2^{32}+1$	32	4	13	16
414	$2^{32}+1$	64	2	13	32
926	$2^{64}+1$	64	4	29	32
1790	$2^{64}+1$	128	2	28	64
3838	$2^{128}+1$	128	4	60	64
7678	$2^{128}+1$	256	2	60	128

Corollary 1. *ISM^3 can apply to compute modular exponentiation without overflow if the parameters s, b and q satisfied*

$$2sb^2 < q \tag{5}$$

while modulus n should satisfy the following inequality,

$$n < 2^{s\mu-2}. \tag{6}$$

To make sure that the length d of NTT/INTT enable the regular FFT/IFFT, the best choice for q is Fermat numbers $F_t = 2^{2^t}+1$. It is proved in Section 8.3 of [19] that when F_t is composite ($t \geq 5$), one can always define NTT modulo $q = F_t$ of length $d = 2^{t+1-i}$ with root $\omega = 2^{2^i}$ where i is integer. This indicates that for a fixed Fermat number $q = F_t$, a smaller root ω is accompany by a longer length d. In order to have a larger operand size of ISM^3 with a fixed q, we usually set the value of ω small.

We summarize the parameter specifications for ISM^3 as follows:

1) q is Fermat number of form 2^v+1 where $v = 2^t$.
2) ω, the d-th principle root of $\mathbb{Z}_q$, is a power of 2.
3) d is a power of 2 for FFT, and $s = d/2$.
4) The greatest $b = 2^\mu$ such that $2sb^2 < q$.
5) $l = \mu s - 2$, where l is the maximum operand size of n.

The selection of parameters begin with the determination of ring $\mathbb{Z}_q$. With a fixed q, one can find the NTT length d and its correspondent root ω. After we get $s = d/2$, the largest $b = 2^\mu$ that satisfied Inequality (5) can be found. The maximum operand size l of ISM^3 can then be determined. Following the selection method above, we carefully select six example parameter sets for spectral modular exponentiation by using ISM^3 as shown in Table II. For a larger l, v will be larger, and one can use the SSA recursively for the component-wise multiplications. Therefore, the ISM^3 follows the complexity of SSA, which is $O(l \log l \log\log l)$.

C. Fast Computation for Modulo-r and Division-r Operations

For the IFFT computation in Algorithm 3, we need to accumulate the coefficients and generate the time domain integer by using the following equation

$$z = z(b) = z_{d-1}b^{d-1} + \dots z_1 b + z_0.$$

After IFFT produces the time polynomial, it shifts and accumulates the coefficients to generate the time domain integer. Since $b = 2^\mu$ is a power of 2, the whole accumulation can be easily accomplished by shift and add in hardware. The shift bias of each coefficient is a multiple of μ, one can separate each coefficient z_i into 3 μ-bit trunks z_{i2}, z_{i1}, z_{i0} and accumulate them as shown in Figure 2.

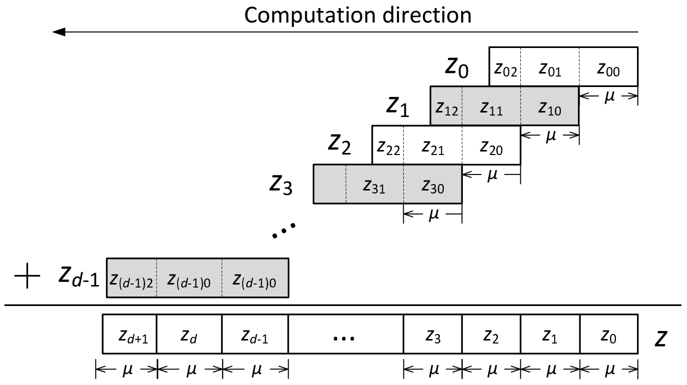

Figure 2. Generate time domain coefficients using serial adders.

The modulo-r operations in Step 3 and 7 of Algorithm 3 can be accomplished simply by selecting the s coefficients with lower degrees. Therefore, half of the accumulation time can be saved by omitting the accumulation process for the other coefficients.

