electronics

MDPI

Article

High Precision Multiplier for RNS $\{2^n - 1, 2^n, 2^n + 1\}$

Shang Ma *,†, Shuai Hu †, Zeguo Yang, Xuesi Wang, Meiqing Liu and Jianhao Hu

National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China, 510, Main Building, No. 2006, Xiyuan Ave, West Hi-Tech District, Chengdu 611731, China; 201911220502@std.uestc.edu.cn (S.H.); yzg931228@163.com (Z.Y.); 201821220423@std.uestc.edu.cn (X.W.); 201821220360@std.uestc.edu.cn (M.L.); jhhu@uestc.edu.cn (J.H.)

* Correspondence: mashang@uestc.edu.cn

† These authors contributed equally to this work.

Abstract: The Residue Number System (RNS) is a non-weighted number system. Benefiting from its inherent parallelism, RNS has been widely studied and used in Digital Signal Processing (DSP) systems and cryptography. However, since the dynamic range in RNS has been fixed by its moduli set, it is hard to solve the overflow problem, which can be easily solved in Two's Complement System (TCS) by expanding the bit-width of it. For the multiplication in RNS, the traditional way to deal with overflow is to scale down the inputs so that the result can fall in its dynamic range. However, it leads to a loss of precision. In this paper, we propose a high-precision RNS multiplier for three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, which is the most used moduli set. The proposed multiplier effectively improves the calculation precision by adding several compensatory items to the result. The compensatory items can be obtained directly from preceding scalers with little extra effort. To the best of our knowledge, we are the first one to propose a high-precision RNS multiplier for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$. Simulation results show that the proposed RNS multiplier can get almost the same calculation precision as the TCS multiplier with respect to Mean Square Error (MSE) and Signal-to-Noise Ratio(SNR), which outperforms the basic scaling RNS multiplier about 2.6–3 times with respect to SNR.

Keywords: residue number system; scaler; multiplier; Chinese remainder theorem; very large scale integration circuits

Citation: Ma, S.; Hu, S.; Yang, Z.; Liu, M.; Hu, J. High Precision Multiplier for RNS $\{2^n - 1, 2^n, 2^n + 1\}$. *Electronics* **2021**, *10*, 1113. https://doi.org/10.3390/electronics10091113

Academic Editor: Fabian Khateb

Received: 4 March 2021
Accepted: 3 May 2021
Published: 8 May 2021

1. Introduction

The Residue Number System (RNS) is a non-weighted parallel numerical representation system, which divides the integers into multiple independent ones through modular operations. Thus, the bit-width of each channel is greatly reduced. As a result, RNS-based systems have the potential to achieve high calculation speed and low complexity. RNS is very suitable to process large integer numbers, which makes it extremely useful in cryptography [1], such as Elliptic Curve Cryptography (ECC) [2] and Lattice-based Cryptography (LBC) [3]. RNS has also been widely used in DSP units, such as digital Finite Impulse Response (FIR) filter in [4,5], adaptive filter in [6], 8-point [7,8], and 16-point [8] Discrete Cosine Transforms (DCT) and Discrete Fourier Transforms (DFT) [9]. However, since the calculation of RNS is defined on the modular operations, there are challenges in some basic operations, such as sign detection [10,11], magnitude comparison [12], residue-to-binary (R/B) conversion [13], and scaling [14,15], which limits the wide application of RNS.

In DSP application, overflow in fixed point representation is a common issue when the dynamic range is limited. It mostly happens in multiplication and addition operations, especially in applications with cascaded architecture, such as Fast Fourier Transform (FFT). For TCS, this issue can be easily addressed by expanding the bit-width of intermediate computation results and then converting it back to the original bit-width. This means that the precision of input operands will not lose, and the computation accuracy is only

determined by the last conversion step. Meanwhile, the bit-width expansion step is very simple. In a word, the overflow can be solved simply and accurately by TCS fixed-point calculation.

However, since the dynamic range of the RNS is determined by its moduli set and the dynamic expanding in RNS is difficult, it is much more complicated to avoid overflow compared to TCS. Usually, there are three approaches to handle the overflow problem in RNS. Figure 1 gives the basic structure of these three approaches in a three-moduli RNS, for example.

