Low-Latency Preprocessing Architecture for Residue Number System via Flexible Barrett Reduction for Homomorphic Encryption

Sin-Wei Chiu, *Graduate Student Member, IEEE*; and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—Data privacy has become a significant concern due to the rapid development of cloud services, Internet of Things, edge devices, and other applications. Homomorphic encryption (HE) addresses the issue by enabling computations to be performed without the decryption of the encrypted message. However, the bottleneck of designing homomorphic encryption hardware is the complexity of computation. To tackle the long integer arithmetic, the residue number system based on the Chinese remainder theorem is used. In this paper, we propose a novel modular reduction architecture that computes the mapping of residual polynomials in parallel with high speed and low latency. We implement our proposed design in the Xilinx Ultrascale+ FPGA board (VCU118). When the input sizes are 360-bit (1440-bit), the frequency is 180MHz (168MHz) with 4 pipelining stages. Also, the area delay product (ADP) of DSP blocks of our design is reduced by 23 and 31 percent, respectively, for 360 and 1440 bits, compared to prior work.**

***Index Terms*—Homomorphic encryption, Residue number system, Chinese remainder theorem, Modular reduction, Barrett reduction, Hardware accelerator**

I. Introduction

Homomorphic encryption (HE) allows operations on plaintexts in the encrypted domain and preserves the privacy of user data [1]. Polynomial modular multiplication is an important building block in all HE schemes. The degree of polynomials, n, and the dynamic range of the coefficients, also referred as the ciphertext modulus, q, in FHE schemes are very large. For example, to achieve at least 80-bit security, we require n and q to be 4096 and 180 bits, respectively [2]. To efficiently compute the long word-length operations, the residue number system (RNS) representation is used [2]–[7]. RNS decomposes a long word-length number into multiple shorter word-length numbers. Although parallelism can reduce the latency of the operations, it also increases the cost of the design. Designing an architecture that can exploit parallelism without significantly increasing the cost is non-trivial.

Traditionally, the computation of modular reduction can be carried out in several ways. It can be computed using integer division (finding the quotient), Montgomery reduction [8], Will and Ko [9], or Barrett reduction [10]. However, the cost of computing the residue using these methods drastically increases when the input word-length increases due to the long integer multiplication or division operations. The approaches in this paper reduce the area cost of the parallel low-latency residual polynomials computation unit for a very long input word-length (i.e., $\lceil \log_2 q \rceil \geq 360$), as compared to prior works [11], [12]. The contributions of the paper are two-fold. First, a flexible special prime selection method is proposed that can significantly increase the number of feasible moduli for HE. Second, we propose a novel architecture to achieve high-speed parallel computation with low latency and lower cost compared to prior designs. The key to the proposed architecture is the proposed novel *flexible* Barrett reduction where the input word-length is *not* constrained to be less than or equal to two times the word-length of the ciphertext modulus q.

This paper is organized as follows. Section II provides the background including RNS representation, polynomial ring, and Barrett reduction. Section III introduces the implementations of the prior works. Section IV presents our novel parallel architecture of residual polynomial computation unit. Section V presents the experimental results of our proposed design.

II. Background

A. RNS Representation

The RNS Representation is based on the Chinese remainder theorem (CRT). The CRT algorithm can be stated as follows: If $q_1, q_2, \ldots, q_t$ are pairwise coprime, where $q = \prod_{i=1}^{t} q_i$, the map $a \mod q \mapsto (a \mod q_1, a \mod q_2, \ldots, a \mod q_t)$ defines a ring isomorphism $\mathbb{Z}/q\mathbb{Z} \cong \mathbb{Z}/q_1\mathbb{Z} \times \mathbb{Z}/q_2\mathbb{Z} \times \cdots \times \mathbb{Z}/q_t\mathbb{Z}$, that is between the ring of integer modulo q and the direct product of the ring of integer modulo q_i. In other words, we can perform a sequence of arithmetic operations independently and in parallel in each $\mathbb{Z}/q_t\mathbb{Z}$ and then map them back in $\mathbb{Z}/q\mathbb{Z}$. This is more efficient than performing a sequence of arithmetic operations in $\mathbb{Z}/q\mathbb{Z}$, because the word-length in $\mathbb{Z}/q_t\mathbb{Z}$ is significantly smaller than $\mathbb{Z}/q\mathbb{Z}$.