Similarly, for the division-r operation, we can simply perform by eliminating the computation of the s coefficients with lower degrees. However, the result z in Step 12 of ISM^3 may have an ϵ difference from the correct result. This is because the carry from the lower degree coefficients is missed. Note that MMM algorithm guarantees that $(t+mn)$ is divisible by r, thus, the least significant $s\mu$-bit of z in Step 11 of ISM^3 should all be zero. Using this characteristic, we can determine ϵ from the summation of the two most significant bits of trunks $z_{(s-1)0}$ and $z_{(s-2)1}$, i.e. $\delta = \lfloor 4z_{(s-1)0}/b \rfloor + \lfloor 4z_{(s-2)1}/b \rfloor$:

$$\epsilon = \left\lceil \frac{\lfloor 4z_{(s-1)0}/b \rfloor + \lfloor 4z_{(s-2)1}/b \rfloor}{4} \right\rceil. \tag{7}$$

This is because the carry-in from the lower bits could only be 0, 1, or 2, and $\delta \bmod 4$ must be 0. Therefore, a look-ahead logic could accelerate the computation of the corrected division-r result.

IV. ARCHITECTURE FOR ISM^3

A. Architecture Overview

It can be observed that ISM^3 algorithm has the same operation combination throughout the algorithm (component-wise multiplication, IFFT, mod/div, and FFT) by 3 times. Thus, an area saving architecture which supports this operation combination can be designed by reusing the architecture under data flow control. Furthermore, since the data flow of FFT and IFFT are similar, they can share the same computation resources. The top level architecture of ISM^3 is shown in Figure 3. The multiplier and FFT/IFFT module are working in pipeline.

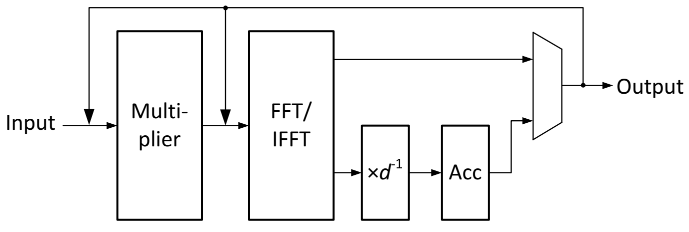

Figure 3. The top level architecture of ISM^3.

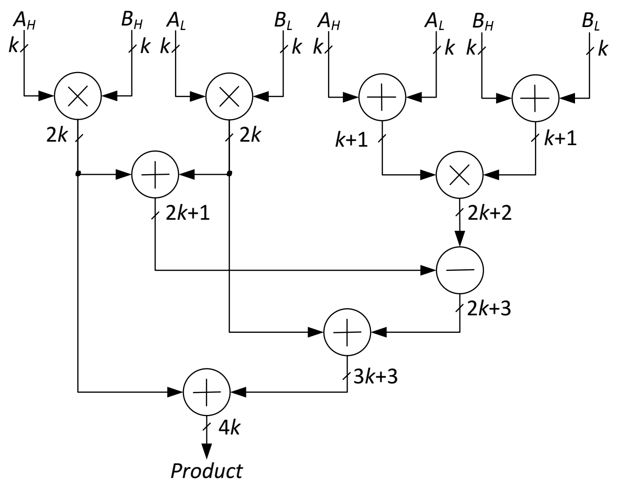

Figure 4. Architecture of a $2k$-bit Karatsuba Multiplier. Registers are omitted for simplicity

B. Modulo-q Arithmetic

Modulo-q reduction is in great demand in ISM^3, hence a fast computation can speedup the whole system significantly. Redundant number representation can be utilized in the computation. If $|A| < 2^{2v}$, one time computation of Step 2 in Algorithm 1, the result will stay in the range $[-2^v+1, 2^v-1]$. Thus, the residue only needs $(v+1)$-bit for storage including a sign bit. This operation can be finished by one adder in one cycle, thus we use this method in most of the modulo-q cases in the system. The overhead is that the following operators should support signed operations. Therefore, using the redundant number representation can save the correction step with negligible overhead. The signed coefficients are corrected to positive integers before the accumulation in IFFT. Therefore, the data range for storage is $[-2^v-1, 2^v]$, and the memory data-width is $(v+2)$-bit.