(1) The first approach is based on scaling, as shown in Figure 1a. The input operands are firstly scaled down to ensure the product result falls in the dynamic range of the RNS [16]. We call this multiplier a basic scaling RNS multiplier. Unfortunately, the scaling operation definitely reduces the precision of the input, resulting in a loss of precision of the product result.

(2) The second approach is based on base conversion, as shown in Figure 1b. The input operands are firstly converted to a new RNS with larger dynamic range to avoid the overflow problem, the multiplication results are then scaled down to the original RNS. This approach will be helpful in some applications, such as the FIR filter. However, in some DSP algorithms with cascaded multiplication or feedback structure, such as FFT computation and IIR filter, the overall dynamic range can vary significantly. Thus, the overhead of base conversion will become unacceptable.

(3) The last approach is based on base extension, as shown in Figure 1c. When the dynamic range is not enough, one or more bases will be added to extend the dynamic range of original RNS. The base extension operation is still too complicated to be acceptable.

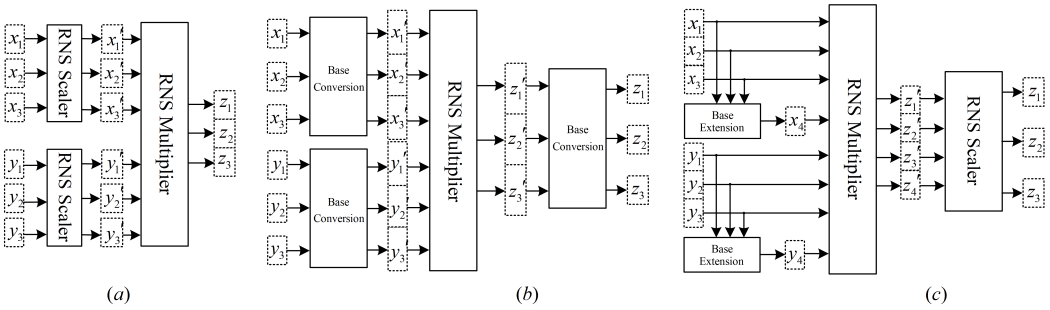

Figure 1. Three traditional overflow-free RNS multipliers. (**a**) Scaling-based scheme, (**b**) Base Conversion-based scheme, (**c**) Base Extension-based scheme

All of the above approaches cannot achieve similar performance to that in TCS. The first will lose accuracy, while the latter two will require complex algorithms and huge hardware resources.

In the RNS multiplier research, previous work mainly focuses on the efficient implementation of specific moduli. Chen [17] proposed an efficient modulo $2^n + 1$ multiplier. Muralidharan [18] proposed a high dynamic range modulo $2^n - 1$ multiplier. Zimmermann [19] proposed a joint implementation of the modulo $(2^n \pm 1)$ multiplier. Hiasat [20] proposed a generic multiplier for any modulo. Although these implementations can efficiently and accurately calculate the modular multiplication in each RNS channel, the precision loss caused by scaling before modular multiplier is ignored. These designs didn't consider the overflow problem caused by multipliers in specific DSP applications.

In this paper, we propose a high precision overflow-free RNS multiplier design method. The proposed RNS multiplier uses a similar idea of avoiding overflow in TCS to get high computation accuracy and low complexity. Throughout this paper, we choose the common used high precision RNS multiplier for three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, which is widely used in RNS, to illustrate our idea for RNS multiplier design. The proposed multiplier improves the calculation precision by adding several compensation items to improve the precision of the result calculated by the scaled inputs. The compensation

items can be obtained directly from preceding RNS scalers with little extra effort. Figure 2 illustrates the concise structure of the proposed RNS multiplier.

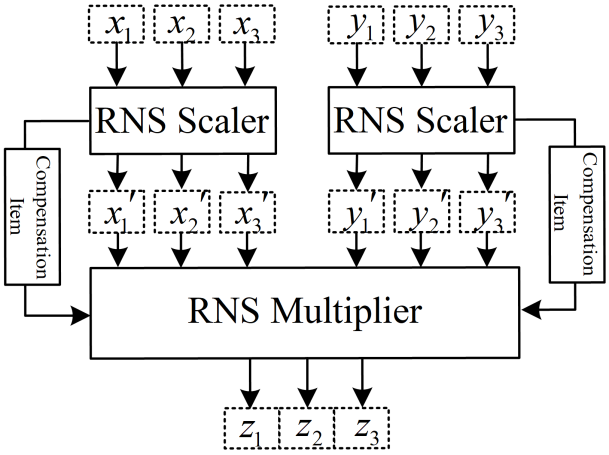

Figure 2. The proposed high precision overflow-free RNS multiplier structure.