B. Polynomial Ring

Consider the operations over a polynomial ring $R_{n,q} = \mathbb{Z}_q[x]/(x^n + 1)$, There are two major constraints. First, the coefficients of the polynomial ring have to be modulo q (i.e., the coefficients are integers and lie in the range $[0, q-1]$). Second, the degree of the polynomial is less than n, where n is a power-of-two integer. It is important to note that n, q should satisfy $q \mod 2n \equiv 1$. We can represent a polynomial over the ring $R_{n,q}$ by the equation:

This research was supported in parts by the Semiconductor Research Corporation under contract number 2020-HW-2998, and by the National Science Foundation under grant number CCF-2243053.

Sin-Wei Chiu and Keshab K. Parhi are with Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA, E-mail: {chiu0091, parhi}@umn.edu

$$a_q(x) = \sum_{j=0}^{n-1} a_{j,q} x^j \tag{1}$$

where $a_{j,q}$ is an integer that is in the range $[0, q-1]$.

C. Barrett Reduction

Barrett reduction is a reduction algorithm introduced by Barrett. It computes the modular reduction of a by q:

$$r = a \mod q \tag{2}$$

where r is the remainder. Barrett reduction is designed to optimize modular reduction by replacing the divisions with multiplications. We can rewrite Equation (2) as:

$$\begin{aligned} a \mod q &= a - (\frac{am}{2^k}) \cdot q \\ &= a - ((am) \gg k) \cdot q \end{aligned} \tag{3}$$

where $m = \lfloor \frac{2^k}{q} \rfloor$. Notice that the division in Equation (3) is replaced by a multiplication and a right shift. Since $\frac{m}{2^k} \leq \frac{1}{q}$, the value of the remainder r can end up being too small, and thus r is only guaranteed to be within $[0, 2q)$ instead of $[0, q)$ as is generally required. A conditional subtraction can be used to correct this.

To ensure r is within $[0, 2q)$, a has to be smaller than 2^k. The proof is presented in the Supplementary Information.

The traditional block diagram of Barrett reduction is shown in Figure 1. The partially reduced value, p_r, is the value before the conditional subtraction.

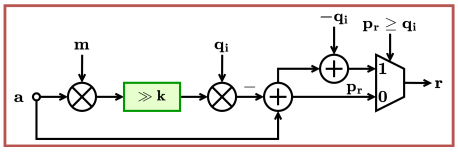

Fig. 1. Block diagram for a traditional Barrett reduction.

D. Flexible Barrett Reduction

Conventionally, when Barrett reduction is used in modular multiplier, k is set to be $2 \cdot \lceil \log_2 q_i \rceil$ to satisfy the constraint. However, in our proposed design, k is set to be the same as the word-length of the input dynamic range q_a, i.e., input a lies in the range $[0, q_a)$. This means k can be smaller or larger than $2 \cdot \lceil \log_2 q_i \rceil$ to increase flexibility. Figure 2 shows the block diagram of the proposed flexible Barrett reduction unit. When k is set to be the same as the word-length of the input dynamic range q_a, the hardware cost of the block is fully dependent on and proportional to the input dynamic range. It is easy to see that when q_a increases, the cost of the entire block also increases.

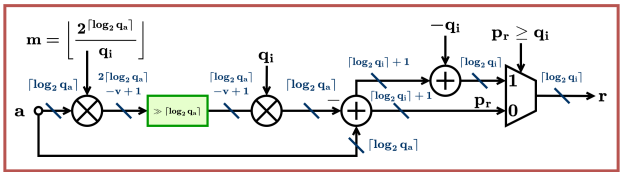

Fig. 2. Block diagram for a flexible Barrett reduction.

III. Prior Works on Residual Polynomial Computation

The goal of the residual polynomial computation unit is to compute the residual polynomial $a_{q_i}(x)$ from the input polynomial $a_q(x)$ by the equation:

$$a_{j,q_i} = a_{j,q} \mod q_i \text{ for } j = 0, 1, \ldots, n-1 \tag{4}$$

In the literature, prior works have addressed dedicated modular reduction implementations using Barrett reduction [11], [12]. In [11], a 1228-bit input is reduced using a 30-bit coprime with two stages of partial reduction and one Barrett reduction block. The design in [11] utilizes only one computational unit to compute the modular reduction with many cycles. However, to achieve low latency, the design can be implemented in a parallel fashion. The parallel implementation duplicates the computation units in the first step of the algorithm. The block diagram of the architecture is presented in the Supplementary Information. This design decomposes a long word-length input a into t segments of word-length v. It requires $t-1$ multipliers and one Barrett reduction block when implemented in parallel. This will not be ideal when the input word-length is long.