The coefficients of the polynomial are needed to be stored during the ISM^3 computation, and the total size of the coefficients is $(v+2)\cdot d$, which is quite a big size when q and d is large. Therefore, these coefficients are better stored in Memory blocks rather than registered on the fly.

C. Karatsuba Multiplier

We use Karatsuba's method [11] [13] to optimize the construction of the $(v+2)$-bit multiplier. To explain the Karatsuba method, we let A and B be two $2k$ bits integer and divide them

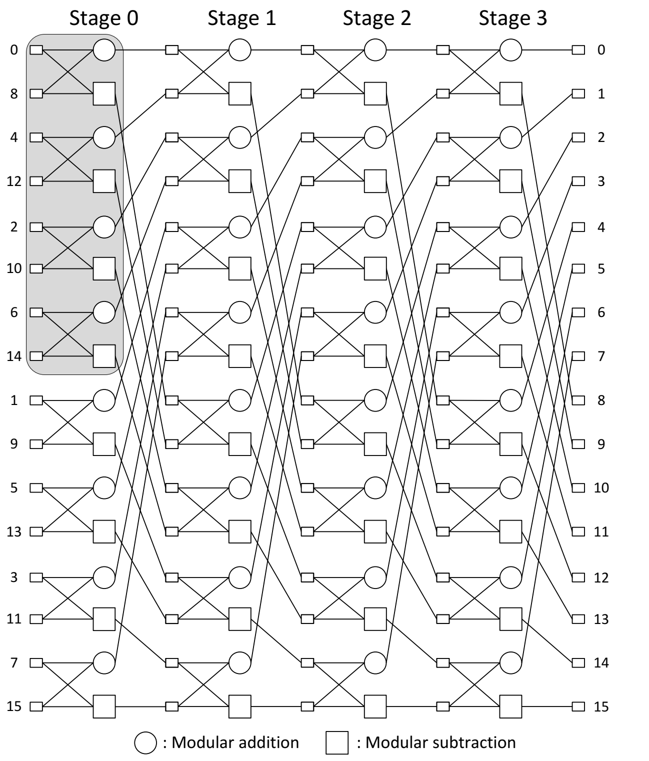

Figure 5. Example of a length-16 constant geometry FFT. The shadow area is used as the primitive for the sub-stage FFT/IFFT architecture design.

into higher half and lower half as follows

$$A = 2^k A_H + A_L,\; Y = 2^k B_H + B_L.$$

Multiplication of A and B in traditional way require four k-bit multiplication and a few additions as shown below

$$AB = 2^{2k}c_2 + 2^k c_1 + c_0$$

where

$$\begin{aligned} c_2 &= A_H B_H \\ c_1 &= A_L B_H + A_H B_L \\ c_0 &= A_L B_L. \end{aligned}$$

Karatsuba found that c_1 can be computed by using only one multiplication with a few additions and subtraction as can be shown below

$$c_1 = (A_H + A_L)(B_H + B_L) - (c_2 + c_0).$$

Therefore, to compute a $2k$-bit multiplication, instead of the traditional four k-bit multiplication, one only needs two k-bit and one $(k+1)$-bit multiplication.

The hardware architecture of the Karatsuba multiplier is shown in Figure 4. In order to use less registers and shorten the output delay, we parallel the computation of $(A_H+A_L)(B_H+B_L)$ with (c_2+c_0). Instead of compute $2^k c_1 + c_0$ by shift and add, we can achieve the same goal by adding only the high half of c_0 to c_1. Similar approach can be use for the computation of $2^{2k}c_2 + (2^k c_1 + c_0)$. This method can reduce the usage of shift registers as well as shorten the carry chain of adders.