The rest of this paper is arranged as follows. In Section 2, we introduce the basic theory of RNS. In Section 3, we propose two joint scalers and two high precision RNS multipliers. In Section 4, we explore the structure of proposed multipliers and scalers. In Section 5, we analyze the calculation performance of the proposed RNS multiplier and implement the multiplier in Very Large Scale Integration circuits (VLSI) to explore its hardware performance. Finally, the paper is summarized in Section 6.

2. Introduction of RNS

RNS is defined by a moduli set $\{m_1, m_2, \ldots, m_L\}$, where m_i and m_j $(i, j = 1, 2, ..., L)$ are coprime when $i \neq j$. An integer X can be represented as

$$X \triangleq \{x_1, x_2, \ldots, x_L\}, \tag{1}$$

where x_i is the residue of X mod m_i, we denote it as $x_i = \langle X \rangle_{m_i}$. Let $M = \prod_{i=1}^{n} m_i$, then M is called as the dynamic range of the RNS, that is, $X \in [0, M-1]$. According to the rules of modular operation [21], for operands X and Y, we have

$$\begin{aligned} \langle X \pm Y \rangle_m &= \left\langle \langle X \rangle_m \pm \langle Y \rangle_m \right\rangle_m \\ \langle X \bullet Y \rangle_m &= \left\langle \langle X \rangle_m \bullet \langle Y \rangle_m \right\rangle_m \\ \langle kX \rangle_{km} &= k \langle X \rangle_m \end{aligned}. \tag{2}$$

For two coprime moduli, m_1 and m_2, the modular operation has properties as

$$\left\langle \langle X \rangle_{m_1 m_2} \right\rangle_{m_1} = \langle X \rangle_{m_1}. \tag{3}$$

The Chinese Remainder Theorem (CRT) is one of the fundamental theorems of RNS, which is common used in scaling, Residue to Binary (R/B) conversion, and so on. If an integer $X \in [0, M-1]$, then

$$X = \left\langle \sum_{i=1}^{n} x_i M_i \langle M_i^{-1} \rangle_{m_i} \right\rangle_M, \tag{4}$$

where $M_i = M/m_i$, and $\langle M_i^{-1} \rangle_{m_i}$ is the multiplicative inverse of M_i for m_i, that is, $\left\langle M_i \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_i} = 1$.

In this paper, our proposed scalers and multipliers are dedicated to the RNS $\{2^n-1, 2^n, 2^n+1\}$. For ease of notation, we let $m_1 = 2^n-1$, $m_2 = 2^n$ and $m_3 = 2^n+1$ so that the residues $x_1 = \langle X \rangle_{m_1}$, $x_2 = \langle X \rangle_{m_2}$ and $x_3 = \langle X \rangle_{m_3}$.

Scaling is actually a constant division operation and the divisor is called the scaling factor. The format of the moduli set and the scaling factor are the two main factors in the complexity of the scaler. For an integer X, if scaling factor is K, the scaling result can be computed by

$$Y = \left\lfloor \frac{X}{K} \right\rfloor = \frac{X - \langle X \rangle_K}{K}, \tag{5}$$

where $\lfloor \cdot \rfloor$ represents floor operation. Different from TCS, the operand X in RNS is represented by multiple residues, and the final scaling result should also be represented by multiple residues.

For RNS with moduli set $\{m_1, m_2, m_3\} = \{2^n-1, 2^n, 2^n+1\}$, we have

$$\begin{aligned} M &= m_1 \times m_2 \times m_3 = 2^{3n} - 2^n \\ M_i &= \{2^{2n} + 2^n, 2^{2n} - 1, 2^{2n} - 2^n\}. \\ \left\langle M_i^{-1} \right\rangle &= \{2^{n-1}, -1, 2^{n-1}+1\} \end{aligned} \tag{6}$$

in which, $i = 1,\ 2,\ 3$. Substituting (6) into (4), we can get

$$\begin{aligned} X &= \langle 2^{n-1}(2^{2n}+2^n)x_1 - (2^{2n}-1)x_2 + (2^{n-1}+1)(2^{2n}-2^n)x_3 \rangle_M \\ &= 2^{n-1}(2^{2n}+2^n)x_1 - (2^{2n}-1)x_2 + (2^{n-1}+1)(2^{2n}-2^n)x_3 - IM' \end{aligned} \tag{7}$$

where I is an integer to ensure $0 \le X \le M-1$.