The modular reduction implementation proposed in [12], [13] performs the operation in parallel, and the design takes advantage of the special coprime selection scheme, which enables d blocks of partial reduction. The block diagram of the architecture is presented in the Supplementary Information. Although the design implements special coprime selection to replace large integer multipliers, the word-length increase introduced by the shift-and-add units will need Barrett reduction blocks to bring the intermediate results back to shorter word-length to contain the hardware cost. The design requires $d-1$ multipliers and d Barrett reduction blocks, where $d = t/l$, and l is an integer. The main drawback of this design is that the large number of Barrett reduction blocks require significant increase in area.

IV. Proposed Architecture of the Residual Polynomial Computation Unit

In this section, we first describe the high-level architecture of our proposed design. Next, we introduce our proposed flexible special coprime selection. This selection scheme enables many hardware-friendly properties that are exploited in our proposed design. Lastly, we describe the design of the Barrett reduction block.

A. High-level Architecture

Figure 3 shows the high-level architecture of the residual polynomial computation unit. The proposed design is based on Algorithm 1. In line 1, the input coefficient is split into d v-bit words, where $v = \lceil \log_2 q_i \rceil$. Each word is stored in the array $\mathbf{A}$ with the least significant word stored in the index position 0. Next in line 4, we first group the t words into $d = \frac{t}{2}$ sets, where each set consists of two consecutive words. We first combine each set into partially reduced words by multiplying the most significant word in the set by β_i and then summing the two words. Note that based on our flexible special coprime selection scheme, multiplication by β_i can be

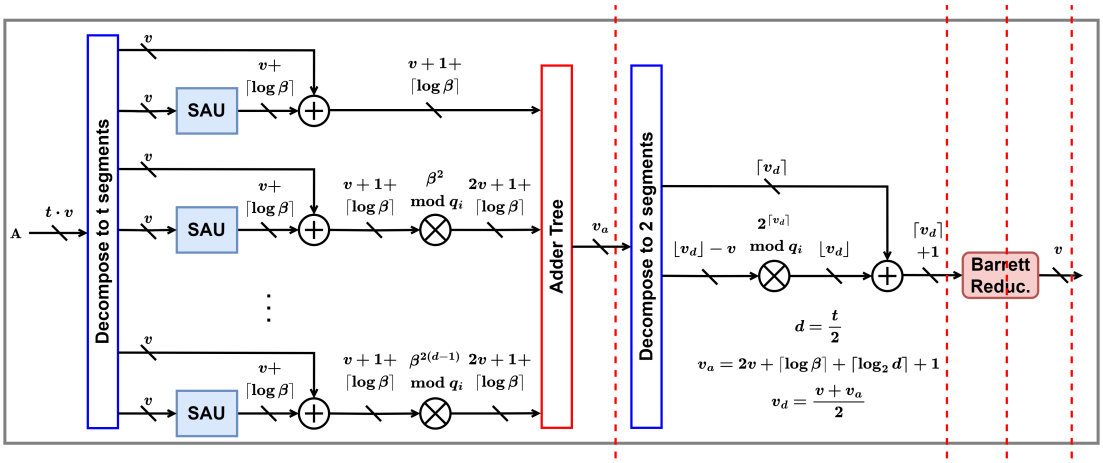

Fig. 3. High-level Architecture of the residual polynomial computation unit.

replaced with shift-and-add operations. Each partially reduced word will then multiply with $\beta_i^{2j} \mod q_i$ to create a partially reduced value. The word-length of the partially reduced value is $v_a = (2 \cdot v) + \lceil \log_2 \beta_i \rceil + \lceil \log_2 d \rceil + 1$. $\lceil \log_2 d \rceil$ is the height of the adder tree. Next in line 7, we decompose the partially reduced result into two segments. The first segment has word-length $\lceil v_d \rceil$ and the second segment has word-length $\lfloor v_d \rfloor - v$, where v_d is defined as $\frac{v_a+v}{2}$. The definition of v_d ensures minimum output word-length after line 7. The most significant segment is multiplied by $2^{\lceil v_d \rceil} \mod q_i$ and the result is added with the least significant segment to get a partially reduced value in R. The size of R will be $\lceil v_d \rceil + 1$, which is the minimum size possible. This value is further reduced by a Barrett reduction block to get the final v-bit result $\mod q_i$.