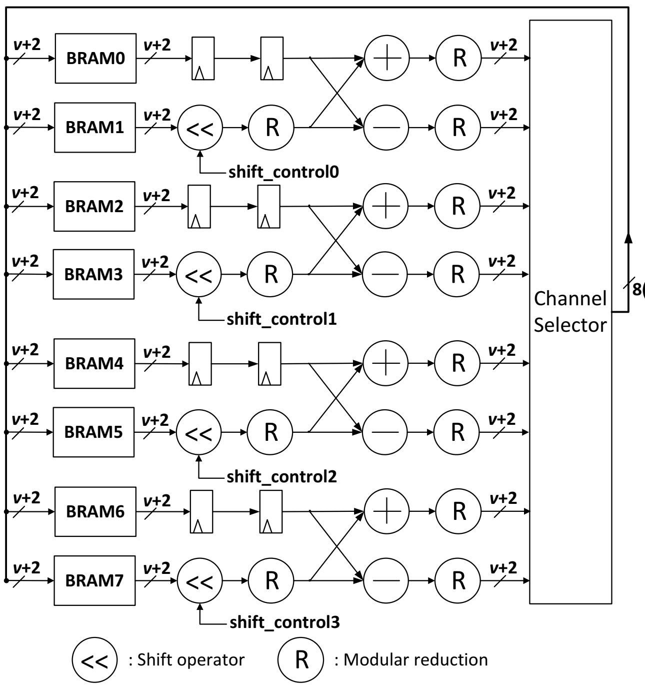

Figure 6. Detailed architecture of constant geometry sub-stage FFT/IFFT. $shift\_controlX$ signals control the bit shift operation of the shift operators.

D. Architecture for FFT/IFFT

To shorten the computation clock cycles of ISM^3, one can design a FFT/IFFT architecture to make all the d coefficients compute in parallel. However, the trade-off is that this architecture will cost a large amount of hardware resources. Furthermore, long routing of the large design will drag down the operating frequency of the whole system. To balance the number of computation clock cycles with the operating frequency, a pipelined architecture is preferred.

We use constant geometry architecture [18] for an area-compressed pipelined FFT/IFFT design [24]. Figure 5 shows an example architecture of a length-16 constant geometry FFT. As shown in Figure 5, the architecture of each stage in the FFT computation shares the same pattern of data path, thus we can reuse this architecture for all the stages. Moreover, we can further shrink the size of the architecture and only build one sub-stage FFT architecture as shown in the shadowed area in Figure 5. A pipelined architecture of this sub-stage FFT can be reused for the whole FFT computation.

The proposed pipelined architecture of sub-stage FFT/IFFT is shown in Figure 6. Note that the registers after the operators are omit for a clear view. It has 4 butterfly structures and can handle 8 inputs at each cycle. Shift operators is responsible for multiplying $\omega^{P_{nk}}$ to x_{2k+1} in equation 3.

E. Memory Management

Simple dual-port Block RAM (BRAM), which has one dedicated read port and one dedicated write port, is used for coefficients storage. An example of coefficients storage situation in BRAMs for a length-64 FFT/IFFT is depicted

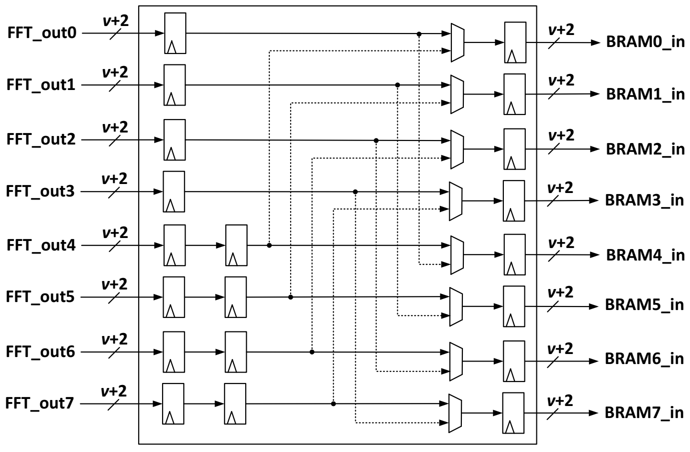

Figure 7. Architecture of Channel Selector. Control signals are omitted for simplicity.

in Figure 8. Since our design attempt to compute eight coefficients at each time, each eight successive coefficients are stored in the BRAMs with the same address. With this design, eight successive coefficients can be read simultaneously under the control of the same address signal.