3. Design of High Precision RNS Multiplier

3.1. Joint RNS Scaler for Moduli Set $\{2^n-1, 2^n, 2^n+1\}$

The proposed RNS multiplier needs the scaling result of scaling factors m_1, m_3, m_1m_2, and m_2m_3. Generally, we need four scalers to implement them. To further reduce the hardware complexity, we propose two joint scalers which can get scaling results for two scaling factors at the same time, one of which is for scaling factors m_1 and m_1m_2, and the other is for scaling factors m_3 and m_2m_3. As shown in the following derivation, the scaling results of scaling factors m_1m_2 and m_2m_3 can be obtained from intermediate products of m_1 scaler and m_3 scaler, respectively. As such, we can greatly save the hardware consumption by combining them with two joint scalers.

According to (7), we derive four calculation methods for these four scaling factors, m_1, m_3, m_2m_3, and m_1m_2, respectively.

3.1.1. Scaling Factor $K = m_3$

According to (7), the scaling operation can be represented as

$$\frac{X}{m_3} = 2^{2n-1}x_1 - (2^n-1)x_2 + (2^{2n-1}-1)x_3 - Im_1m_2 + \frac{x_3}{m_3}. \tag{8}$$

In RNS, all residues are integers, and $x_3 < m_3$, so $x_3/m_3 < 1$. Thus, we can round them down (8) and get

$$\begin{aligned} \left\lfloor \frac{X}{m_3} \right\rfloor &= 2^{2n-1}x_1 - (2^n-1)x_2 + (2^{2n-1}-1)x_3 - Im_1m_2 \\ &= \left\langle 2^{2n-1}x_1 - (2^n-1)x_2 + (2^{2n-1}-1)x_3 \right\rangle_{m_1m_2} \end{aligned}. \tag{9}$$

Mapping (9) to residue channels m_1, m_2, and simplify them, we can obtain

$$\begin{aligned} X_{3,1} &= \left\langle \left\lfloor \frac{X}{m_3} \right\rfloor \right\rangle_{m_1} \\ &= \left\langle 2^{n-1}x_1 + (2^{n-1}-1)x_3 \right\rangle_{m_1} \\ &= \left\langle 2^{n-1}x_1 - 2^{n-1}x_3 \right\rangle_{m_1} \\ X_{3,2} &= \left\langle \left\lfloor \frac{X}{m_3} \right\rfloor \right\rangle_{m_2} \\ &= \langle x_2 - x_3 \rangle_{m_2} \end{aligned}, \tag{10}$$

Because $0 \le X \le M-1$, then $0 \le \lfloor X/m_3 \rfloor \le m_1m_2 - 1$. By using CRT, we can uniquely represent $\lfloor X/m_3 \rfloor$ with the remainders of channels m_1 and m_2. However, in most DSP-oriented applications, the remainder of channel m_3 is also required to match the original three moduli set. Then,

$$\begin{aligned} \left\lfloor \frac{X}{m_3} \right\rfloor &= \langle 2^n X_{3,1} - (2^n-1)X_{3,2} \rangle_{m_1m_2} \\ &= \left\langle 2^n \langle X_{3,1} - X_{3,2} \rangle_{m_1} + X_{3,2} \right\rangle_{m_1m_2} \end{aligned}. \tag{11}$$

Because $0 \le 2^n \langle X_{3,1} - X_{3,2} \rangle_{m_1} + X_{3,2} < m_1m_2$, we can get

$$\begin{aligned} X_{3,3} &= \left\langle 2^n \langle X_{3,1} - X_{3,2} \rangle_{m_1} + X_{3,2} \right\rangle_{m_3} \\ &= \left\langle -\langle X_{3,1} - X_{3,2} \rangle_{m_1} + X_{3,2} \right\rangle_{m_3} \end{aligned}. \tag{12}$$

3.1.2. Scaling Factor $K = m_1$

Similarly with factor $K = m_3$, scaling for $K = m_1$ can be represented as

$$\frac{X}{m_1} = \left\langle (2^{2n-1}+2^n+1)x_1 - (2^n+1)x_2 + 2^n(2^{n-1}+1)x_3 \right\rangle_{m_2m_3} + \frac{x_1}{m_1}. \tag{13}$$

