Reconfigurable Number Theoretic Transform Architectures for Cryptographic Applications

Gavin Xiaoxu Yao [#1], Ray C.C. Cheung [#2], Çetin Kaya Koç [*3], Kim Fung Man [#2]

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

[*] *Istanbul Şehir University & University of California, Santa Barbara*
[3] koc@cs.ucsb.edu

***Abstract*—As an important component of Spectral Modular Arithmetic (SMA) cryptographic co-processor, the efficient architectures of Number Theoretic Transforms (NTTs) on FPGA are discussed in this paper. We analyze characteristics of the NTTs for cryptographic applications, compare different arithmetic approaches, introduce an optimized solution for FPGA implementation, and developed several different architectures. Qualitative and quantitative analyses are provided to show the effectiveness of our proposed architectures.**

I. Introduction

Long integer modular multiplications are used intensively in many public-key cryptographic algorithms, such as RSA. The Montgomery modular multiplication algorithm [1] is the most widely deployed for its efficiency. However, the fully paralleled Montgomery method will quadruple the hardware size when the operand size doubles [2]. To avoid such rapid hardware growth, Spectral Modular Arithmetic (SMA) algorithms are proposed and claimed that the hardware size for such algorithm is linearly proportional to the operand size [3], [4], [5]. The SMA requires transforming the data to frequency domain by Number Theoretic Transform (NTT) at the preprocessing stage, and transforming back to time domain by Inverse NTT (INTT) in post processing.

The FFT architectures for NTT were proposed more than three decades ago [6], and extensively found for DSP applications in the literature [7], [8], [9]. However, the existing methods are not suitable of cryptographic applications. DSP applications employs real or complex arithmetic (as approximated by floating- and fixed-point arithmetic), while cryptographic applications require precise operations on integers or polynomials. Besides, the moduli in SMA algorithms is selected to be of the form $(2^n \pm 1)$, so that modular operations can be achieved in a fast fashion with little area. Hence, a tailored fast NTT architecture of SMA is desired.

On the other hand, scalability and reconfigurability are two desired characteristics in cryptographic applications. One of the reasons is that as the cryptanalysis gets more sophisticated, larger key lengths will have to be employed [10]. For instance, the security level of 1024-bit RSA is currently questioned, and 2048-bit RSA is recommended [11]. The RSA hardware module we use today may not meet the requirements in the future, and the design should be scalable and reconfigurable for the long-term concern. In addition, different design situations compile independent specifications. The mobile device may require little area and power efficiency, while a dedicated RSA coprocessor will require high throughput. Therefore, various architectures targeting different purposes should be developed for different design requirements.

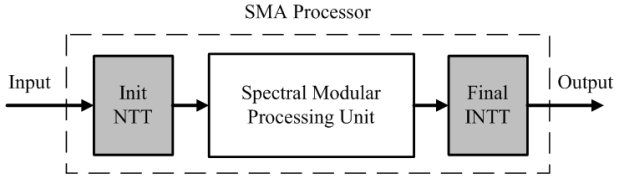

Fig. 1. The architecture of SMA processor and data flow

Therefore, reconfigurable architectures of fast NTT for SMA are proposed in this work. The main contributions of this paper are as follows:

- Arithmetic of modulo $2^n \pm 1$ operations is examined and optimized for FPGA implementation.
- Combinational, pipe-lined and area-compressed architectures for NTT are proposed in the scalable fashion using a design generator for different purposes and specifications.
- The complexity of the architectures is analyzed and FPGA implementations validate our analysis.

The rest of this paper is organized as follows. Section II introduces the background. Section III analyzes the arithmetic of basic operations. Section IV develops different architectures of NTT module. Section V provides the FPGA implementation results and analyses. Finally, Section VI concludes this paper.

II. Background

Multiplication of two integers can be calculated by the convolution of the polynomials if the integers are represented as the coefficient of the polynomials. The convolution is equivalent to the pair-wise multiplication in the transformed domain, which is less computational intensive if the transform process is not considered. In order to take the computational advantage of transformed domain, Spectral Modular Arithmetic (SMA) is developed. The signal flow of such SMA are shown in Fig. 1. The initial step is NTT, which transforms the polynomial to

the spectral representation, and the final step is INTT, which transforms the spectral representation back to the time domain. As our main concern in this work is about NTT and INTT the details about other part of SMAs are not introduced here.