The write control is more complex because after each stage of FFT/IFFT, the order of the coefficients is shuffled. The input/output sequence for a length-64 FFT/IFFT is shown in Figure 9. For instance, in Time 0, Coefficients 0–7 is fetched from BRAMs 0–7. After i cycles, the FFT/IFFT module finishes the processing. However, the outputs are Coefficients x_0–x_3 and Coefficients x_{32}–x_{35}. Note that both Coefficients x_0–x_3 and Coefficients x_{32}–x_{35} belong to BRAMs 0–3, and a direct write back will produce a collision. The similar situation also happens to the other outputs.

In order to avoid collisions and pipeline bubbles caused by the coefficient storage, a channel selector is designed as Figure 7. The design of the channel selector is based on the observation that the 4 outputs with lower orders at Time $i + j$ can be restored with the 4 outputs with higher orders at Time $i + j - 1$ simultaneously (shown as bounding box in Figure 9). With the architecture shown in Figure 7, the channels represented by the dash lines can interleave the outputs with the channels in the solid lines.

If the pipeline depth is relatively small for FFT/IFFT structure, a racing problem will occur. For example, at Time i+0, Coefficients x_{32}–x_{35} is not read out of BRAMs, while the new Coefficients x_{32}–x_{35} have already come. Since the address for these coefficients in BRAMs is overlapped, the storage of the new coefficients will overwrite the coefficients of current stage. This problem occurs in the 7678-bit ISM^3 design. There are two ways to tackle this problem: using FIFOs to lengthen the pipeline depth, or adding one more set BRAMs for using ping-pong alternative storage mechanism. Observed that the total stage number of 7678-bit ISM^3 is an even number ($\log_2 d = 8$). If using the second method, the control signal for the ping-pong storage is simply the Least Significant Bit

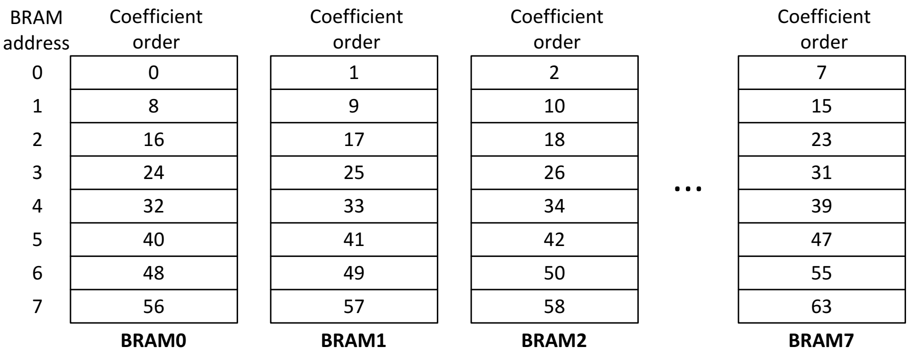

Figure 8. An example of a length-64 FFT/IFFT coefficients storage situation in BRAMs.

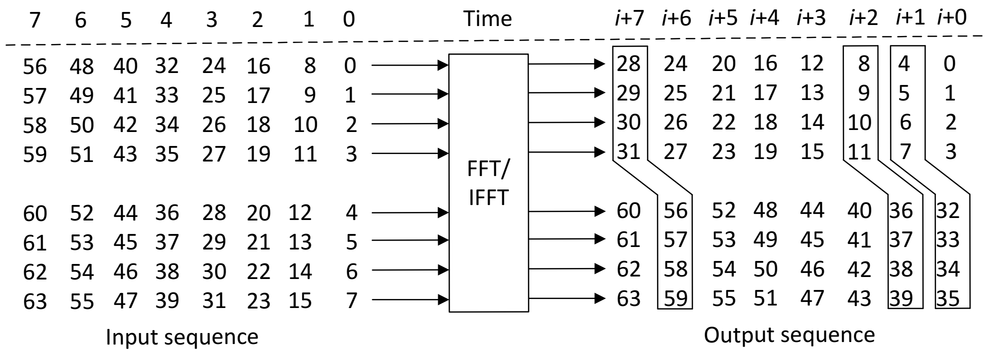

Figure 9. Input and output orders of a length-64 FFT/IFFT. Eight successive coefficients are inputed at each clock cycle. After i (the pipeline stage of FFT/IFFT) clock cycles the coefficients will be outputted.