Given any q and q_i, where q_i can be represented as a few signed power-of-two terms, our proposed design can reduce any coefficient $a \mod q$ by q_i with d multipliers and one Barrett reduction block.

Algorithm 1 Reduction of $\lceil \log_2 q \rceil$-bit coefficient by q_i

Input: $a \mod q$, $\lceil \log_2 q \rceil = (t \times v)$, $t \geq 2$ and $v = \lceil \log_2 q_i \rceil$
Output: $R = a \mod q_i$

1: $\mathbf{A} \leftarrow [a_0, a_1, \ldots, a_{t-1}]$
2: $d \leftarrow \lceil \frac{t}{2} \rceil$
3: $\beta_i \leftarrow 2^v \mod q_i$
4: $R \leftarrow \sum_{j=0}^{d-1} (\mathbf{A}[2j] + \mathbf{A}[2j+1] \cdot \beta_i) \cdot \left(\beta_i^{2j} \mod q_i\right)$
5: $v_a \leftarrow (2 \cdot v) + \lceil \log_2 \beta_i \rceil + \lceil \log_2 d \rceil + 1$
6: $v_d \leftarrow \frac{v_a+v}{2}$
7: $R \leftarrow R\left[\lceil v_d \rceil - 1 : 0\right] + R\left[v_r - 1 : \lceil v_d \rceil\right] \cdot \left(2^{\lceil v_d \rceil} \mod q_i\right)$
8: $R \leftarrow BarrettReduction(R, q_i)$
9: **return** R

B. Flexible Special Coprime Selection

Instead of randomly selecting coprimes for the RNS representation, we can carefully select coprimes with special properties and exploit them.

In the proposed architecture, there are three constraints that each coprime has to satisfy. First, each coprime has to be an NTT-compatible prime, that is, $q_i - 1$ has to be divisible by $2n$, where n is the degree of the polynomials. Second, each coprime can only have a few signed power-of-two terms [14], which is defined as:

$$q_i = 2^v - \beta_i, \quad \beta_i = 2^{v_{1,i}} \pm 2^{v_{2,i}} \pm \ldots \pm 2^{v_{(p-2),i}} - 1,$$

where p is the number of power-of-two terms of a coprime, v is the word-length of q_i, and $v_{1,i} > v_{2,i} > \ldots > v_{p-2,i}$. Selecting coprimes with a few signed power-of-two terms reduces the complexity of multiplication by q_i as it can be replaced by a few shift-and-add operations [15].

TABLE I
THE NUMBER OF COPRIMES WITH DIFFERENT PARAMETER SETS

$(n, \lceil \log_2 q \rceil)$	$\lceil \log_2 q_i \rceil$	c	$\max(\lceil \log_2 \beta_i \rceil)$	p	# coprimes
$(2^{13}, 360)$	30	60	24	4	9
$(2^{13}, 360)$	30	62	28	4	24
$(2^{13}, 360)$	30	60	24	5	35
$(2^{13}, 360)$	30	62	28	5	120
$(2^{14}, 720)$	30	63	29	4	20
$(2^{14}, 720)$	30	63	29	5	121
$(2^{15}, 1440)$	30	63	28	4	14
$(2^{15}, 1440)$	30	63	28	5	70

TABLE II
TIMING RESULTS

Design	$(n, \lceil \log_2 q \rceil)$	$\lceil \log_2 q_i \rceil$	Freq. (MHz)	delay (ns)	Pipe
Proposed	$(2^{13}, 360)$	30	56	17.97	0
Roy [11]	$(2^{13}, 360)$	30	53	18.82	0
Tan [12]	$(2^{13}, 360)$	30	46	21.78	0
Proposed	$(2^{13}, 360)$	30	180	5.57	4
Roy [11]	$(2^{13}, 360)$	30	176	5.69	4
Tan [12]	$(2^{13}, 360)$	30	135	7.4	4
Proposed	$(2^{15}, 1440)$	30	50	19.85	0
Roy [11]	$(2^{15}, 1440)$	30	46	21.72	0
Tan [12]	$(2^{15}, 1440)$	30	39	25.68	0
Proposed	$(2^{15}, 1440)$	30	168	5.95	4
Roy [11]	$(2^{15}, 1440)$	30	152	6.59	4
Tan [12]	$(2^{15}, 1440)$	30	128	7.81	4