The definition of NTT and INTT is provided here without further elaboration on the existence of inverse transform:

Definition 1: Let ω be a principal d-th root of unity in $\mathbb{Z}_q$ and $\{x_m\}$ be a length-d sequence of elements of $\mathbb{Z}_q$. Then the d-degree *number theoretic transform* $\{x_i\}$ to $\{X_i\}$ and its *inverse transform* is defined as:

$$X_i := \sum_{j=0}^{d-1} x_j \omega^{ij} \mod q, i = 0, 1, ..., d-1 \quad (1)$$

$$x_i := d^{-1} \sum_{j=0}^{d-1} X_j \omega^{-ij} \mod q, i = 0, 1, ..., d-1 \quad (2)$$

From the computational perspective, the rings q of the form $2^n \pm 1$ or $(2^n \pm 1)/p$, with principal root a power of 2, are most suitable because the arithmetic is simplified (see Sec. III). The rings of the form $(2^n \pm 1)$ are called the Fermat/Mersenne rings, and ones of the form $(2^n \pm 1)/p$ are called pseudo Fermat/Mersenne number. NTTs over these rings are also called the Mersenne Number Transform (MNT), the Fermat Number Transform (FNT), and Pseudo Number Transform (PNT) (specifically PFNT and PMNT) respectively.

FFT for NTT was proposed by Pollard [6]. Unlike the FFT in complex-number setting, it is always precise for FFT in a finite ring, group, or field, because the principal root of unity is infinite precise and there is no round-off errors. Compared to the matrix-vector product to compute NTT, the total number of operations of FFT is reduced to $O(d \cdot \log d)$ from $O(d^2)$. Furthermore, the multiplication can be achieved by simple shift as the principal root of unity is of the form 2^e. Hence, an even faster NTT can be achieved.

III. Basic Algorithms

Reduction. The mod $q = (2^n \pm 1)$ reduction of a number A can be simply computed by an addition or subtraction as (3) if A is at most $2n$-bit wide [12].

$$A \bmod (2^n \pm 1) = (A \bmod 2^n \mp \lfloor A/2^n \rfloor) \bmod (2^n \pm 1) \quad (3)$$

We take $(2^n - 1)$ as another representation of 0 for mod $(2^n - 1)$ operation, and the computation of mod $q = (2^n \pm 1)$ reduction can be expressed as Algorithm 1, where two n-bit adders are required. For FPGA, such algorithm utilizes $2n$ LUTs. If Line 1 yields underflow/overflow, it needs a correction as in Line 2. For PNT, mod q/p reduction is needed. A is first multiplied by p, then Algorithm 1 is performed, then the intermediate result is divided by p to get the final result.

Algorithm 1 Mod $q = (2^n \pm 1)$ reduction

1: $(c_{out}, temp) = A \bmod 2^n \mp \lfloor A/2^n \rfloor$
2: $A \bmod (2^n - 1) = temp + c_{out}$

Addition/Subtraction. The mod $(2^n - 1)$ addition and reduction is given as Algorithm 2, which utilizes 2 n-bit adders and therefore consumes $2n$ LUTs for FPGA implementation. Correction for over/underflow is performed in Line 2.

Algorithm 2 Mod $q = (2^n - 1)$ Addition/Subtraction

1: $(c_{out}, temp) = A \pm B$
2: $(A \pm B) \bmod (2^n - 1) = temp \pm c_{out}$

TABLE I
Area cost of modular addition and subtraction algorithms for FPGA implementation

Algorithm		Number of LUTs		
		Our work	Diminished-1	Prefix
FNT	Add	$2.2n$	$2n+1$	$(n \cdot \log_2 n)/4 + 2.5n$
	Sub	$2n+2$	$2.2n$	$(n \cdot \log_2 n)/4 + 2.5n$
	Pre.	-	n	-
	Post.	-	$1.2n$	-
MNT	Add	$2n$	-	$(n \cdot \log_2 n)/4 + 2.5n$
	Sub	$2n$	-	$(n \cdot \log_2 n)/4 + 2.5n$
	Post.	$1.2n$	-	n

The mod $(2^n + 1)$ addition/subtraction is performed by Algorithm 3 and Algorithm 4 respectively. Algorithm 3 employs 2 n-bit adders and a zero-comparator, which consumes $2.2n$ LUTs, while Algorithm 4 utilizes 2 $(n+1)$-bit adders, i.e. $2n+2$ LUTs. These two algorithms employ two carry bits to distinguish the special case $A \pm B = 2^n$ from over/underflow.