Then, the values in residue channels m_2 and m_3 are

$$\begin{aligned} X_{1,2} &= \left\langle \left\lfloor \frac{X}{m_1} \right\rfloor \right\rangle_{m_2} \\ &= \langle x_1 - x_2 \rangle_{m_2} \\ X_{1,3} &= \left\langle \left\lfloor \frac{X}{m_1} \right\rfloor \right\rangle_{m_3} \\ &= \left\langle 2^{n-1}x_3 - 2^{n-1}x_1 \right\rangle_{m_3} \end{aligned}. \tag{14}$$

For the precise representation of $\lfloor X/m_1 \rfloor$, the residues in channel m_2 and m_3 are enough. However, in most applications, the residue in channel m_1 is still needed in the following processing. Thus, by using CRT, we can get

$$\left\lfloor \frac{X}{m_1} \right\rfloor = \left\langle 2^n \langle X_{1,2} - X_{1,3} \rangle_{m_3} + X_{1,2} \right\rangle_{m_2m_3}. \tag{15}$$

Because $0 \le 2^n \langle X_{1,2} - X_{1,3} \rangle_{m_3} + X_{1,2} < m_2m_3$, we can get

$$X_{1,1} = \left\langle \langle X_{1,2} - X_{1,3} \rangle_{m_3} + X_{1,2} \right\rangle_{m_2m_3}. \tag{16}$$

3.1.3. Scaling Factor $K = m_2m_3$

Low proposed two scaling structures for scaling factor $K = m_2m_3$ in [22,23], respectively. However, they have a unit error when $x_1 < x_2$. In this paper, we propose a scaling structure which can get the scaling results accurately. According to (7), we can get

$$\frac{X}{m_2m_3} = 2^{n-1}x_1 - x_2 - 2^{n-1}x_3 + x_3 - Im_1 + \frac{x_2 - x_3}{m_2} + \frac{x_3}{m_2m_3}. \tag{17}$$

x_2 and x_3 are non-negative integers which are defined on the radixes of m_2 and m_3. Thus, if and only if $x_2 - x_3 < 0$, $(x_2 - x_3)/m_2 + x_3/(m_2m_3) < 0$. Then, from (17), we can get

$$\left\lfloor \frac{X}{m_2(m_3)} \right\rfloor = \begin{cases} \left\langle 2^{n-1}x_1 - x_2 - 2^{n-1}x_3 + x_3 \right\rangle_{m_1} & x_2 - x_3 \geq 0 \\ \left\langle 2^{n-1}x_1 - x_2 - 2^{n-1}x_3 + x_3 - 1 \right\rangle_{m_1} & x_2 - x_3 < 0 \end{cases}. \tag{18}$$

When $x_2 - x_3 < 0$, according to (18),

$$\left\langle 2^{n-1}x_1 - x_2 - 2^{n-1}x_3 + x_3 - 1 \right\rangle_{m_1} = \left\langle 2^{n-1}x_1 - 2^{n-1}x_3 - (x_2 - x_3 + 2^n) \right\rangle_{m_1}. \tag{19}$$

Thus, (19) can be converted to

$$\begin{aligned} \left\lfloor \frac{X}{2^n(2^n+1)} \right\rfloor &= \left\langle 2^{n-1}x_1 - 2^{n-1}x_3 - \langle x_2 - x_3 \rangle_{m_2} \right\rangle_{m_1}. \\ &= \langle X_{3,1} - X_{3,2} \rangle_{m_1} \end{aligned} \tag{20}$$

When $x_2 - x_3 > 0$, it is obviously that $x_2 - x_3$ has the same dynamic range with x_2, so the modulo m_2 operations in (20) do not change the calculation result. These two situations can be combined into (20).

From (20), we can see that the scaling result of scaling factor m_2m_3 can be calculated by the result from the scaler with scaling factor m_2. Because $X \in [0, M-1]$, $0 \leq \lfloor X/2^n(2^n+1) \rfloor < m_1$, the results in all channels of moduli set $\{m_1, m_2, m_3\}$ are $\lfloor X/2^n(2^n+1) \rfloor$.