(LSB) of the stage counter. Moreover, different with the first method, the second method will not introduce pipeline bubbles into the computation. Thus the second method is used in our 7678-bit ISM^3 design.

V. Implementations and Performance Comparisons

We implemented the proposed architectures on a Xilinx Virtex-6 (xc6vlx130t) FPGA. Note that each DSP48E1 in Xilinx Virtex 6 device has a 18-bit signed multiplier, we use it as the base multiplier to build the $(v+2)$-bit Karatsuba pipelined multiplier. The base multipliers in DSP48E1s is pipelined with 3 stages in order to increase the frequency. Registers are used both for the input and output of each operation. The delay of the 34-bit, 66-bit, and 130-bit Karatsuba multipliers are 7 cycles, 11 cycles, and 15 cycles, respectively. In our design, the 66-bit and 130-bit Karatsuba multipliers uses only 9 and 27 DSP48E1s instead of 16 and 64 by using the traditional approach, respectively.

We describe the ISM^3 architectures in Verilog-HDL and use Xilinx ISE 13.3 to synthesize and place & route. The 7678-bit and 3838-bit ISM^3 have the same $q = 2^{128} + 1$, and therefore, the data-width for these two designs are the same. One interesting point is that the 7678-bit ISM^3 use approximately 2% less LUTs than the 3838-bit design. This is because in the 7678-bit design, the number of FFT stages is even, and we can simply use the LSB of the stage counter for BRAMs set selection. However, the controller for 3838-bit design is more complex.

In order to have a fair comparison with the previous works, we also implemented our 3838-bit ISM^3 on a Xilinx Virtex-II (xc2v6000) device using Xilinx ISE 10.1. We compare our work with the famous Montgomery modular multiplication architectures from [6]–[8]. As shown in Table IV, using similar hardware resources, the architecture of McIvor [7] can only compute 2048-bit modular multiplication while our design can achieve a maximum operand size of 3838-bit with approximately 6% reduction on computation time on average.

The architecture by Tenca and Koç [6] is implemented on FPGA by Huang *et al.* [8]. Compared with the 3072-bit Tenca-Koç MMM, our 3838-bit ISM^3 use more hardware resources but the latency is shorten by nearly 60%. We also have approximate 30% speedup when compared with Huang's MMM [8]. The speedup mainly comes from the low complexity of the proposed ISM^3 algorithm and the pipelined design, which reduces the clock cycles of the computation significantly.

VI. Conclusions

We propose the Interleaved Spectral Montgomery Modular Multiplication algorithm in this paper, which combines both the Schönhage-Strassen algorithm and the Montgomery modular multiplication algorithm. The proposed low complexity algorithm is friendly for hardware implementations.

Table III
VIRTEX 6 IMPLEMENTATION RESULTS AND COMPARISON OF THE INTERLEAVED SPECTRAL MONTGOMERY MULTIPLIERS

Max. operand size (bit)	LUTs	Slice	DSP48E1s	RAMB36E1s/ RAMB18E1s	Cycles	Period (ns)	Latency (μs)
926	9,355	3,055	9	19/12	1,046	4.49	4.70
1,790	9,419	2,986	9	19/12	1,775	4.58	8.13
3,838	20,759	6,779	27	46/4	1,787	5.20	9.29
7,678	20,347	6,607	27	46/4	3,398	5.29	17.98

Table IV
HARDWARE RESOURCE AND PERFORMANCE COMPARISON OF MODULAR MULTIPLIERS ON XILINX VIRTEX-II FPGA

Max. operand size (bit)	Architecture	Hardware resource	Cycles	Period (ns)	Latency (μs)	Speedup (%)
2,048	[7] (5 to 2)	20,986 slices	-	11.10	22.76	6.6
2,048	[7] (4 to 2)	23,108 slices	-	11.02	22.59	5.9
3,072	[6]'s architecture from [8]	19,110 LUTs	6,336	8.30	52.59	59.6
3,072	[8] (radix-2, ω=16)	16,331 LUTs	3,264	9.55	31.17	31.8
3,072	[8] (radix-2, ω=32)	15,197 LUTs	3,168	9.76	30.94	31.3
3,838	Our design	21,477 slices / 26,021 LUTs	1,787	11.89	21.25	-