Algorithm 3 Mod $q = (2^n + 1)$ Addition

1: $(c_{out0}, c_{out1}, temp) = A + B$
2: $flag0 = \text{OR}_{i=0}^{n-1}\ temp_i$
3: $flag1 = \overline{flag0} \cdot (c_{out0} \oplus c_{out1})$
4: $flag2 = c_{out0} + flag0 \cdot \overline{c_{out0}} \cdot c_{out1}$
5: $(A + B) \bmod (2^n + 1) = (flag1, temp) - flag2$

Algorithm 4 Mod $q = (2^n + 1)$ Subtraction

1: $(c_{out0}, c_{out1}, temp1) = A - B$
2: $temp2 = temp1 + (c_{out0} \cdot c_{out1})$
3: $(A - B) \bmod (2^n + 1) = temp2 + \overline{c_{out0}} \cdot c_{out1} \cdot 2^n$

There are also other ways to implement modular addition $2^n \pm 1$ in literature [12]. The most representative methods are parallel prefix adder and diminished-1 number system. Such architectures are designed for ASIC but not suitable for FPGA implementation. The comparison of different algorithms of modular addition and subtraction for FPGA implementation is given by Table I.

Multiplication. As ω is a power of 2, the most suitable way to implement modular multiplication is to shift and reduce. For a constant multiplicand, the shift is just route and does not utilize logic device. For a multiplier that can multiply several values, a shifter is needed with additional $2n$ LUTs for reduction process. Furthermore, the simplicity of operation on ring $2^n - 1$ results that add/sub and multiplication by constant can be integrated, which result that the add/sub-multiply-reduce module only takes $2n$ LUTs.

Division. Division by constant is employed when performing PNT. An efficient division can be achieved if there exists certain β such that $\beta \cdot p = 2^\sigma - 1$, where σ is a nonnegative

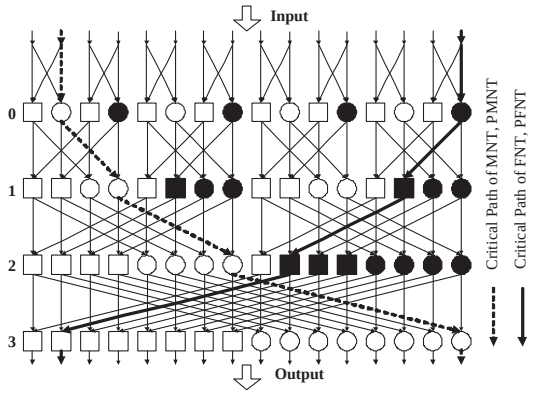

Fig. 2. Fixed architecture of a 16-degree NTT

integer. The formula is as Eq (4) [13] and performed by Algorithm 5, which involves$(\lceil \log_2(n/\sigma) \rceil + 1)$ $\lceil n + \sigma - \log_2 p \rceil$-bit adders, a constant multiplier, and a zero-comparator.

$$\frac{1}{p} = \frac{\beta}{2^\sigma - 1} = \frac{\beta}{2^\sigma(1 - 2^{-\sigma})}$$

$$= \frac{\beta}{2^\sigma}(1 + 2^{-\sigma})(1 + 2^{-2\sigma})(1 + 2^{-4\sigma})... \quad (4)$$

Algorithm 5 Compute B=A/p

1: $z \leftarrow A \cdot \beta$
2: **for** $i = 0 : \lceil \log_2(n/\sigma) \rceil - 1$ **do**
3: $\quad z \leftarrow z + 2^{-2^i \sigma} \cdot z$
4: **end for**
5: $z \leftarrow z/(2^\sigma)$
6: $B \leftarrow z + \text{OR}_{i=0}^{n-1}\ z_i$

IV. Proposed NTT Architecture Designs

Combinational Design. The position of each operation is fixed for a combinational design. The multiplication achieved by constant shift is just route and takes no logic device. The logic utilization mainly contributes to the modular addition, subtraction and reduction. Fig. 2 shows the architecture of a 16-degree FFT for NTT. Symbol □ or ■ represents modular addition, ○ or ● represents modular subtraction, and ■ or ● also denotes the modular reduction following the addition or subtraction. For MNT/PMNT, the addition is simpler than subtraction, hence, consumes less time, and the multiply-reduce module is integrated with the addition/subtraction, whose latency can be ignored. Therefore, the critical path goes through the most subtractors as the *dotted line* in Fig. 2. For FNT/PFNT, the add operation takes more time because zero-comparator and reducer are involved as well as two n-bit adders. Hence, the critical path is the *bold line*, which goes through reduction and addition operations. For PNT, the pre- and post-processing is also required and consumes more time.