TABLE III
PERFORMANCE AND RESOURCE UTILIZATION RESULTS

Design	$(n, \lceil \log_2 q \rceil)$	$\lceil \log_2 q_i \rceil$	Frequency (MHz)	LUT/Util./ADP	DSP/Util./ADP	FF/Util./ADP
Proposed	$(2^{13}, 360)$	30	180	1.6k/0.13%/8.85	38/0.56%/0.21	1k/0.04%/5.83
Roy [11]	$(2^{13}, 360)$	30	176	1.4k/0.12%/7.90	48/0.7%/0.27	1.3k/0.05%/7.36
Tan [12]	$(2^{13}, 360)$	30	135	2.4k/0.2%/17.41	44/0.64%/0.33	1.1k/0.05%/8.12
Proposed	$(2^{15}, 1440)$	30	168	5.6k/0.48%/33.42	142/2.08%/0.84	3.6k/0.15%/21.54
Roy [11]	$(2^{15}, 1440)$	30	152	4k/0.33%/26.04	186/2.72%/1.23	4.5k/0.19%/29.89
Tan [12]	$(2^{15}, 1440)$	30	128	9.7k/0.82%/76.01	187/2.73%/1.46	4k/0.17%/31.36

Third, we want to minimize β_i. This is because $\beta_i = 2^v \bmod q_i$, we can replace multiplication by β_i with shift-and-add operations. However, it will introduce an increase in word-length by $\lceil \log_2 \beta_i \rceil$ bits. This becomes a major drawback in the architecture presented in [12]. Therefore, to limit the word-length increase for each datapath, we need to minimize the size of β_i. The constraint can be set by limiting the input word-length of the Barrett reduction block (line 8 in Algorithm 1). The main reason for applying the constraint to the input word-length of the Barrett reduction block is that it is the most hardware-expensive block in the system. Limiting the input word-length directly translates to significant saving in cost. From Figure 3, we set the maximum word-length to be $c \geq \lceil v_d \rceil + 1$. We then have the inequality:

$$\left\lceil \frac{(3 \cdot v) + \lceil \log_2 \beta_i \rceil + \lceil \log_2 d \rceil + 1}{2} \right\rceil + 1 \leq c$$

Solving this inequality, we have:

$$\lceil \log_2 \beta_i \rceil \leq (2 \cdot c) - (3 \cdot v) - \lceil \log_2 d \rceil - 3 \qquad (5)$$

Equation (5) becomes the constraint on β_i. Therefore, given n, c, p and the size of q and q_i, we can find the number of coprimes available. Based on the three constraints above, we can select the appropriate moduli set using an exhaustive search to find the coprime pool.

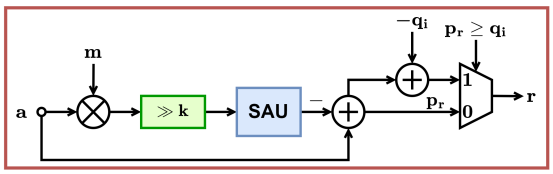

Fig. 4. Proposed Barrett reduction block diagram.

C. Barrett Reduction

The design of the Barrett reduction block follows the architecture presented in Section II-D. Also, it utilizes the hardware-friendly property of our selected coprime [16]. From Equation 3, it is easy to see that q_i only has a few signed power-of-two terms, and multiplication by q_i can be replaced by a simple shift-and-add unit (SAU). This can reduce the number of integer multipliers. Hence, the cost of implementing the Barrett reduction blocks is reduced. Figure 4 shows the block diagram of the proposed Barrett reduction block. Inside this block, we have one integer multiplication. It is used to multiply a precomputed constant m that is determined by q_i and k, where k is equal to $\lceil \log_2 q_a \rceil$ in our flexible Barrett design. After the integer multiplication, the result is right-shifted by $\lceil \log_2 q_a \rceil$-bits and then multiplied by q_i using shift-and-add operations. The input will then be subtracted by this value and a conditional subtraction by q_i will ensure that the output r will be in the range $[0, q_i)$.