3.1.4. Scaling Factor $K = m_1m_2$

Scaler with scaling factor m_1m_2 has a similar structure with that of m_2m_3, From (7) we can get

$$\frac{X}{2^n(2^n-1)} = x_1 - x_2 + 2^{n-1}x_1 - 2^{n-1}x_3 - Im_3 + \frac{x_1 - x_2}{m_2} + \frac{x_1}{m_1m_2}. \tag{21}$$

If and only if $x_1 - x_2 < 0$, $(x_1 - x_2)/m_2 + x_1/(m_1m_2) < 0$, we have

$$\left\lfloor \frac{X}{2^n(2^n-1)} \right\rfloor = \left\langle (2^n + x_1 - x_2) + 2^{n-1}x_1 - 2^{n-1}x_3 \right\rangle_{m_3}. \tag{22}$$

Then, (22) can be converted to

$$\begin{aligned} \left\lfloor \frac{X}{2^n(2^n-1)} \right\rfloor &= \left\langle \langle x_1 - x_2 \rangle_{m_2} + 2^{n-1}x_1 - 2^{n-1}x_3 \right\rangle_{m_3}. \\ &= \langle X_{1,2} - X_{1,3} \rangle_{m_3} \end{aligned} \tag{23}$$

The same as with scaling factor m_2m_3, the situation $x_1 - x_2 > 0$ can be combined into (23).

From (23), we can see that the scaler with scaling factor m_1m_2 can also be calculated by the result from the scaler with scaling factor m_1. Because $0 \leq X < M$, $0 \leq$

$\lfloor X/2^n(2^n-1)\rfloor < m_3$. We can use the value in channel m_3 to represent it. If we need to expose it to channels m_1 and m_2, it can be simply implemented by modular operations.

In summary, the calculation methods of these four different scaling factors are shown in Table 1.

Table 1. Calculation methods for four different scaling factors.

K	$\langle\lfloor X/K\rfloor\rangle_{m_1}$	$\langle\lfloor X/K\rfloor\rangle_{m_2}$	$\langle\lfloor X/K\rfloor\rangle_{m_3}$
m_3	$\langle 2^{n-1}x_1 - 2^{n-1}x_3\rangle_{m_1}$	$\langle x_2 - x_3\rangle_{m_2}$	$\left\langle -\langle X_{3,1} - X_{3,2}\rangle_{m_1} + X_{3,2}\right\rangle_{m_3}$
m_1	$\left\langle \langle X_{1,2} - X_{1,3}\rangle_{m_3} + X_{1,2}\right\rangle_{m_1}$	$\langle x_1 - x_2\rangle_{m_2}$	$\langle 2^{n-1}x_3 - 2^{n-1}x_1\rangle_{m_3}$
m_2m_3	$\langle X_{3,1} - X_{3,2}\rangle_{m_1}$	$\langle X_{3,1} - X_{3,2}\rangle_{m_1}$	$\langle X_{3,1} - X_{3,2}\rangle_{m_1}$
m_1m_2	$\left\langle \langle X_{1,2} - X_{1,3}\rangle_{m_3}\right\rangle_{m_1}$	$\left\langle \langle X_{1,2} - X_{1,3}\rangle_{m_3}\right\rangle_{m_2}$	$\langle X_{1,2} - X_{1,3}\rangle_{m_3}$

3.2. High Precision RNS Multipliers for Moduli Set $\{2^n-1, 2^n, 2^n+1\}$

We propose two high precision RNS multipliers based on CRT. For multiplicands X and Y defined on the moduli set $\{m_1, m_2, m_3\}$, the product result is scaled by a fixed scaling factor $M = m_1m_2m_3$. Since $0 \le \lfloor X\cdot Y/M\rfloor < M$, we can guarantee that the result is still in the dynamic range of the RNS. As shown in Figure 2, we add several complementary items to the result of Figure 1a to get a high precision multiplication result. We can see from the following derivation how the complementary items work to obtain high precision. As mentioned before, the adding complementary items can be obtained directly from the proposed scalers, so the proposed RNS multiplier needs less extra hardware consumption in comparison with the basic scaling multiplier.

The derivation process is as follows.