The parameter requirements of the proposed algorithm are analyzed. We also summarize the approach for parameter set selection. Four ISM^3 designs with different operand sizes are implemented on FPGA. The experimental results show that our 3708-bit and 7420-bit ISM^3 designs are faster than the previous Montgomery modular multipliers.

For future works, we will refine the architecture design, and speedup the accumulation process in IFFT. We will also investigate how the ISM^3 arithmetic can benefit other public-key cryptosystem such as elliptic curve cryptosystem.

VII. ACKNOWLEDGEMENT

This work was supported by research grants from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. CityU 9041799), and the CityU Strategic Research Grant (Project No. CityU 7008185).

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] N. Smart, "ECRYPT II yearly report on algorithms and keysizes (2010–2011)," ECRYPT II, Tech. Rep. ICT-2007-216676, 2011.

[3] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[4] Ç. K. Koç, T. Acar, and S. B. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, 1996.

[5] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[6] A. Tenca and Ç. K. Koç, "A scalable architecture for modular multiplication based on Montgomery's algorithm," *Computers, IEEE Transactions on*, vol. 52, no. 9, pp. 1215–1221, 2003.

[7] C. McIvor, M. McLoone, and J. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," in *IEE Proceedings – Computers and Digital Techniques*, vol. 151, no. 6, 2004, pp. 402–408.

[8] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for Montgomery modular multiplication algorithm," *Computers, IEEE Transactions on*, vol. 60, no. 7, pp. 923–936, 2011.

[9] D. E. Knuth, *The Art of Computer Programming, volume 2, Seminumerical Algorithms, 3rd edition*. Addison Wesley, 1998.

[10] "The GNU Multiple Precision Arithmetic Library (GMP)." [Online]. Available: www.gmplib.org

[11] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, no. 7, pp. 595–596, 1963.

[12] G. Chow, K. Eguro, W. Luk, and P. Leong, "A Karatsuba-based Montgomery multiplier," in *Field Programmable Logic and Applications – FPL 2010*, 2010, pp. 434–437.

[13] M. K. Jaiswal and R. C. C. Cheung, "Area-efficient architectures for large integer and quadruple precision floating point multipliers," in *Field-Programmable Custom Computing Machines – FCCM 2012*, 2012, pp. 25–28.

[14] S. A. Cook, "On the minimum computation time of functions," Ph.D. dissertation, 1966.

[15] A. Schönhage and V. Strassen, "Schnelle multiplikation großer zahlen," *Computing*, vol. 7, pp. 281–292, 1971.

[16] M. Fürer, "Faster integer multiplication," in *The thirty-ninth annual ACM symposium on Theory of computing – STOC 2007*. ACM, 2007, pp. 57–66.

[17] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of Computation*, vol. 25, pp. 365–374, 1971.

[18] M. Pease, "An adaptation of the fast Fourier transform for parallel processing," *Journal of the Association for Computing Machinery*, vol. 15, pp. 252–264, 1968.

[19] H. J. Naussbaumer, *Fast Fourier transform and convolution algorithms*. Springer, Berlin, Germany, 1982.

[20] G. Saldamlı, "Spectral modular arithmetic," Ph.D. dissertation, Oregon State University, May 2005.

[21] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[22] R. Zimmermann, "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," in *Computer Arithmetic – ARITH 1999*, 1999, pp. 158–167.

[23] G. Saldamlı and Ç. K. Koç, "Spectral modular exponentiation," in *Computer Arithmetic – ARITH 2007*, 2007, pp. 123–132.

[24] G. X. Yao, R. C. C. Cheung, Ç. K. Koç, and K. F. Man, "Reconfigurable number theoretic transform architectures for cryptographic applications," in *Field-Programmable Technology – FPT 2010*, 2010, pp. 308–311.