It is important to point out that based on the proof in the supplementary information, no matter the size of the input, as long as it is smaller than $2^{\lceil \log_2 q_a \rceil}$, the output r is guaranteed to be smaller than q_i. However, increasing the size of the input will increase $\lceil \log_2 q_a \rceil$, which will increase the size of m. Hence, it will also increase the cost of the integer multiplier inside the Barrett reduction block.

V. EXPERIMENTAL RESULTS

A. Flexible Coprime Selection Reults

Based on the flexible coprime selection scheme discussed in Section IV-B, we can obtain the coprime selection pool via an exhaustive search. Table I shows the number of coprimes available with different parameter sets. It is generated by a Python program based on the previously discussed constraints. When $(n, \lceil \log_2 q \rceil) = (2^{13}, 360)$, and the number of power-of-two terms is 4, $c = 60$ only gives 9 available coprimes. Since this number is smaller than what is required $(360/30 = 12)$, we can expand the selection pool by increasing c or p. Increasing c by 2 gives us 24 available coprimes, and increasing p by 1 gives us 35 available coprimes. To find the optimal parameter set, start with a strict parameter set and loosen them one by one until the required number of coprimes are available. The same procedure can be applied when $(n, \lceil \log_2 q \rceil)$ are $(2^{14}, 720)$ and $(2^{15}, 1440)$. The results are also shown in Table I.

B. Timing Results

We describe our design in SystemVerilog HDL and implemented our proposed design using the Xilinx Ultrascale+ FPGA board (VCU118). We consider two different parameter sets. The first parameter set is $(n, \lceil \log_2 q \rceil) = (2^{13}, 360)$ and $q_i = 1073692673$, and the second parameter set is $(n, \lceil \log_2 q \rceil) = (2^{15}, 1440)$ and $q_i = 1073479681$. We implement our design with and without additional pipelining [15]. The cycle count of our proposed design is equivalent to the number of pipelining stages. The placement of the pipelining cutsets is shown in red dashed lines. For the first parameter set, without pipelining, the frequency is 56MHz. After inserting 4 pipelining cutsets, the frequency increases to 180MHz. For the second parameter set, without pipelining, the frequency is 50MHz. After inserting 4 pipelining cutsets, the frequency increases to 168MHz. The full timing results are shown in Table II. Also, the pipeline efficiency is summarized in Table 1 of the Supplementary Information.

C. Performance and Resource Utilization Results

Table III shows the performance and resource utilization results per coefficient when implemented on Xilinx Ultrascale+ FPGA board (VCU118). The results shown in the table all have 4 pipelining cutsets. The area delay product (ADP) of each component is calculated by the product of LUTs/DSPs/FFs and delay (μs).

D. Comparison with Related Works

We first compare our work with [11]. In order to provide a better comparison, we implement the design in a parallel form, i.e., we duplicate the computation units in the first step of the algorithm. The design requires $t-1$ multipliers when implemented in parallel. Our proposed design replaces half of the multipliers in the first stage by shift-and-add units. Additionally, our design does not require more Barrett reduction blocks. In Table III, when $\lceil \log_2 q \rceil$ is 360, although the ADP of LUT in our design is increased by 12%, the ADP of DSP is reduced by 23% per coefficient. When $\lceil \log_2 q \rceil$ is 1440, the ADP of LUT in our design is increased by 28%, and the ADP of DSP is reduced by 31% per coefficient. Our design has higher LUT utilization because the shift-and-add units in our design introduce a word-length increase. Nevertheless, the shift-and-add units reduce the number of multipliers, which are usually implemented using DSP. Our proposed design aims to minimize the usage of DSP due to the fact that DSP has a higher cost and is relatively scarce in FPGA boards compared to LUT. Our design also has a higher frequency; this is because the height of the adder tree is shorter. Hence, the critical path is reduced.

Next, we compare our design with [12]. This design implements d Barrett reduction blocks to deal with the word-length increase introduced by the shift-and-add units. Our proposed design addresses this issue by implementing fewer shift-and-add units on each data path. As a result, our design only requires one Barrett reduction block. Since Our proposed design incorporates both a special coprime selection scheme and low Barrett reduction block utilization, it has significant improvements in terms of timing and resource utilization. When $\lceil \log_2 q \rceil$ is 360, the ADP of LUT/DSP is reduced by $49\%/35\%$ per coefficient, and when $\lceil \log_2 q \rceil$ is 1440, the ADP of LUT/DSP is reduced by $56\%/42\%$ per coefficient. The design also has a lower frequency compared to ours; This is due to the fact that the input word-length of the final Barrett reduction is long.