According to (21), we have

$$\begin{aligned}\frac{Y}{m_1m_2} &= \begin{cases} \left\lfloor \frac{Y}{m_1m_2}\right\rfloor + \frac{y_1-y_2}{m_2} + \frac{y_1}{m_1m_2} & \text{when} \quad y_1 \ge y_2 \\ \left\lfloor \frac{Y}{m_1m_2}\right\rfloor + \frac{y_1-y_2+m_2}{m_2} + \frac{y_1}{m_1m_2} & \text{when} \quad y_1 < y_2 \end{cases} \\ &= \left\lfloor \frac{Y}{m_1m_2}\right\rfloor + \frac{\langle y_1-y_2\rangle_{m_2}}{m_2} + \frac{y_1}{m_1m_2} \\ &= \left\lfloor \frac{Y}{m_1m_2}\right\rfloor + \frac{m_1Y_{1,2}+y_1}{m_1m_2}\end{aligned}. \tag{24}$$

By using (8), (21), and (24), we can get

$$\begin{aligned}\frac{X\cdot Y}{M} &= \frac{X}{m_3}\cdot\frac{Y}{m_1m_2} \\ &= \left(\left\lfloor\frac{X}{m_3}\right\rfloor + \frac{x_3}{m_3}\right)\left(\left\lfloor\frac{Y}{m_1m_2}\right\rfloor + \frac{m_1Y_{12}+y_1}{m_1m_2}\right) \\ &= \left\lfloor\frac{X}{m_3}\right\rfloor\left\lfloor\frac{Y}{m_1m_2}\right\rfloor + \frac{x_3}{m_3}\left\lfloor\frac{Y}{m_1m_2}\right\rfloor + \frac{X}{m_3}\left(\frac{Y_{1,2}}{m_2} + \frac{y_1}{m_1m_2}\right)\end{aligned}, \tag{25}$$

According to (17), we have

$$\frac{X}{m_2m_3} = \left\lfloor\frac{X}{m_2m_3}\right\rfloor + \frac{X_{3,2}}{m_2} + \frac{x_3}{m_2m_3}. \tag{26}$$

Thus, the last add item in (25) can be expanded as

$$\begin{aligned}
&\frac{X}{m_3}\left(\frac{Y_{1,2}}{m_2}+\frac{y_2}{m_1m_2}\right)\\
&=\frac{X}{m_2m_3}\left(Y_{1,2}+\frac{y_1}{m_1}\right)\\
&=\left(\left\lfloor\frac{X}{m_2m_3}\right\rfloor+\frac{X_{3,2}}{m_2}+\frac{x_3}{m_2m_3}\right)\left(Y_{1,2}+\frac{y_1}{m_1}\right)\\
&=\left\lfloor\frac{X}{m_2m_3}\right\rfloor Y_{1,2}+\frac{X_{3,2}Y_{1,2}}{m_2}+\frac{Y_{1,2}x_3}{m_2m_3}+\frac{y_1}{m_1}\left\lfloor\frac{X}{m_2m_3}\right\rfloor+\frac{X_{3,2}y_1}{m_1m_2}+\frac{x_3y_1}{m_1m_2m_3}
\end{aligned}. \tag{27}$$

Then, (25) can be rewritten as

$$\begin{aligned}
\frac{X\cdot Y}{M}=&\left\lfloor\frac{X}{m_3}\right\rfloor\left\lfloor\frac{Y}{m_1m_2}\right\rfloor+\frac{x_3}{m_3}\left\lfloor\frac{Y}{m_1m_2}\right\rfloor+\left\lfloor\frac{X}{m_2m_3}\right\rfloor Y_{1,2}\\
&+\frac{X_{3,2}Y_{1,2}}{m_2}+\frac{Y_{1,2}x_3}{m_2m_3}+\frac{y_1}{m_1}\left\lfloor\frac{X}{m_2m_3}\right\rfloor\\
&+\frac{X_{3,2}y_1}{m_1m_2}+\frac{x_3y_1}{m_1m_2m_3}
\end{aligned}. \tag{28}$$