The area complexity of combinational design is given by Table II according to Algorithm 2-4. The time complexity analysis is omitted due to paper length. Readers can perform the analysis according to the following truth. For the FPGA we use, the latency of an n-bit adder is $(c + \xi n)$, where c is a constant and ξ is the carry propagation rate. The latency of zero-comparator is $\lceil \log_6 n \rceil$ as 6-input LUTs is deployed. The time complexity of combinational design can be estimated by sum all the latency of the operators on the critical path.

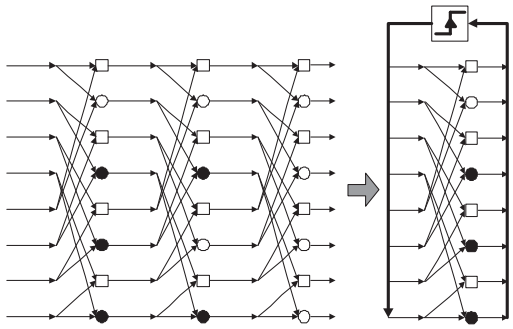

Fig. 3. Constant geometry FFT architecture and area-compressed Design

TABLE III
Qualitative comparison of different architectures

	Combinational	Pipelined	Area-compressed
Area	Light High	High	Low
Delay	Little	Light Medium	Medium
Throughput ratio	Medium Low	High High	Light Medium Medium

Pipelined Design. We can also utilize the flip-flops inside an FPGA slice by constructing a pipelined architecture. The area complexity of the pipelined design is almost the same as that of combinational design, as it mainly utilized the FFs in the slices whose LUTs are already occupied. The total latency would increase due to the clock signal needs time margin to establish for the robust design. However, the more stages, the higher throughput. Therefore, pipelined design is more efficient than combinational one with almost the same area.

Area-compressed Design. The area of the above designs yields the complexity of $O(nd \log_2 d)$. Such area consumption is unaffordable for limited area scenario. Therefore, the area-compressed design is proposed based on the constant geometry FFT which has the same signal path between layers as Fig. 3. We store the signal after the computation, and refresh the registers each clock cycle. Therefore, the ideal area compression ratio is $1/\log_2 d$. However, the shift bias is no longer constant for multiplication, and the shifter not only costs LUTs, but also makes the operation integration impossible. The shift bias varies during processing, hence, the counter and additional control unit are involved. The area is about two times that of a layer of the combinational design, and longer time is taken to complete the computation as the shifter costs addition time as well. The qualitative comparison of different architectures are provided by Table III.

V. FPGA Implementation

The FPGA we used is the Xilinx XC5VLX110T and the synthesis tool is Xilinx ISE 11.4 with the default constraints setting. The parameters in Table IV is used as examples to compare different arithmetic and architecture.

The FPGA implementation of combinational design for the examples shows the correctness of our complexity analyses as

TABLE II
AREA COMPLEXITY ANALYSIS OF COMBINATIONAL DESIGN

Operation unit			Modular Addition	Modular Subtraction	Modular Reduction	Pre-processing	Post-processing	Total Number
Number of units			$(d\log_2 d)/2$	$(d\log_2 d)/2$	$(d\log_2 d)/2$-d+1	d	d	-
#LUTs	MNT	Per unit	$2n$	$2n$	-	-	$1.2n$	$2nd(\log_2 d$+0.6)
		Sum	$nd\log_2 d$	$nd\log_2 d$	-	-	$1.2nd$	
	FNT	Per unit	$2.2n$	$2n+2$	$2n$	-	-	$(3.1n$+1$)d\log_2 d$-$2nd$+$2n$
		Sum	$1.1nd\log_2 d$	$(n$+1$)d\log_2 d$	$nd(\log_2 d$-2$)$+$2n$	-	-	
	PMNT	Per unit	$2n$	$2n$	-	$n\cdot(\bigtriangledown p-1)$ [a]	D+$1.2n$ [b]	$nd(2\log_2 d$+$\bigtriangledown p$+0.2$)$+Dd
		Sum	$nd\log_2 d$	$nd\log_2 d$	-	$nd\cdot(\bigtriangledown p-1)$	Dd+$1.2nd$	
	PFNT	Per unit	$2.2n$	$2n+2$	$2n$	$n\cdot(\bigtriangledown p-1)$	D+$0.2n$	$(3.1n$+1$)d\log_2 d$+
		Sum	$1.1nd\log_2 d$	$(n$+1$)d\log_2 d$	$nd(\log_2 d$-2$)$+$2n$	$nd\cdot(\bigtriangledown p-1)$	Dd+$0.2nd$	$(\bigtriangledown p$-2.8$)nd$+$2n$+Dd