VI. Conclusion

This paper presents a novel design for the preprocessing architecture for the residue number system. Our work utilizes a special coprime selection scheme and a highly parallel architecture to achieve low latency and cost. It is important to point out that the special coprime selection scheme proposed in this paper can be further extended to the entire homomorphic encryption scheme as coprimes with special properties can reduce the cost of a hardware accelerator for homomorphic encryption [12], [17]–[20]. Future research should be directed toward exploring the impact of different modular reduction methods on the performance of different homomorphic encryption schemes.

References

[1] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[2] F. Turan, S. S. Roy, and I. Verbauwhede, "HEAWS: An accelerator for homomorphic encryption on the amazon aws fpga," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1185–1196, 2020.

[3] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "Fpga-based accelerators of fully pipelined modular multipliers for homomorphic encryption," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2019, pp. 1–8.

[4] J. Cathébras, A. Carbon, P. Milder, R. Sirdey, and N. Ventroux, "Data flow oriented hardware design of rns-based polynomial multiplication for she acceleration," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 69–88, 2018.

[5] Y. Su, B.-L. Yang, C. Yang, and S.-Y. Zhao, "Remca: A reconfigurable multi-core architecture for full rns variant of bfv homomorphic evaluation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 7, pp. 2857–2870, 2022.

[6] S. Halevi, Y. Polyakov, and V. Shoup, "An improved rns variant of the bfv homomorphic encryption scheme," in *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer, 2019, pp. 83–105.

[7] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

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

[9] M. A. Will and R. K. Ko, "Computing mod without mod," *Cryptology ePrint Archive*, 2014.

[10] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[11] S. S. Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1637–1650, 2018.

[12] W. Tan, S.-W. Chiu, A. Wang, Y. Lao, and K. K. Parhi, "PaReNTT: Low-latency parallel residue number system and NTT-based long polynomial modular multiplication for homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 19, 2024, doi: 10.1109/TIFS.2023.3338553.

[13] K. K. Parhi, W. Tan, S.-W. Chiu, A. Wang, and Y. Lao, "Parallel polynomial modular multiplication using NTT and inverse NTT," U.S. Patent Application 18/500,670, Nov. 2, 2023.

[14] W. Tan, B. M. Case, A. Wang, S. Gao, and Y. Lao, "High-speed modular multiplier for lattice-based cryptosystems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2927–2931, 2021.

[15] K. K. Parhi, *VLSI digital signal processing systems: design and implementation*. John Wiley & Sons, 2007.

[16] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi, "Pipelined high-throughput NTT architecture for lattice-based cryptography," in *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2021, pp. 1–4.

[17] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2019.

[18] A. C. Mert, S. Kwon, Y. Shin, D. Yoo, Y. Lee, S. S. Roy *et al.*, "Medha: Microcoded hardware accelerator for computing on encrypted data," *arXiv preprint arXiv:2210.05476*, 2022.

[19] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252.

[20] R. Geelen, M. Van Beirendonck, H. V. Pereira, B. Huffman, T. McAuley, B. Selfridge, D. Wagner, G. Dimou, I. Verbauwhede, F. Vercauteren *et al.*, "Basalisc: Programmable asynchronous hardware accelerator for BGV fully homomorphic encryption," *Cryptology ePrint Archive*, 2022.

Supplementary Information: Low-Latency Preprocessing Architecture for Residue Number System via Flexible Barrett Reduction for Homomorphic Encryption

Sin-Wei Chiu, *Graduate Student Member, IEEE*; and Keshab K. Parhi, *Fellow, IEEE*