[a] $\bigtriangledown A$ denotes the hamming weight of A [b] D=$(\bigtriangledown\beta$+$\lceil\log_2(n/\sigma)\rceil)\cdot(\lceil n$+$\sigma$-$\log_2 p\rceil)$

TABLE IV
THE SYMBOLS AND THE PARAMETERS OF THE EXAMPLES

Sym	Meaning	Parameters of the examples FNT	MNT	PNT
q	Ring size of $\mathbb{Z}_q$	2^{20}+1	2^{19}-1	2^{22}+1
n	The index number of q	20	19	22
ω	The principal root of unity	32	2	4

TABLE V
FPGA IMPLEMENTATION RESULT OF COMBINATIONAL DESIGN

		Implementation result	Theoretical value	Deviation
Number of LUTs	FNT	1258	1232	+2.1%
	MNT	2752	2796	-1.6%
	PNT	6099	6111	-0.2%
Critical path delay (ns)	FNT	25.278	25.3012	-0.09%
	MNT	20.726	20.6516	+0.36%
	PNT	41.454	41.4514	+0.26%

given by Table V. All the implementation results are very close to the theoretical value from Table II and the time complexity estimation. The comparison of different architectures for the MNT example shown as Table VI consists with the qualitative analysis in Table III. It shows the area-compressed design is the most area-friendly, the combinational design has the least latency, and the pipelined design has the highest throughput and the highest throughput/slice ratio, which means it is also the most economy design. We also implement pipelined architecture and area-compressed architecture, which all yield results very close to the value from our analytical model. The details are not included due to the space limit.

VI. CONCLUSIONS

Spectral Modular Arithmetic (SMA) algorithms maybe the solutions to the resource-intensiveness of public-key cryptography. As an important processing step of SMA, the family of various Number Theoretic Transforms (NTTs) are presented. The characteristics of NTT for cryptographic applications are analyzed, and the algorithms are optimized for field-programmable technology. In order to meet various specifications, different architectures are proposed and their complexity is examined. The analytical model is given for efficient timing and area estimate. FPGA implementations are described to analyze the designs qualitatively and quantitatively.

TABLE VI
FPGA IMPLEMENTATION RESULTS OF DIFFERENT ARCHITECTURES

	Combi-national	2-stage Pipelined	4-stage Pipelined	Area-compressed
Number of registers	0	608	1216	311
LUTs	2752	2752	2752	1494
slices	688	760	764	376
	0	320	912	299
utilization	0	10%	29%	19%
Minimum Period (ns)	-	7.531	4.088	8.115
Time offset (ns)	-	14.184	10.741	16.321
(ns)	20.726	21.751	23.005	40.666
(Mbps)	14667	40366	74364	9365
Throughput/slice	21.32	53.11	97.33	24.91

REFERENCES

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

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

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

[4] G. Saldamli and Ç. K. Koç, "Spectral modular exponentiation," in *Computer Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on*, 25-27 2007, pp. 123 –132.

[5] S. Baktir, "Frequency domain finite field arithmetic for elliptic curve cryptography," Ph.D. dissertation, Worcester Polytechnic Institute, 2008.

[6] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of Computation*, vol. 25, no. 114, pp. 365–374, 1971.

[7] R. Agarwal and C. Burrus, "Fast convolution using Fermat number transforms with applications to digital filtering," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 22, no. 2, pp. 87 – 97, apr 1974.

[8] S. Xu, L. Dai, and S. Lee, "Autocorrelation analysis of speech signals using Fermat number transform (FNT)," *Signal Processing, IEEE Transactions on*, vol. 40, no. 8, pp. 1910 –1914, aug 1992.

[9] T. Toivonen and J. Heikkila, "Video filtering with Fermat number theoretic transforms using residue number system," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 1, pp. 92 – 101, jan. 2006.

[10] D. Giry. (2010) Cryptographic key length recommendation. [Online]. Available: http://www.keylength.com/

[11] R. Silverman. (2002, Apr.) Has the RSA algorithm been compromised as a result of Bernstein's paper? [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2007

[12] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on*, 1999, pp. 158 –167.

[13] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, USA: Oxford University Press, 2010.