I. Barrett Reduction

To ensure the output of the Barrett reduction, r, is within $[0, 2q)$, the input a has to be smaller than 2^k [1]. If $a \leq 2^k$, then $q > \frac{a}{2^k} \cdot (2^k \mod q)$. We have:

$$q > \frac{a \cdot (2^k \mod q)}{2^k}$$
$$q > a - \left(a - \frac{a \cdot (2^k \mod q)}{2^k}\right)$$
$$q > a - \frac{aq}{2^k} \cdot \left(\frac{2^k - (2^k \mod q)}{q}\right)$$
$$q > a - \frac{aq}{2^k} \cdot \lfloor \frac{2^k}{q} \rfloor$$

since $q \cdot \frac{am \mod 2^k}{2^k} < q$ regardless of a, and $m = \lfloor \frac{2^k}{q} \rfloor$:

$$2q > a - \frac{aqm}{2^k} + q \cdot \frac{am \mod 2^k}{2^k}$$
$$2q > a - \left(\frac{am - (am \mod 2^k)}{2^k}\right) \cdot q$$
$$2q > a - \lfloor \frac{am}{2^k} \rfloor \cdot q$$
$$2q > r$$

we can rewrite the modular reductions as:

$$a \mod q = \begin{cases} r & \text{if } r < q \\ r - q & \text{otherwise} \end{cases}$$

In conclusion, for any given k, if $a < 2^k$, only one conditional subtraction is required at the end of Barrett reduction.

II. Prior Works

In this section of the supplementary information, we give the block diagrams presented by the prior works discussed in the paper. Figure 1 represents the block diagram of the parallel implementation architecture presented in [2].

Figure 2 represents the block diagram of the architecture presented in [3].

This research was supported in part by the Semiconductor Research Corporation under contract number 2020-HW-2998.

Sin-Wei Chiu and Keshab K. Parhi are with Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA, E-mail: {chiu0091, wtan, parhi}@umn.edu

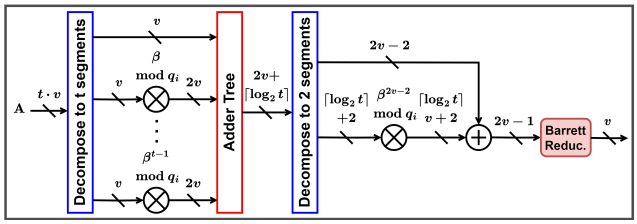

Fig. 1. Block diagram of the parallel implementation architecture presented in [2].

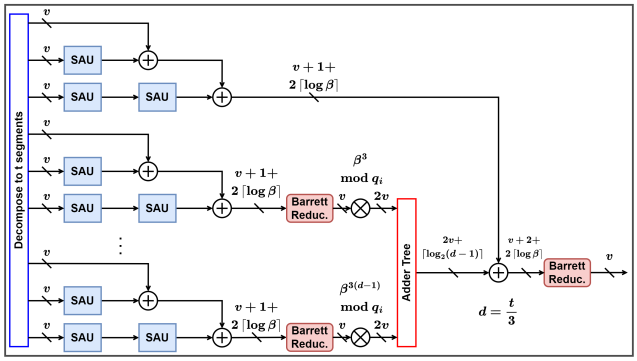

Fig. 2. Block diagram of the architecture presented in [3].

III. Pipeline Efficiency

Table I shows the pipeline efficiency of the results presented in this paper. The pipeline efficiency is calculated by the delay of the design before pipelining (delay b.p.) divided by the total delay of the design after pipelining (delay a.p.), which is equivalent to delay after pipelining times the pipelining stages.

TABLE I
Pipeline Efficiency

Design	$(n, \lceil \log_2 q \rceil)$	$\lceil \log_2 q_i \rceil$	Pipe	delay b.p. (ns)	delay a.p. (ns)	eff. (%)
Proposed	$(2^{13}, 360)$	30	4	17.97	22.78	81
Roy [2]	$(2^{13}, 360)$	30	4	18.82	22.76	83
Tan [3]	$(2^{13}, 360)$	30	4	21.78	29.6	74
Proposed	$(2^{15}, 1440)$	30	4	19.85	23.8	83
Roy [2]	$(2^{15}, 1440)$	30	4	21.72	26.36	82
Tan [3]	$(2^{15}, 1440)$	30	4	25.68	31.24	82

REFERENCES

[1] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[2] S. S. Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1637–1650, 2018.

[3] W. Tan, S.-W. Chiu, A. Wang, Y. Lao, and K. K. Parhi, "PaReNTT: Low-latency parallel residue number system and NTT-based long polynomial modular multiplication for homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 19, 2024, doi: 10.1109/TIFS.2023.3338553.