We can find that the division operations in (28) have a divisor that is not the power of 2, which can be difficult to implement. In order to reduce hardware complexity, we approximately represent the divisors in $\frac{x_3}{m_3}\left\lfloor\frac{Y}{m_1m_2}\right\rfloor$ and $\frac{y_1}{m_1}\left\lfloor\frac{X}{m_2m_3}\right\rfloor$ by m_2. We maintain $\frac{X_{3,2}Y_{1,2}}{m_2}$ and abandon $\frac{Y_{1,2}x_3}{m_2m_3}$, $\frac{X_{3,2}y_1}{m_1m_2}$ and $\frac{x_3y_1}{m_1m_2m_3}$ as approximate errors , for these three items are smaller than 1. After these simplifications, (28) can be approximately expressed as

$$\frac{X\cdot Y}{M}\approx\left\lfloor\frac{X}{m_3}\right\rfloor\left\lfloor\frac{Y}{m_1m_2}\right\rfloor+Y_{1,2}\left\lfloor\frac{X}{m_2m_3}\right\rfloor+\Delta_1, \tag{29}$$

where

$$\Delta_1=\left\lfloor\frac{Y}{m_1m_2}\right\rfloor\frac{x_3}{m_2}+\left\lfloor\frac{X}{m_2m_3}\right\rfloor\frac{y_1}{m_2}+\frac{Y_{1,2}X_{3,2}}{m_2}. \tag{30}$$

The first two items to add in (29) only have integer multiplications and all the multipliers can be obtained directly from the scalers. Moreover, while (30) has decimals, all the divisors are powers of 2, which can be simply implemented by shifting. The totally approximate error in (29) is

$$\delta_1=\frac{x_3}{m_2m_3}\left(Y_{1,2}-\left\lfloor\frac{Y}{m_1m_2}\right\rfloor\right)+\frac{y_1}{m_1m_2}\left(X_{3,2}+\left\lfloor\frac{X}{m_2m_3}\right\rfloor\right)+\frac{x_3y_1}{m_1m_2m_3}. \tag{31}$$

Moreover, this kind of multiplier can also be realized by another method, which is similar to the above derivation procedure. According to (13) and (17), we can get

$$\begin{aligned}
\frac{X\cdot Y}{M}&=\frac{X}{m_1}\cdot\frac{Y}{m_2m_3}\\
&=\left\lfloor\frac{X}{m_1}\right\rfloor\left\lfloor\frac{Y}{m_2m_3}\right\rfloor+\frac{x_1}{m_1}\left\lfloor\frac{Y}{m_2m_3}\right\rfloor+\frac{X}{m_1}\left(\frac{Y_{3,2}}{m_2}+\frac{y_3}{m_2m_3}\right)
\end{aligned}. \tag{32}$$

In the same way, (32) can be approximated as

$$\frac{X\cdot Y}{M}\approx\left\lfloor\frac{X}{m_1}\right\rfloor\left\lfloor\frac{Y}{m_2m_3}\right\rfloor+Y_{3,2}\left\lfloor\frac{X}{m_1m_2}\right\rfloor+\Delta_2, \tag{33}$$

where

$$\Delta_2 = \left\lfloor \frac{X}{m_1 m_2} \right\rfloor \frac{y_3}{m_2} + \left\lfloor \frac{Y}{m_2 m_3} \right\rfloor \frac{x_1}{m_2} + \frac{Y_{3,2} X_{1,2}}{m_2}. \tag{34}$$

In addition, the error of (33) is

$$\delta_2 = \frac{x_1}{m_1 m_2} \left(Y_{3,2} + \left\lfloor \frac{Y}{m_2 m_3} \right\rfloor \right) + \frac{y_3}{m_2 m_3} \left(X_{1,2} - \left\lfloor \frac{X}{m_1 m_2} \right\rfloor \right) + \frac{x_1 y_3}{m_1 m_2 m_3}. \tag{35}$$

Although the multipliers implemented by (29) and (33) have the same structure, they use different scalers, which leads to different hardware complexity. It is worth noting that the hardware complexity can also be reduced by reducing the number of add items in the multipliers with the cost of precision loss. For example, since the complementary items in (30) and (34) are relatively small but consume a lot of hardware resources, we can abandon them and get a simplified RNS multiplier. This provides a trade-off between calculation precision and hardware consumption. From now on, we call the proposed RNS multiplier according to (29) and (33) as a full compensatory RNS multiplier, and the RNS multipliers implemented by abandoning (30) and (34) are represented by a partial compensatory RNS multiplier. The following numerical example is used to illustrate our proposed RNS multiplication algorithm. Letting $X = 64$ and $Y = 460$, Table 2 shows the detailed calculation traces of the RNS multiplication operation step by step based on (29).

Table 2. Computation traces of $X \times Y = 64 * 460$ of the proposed high precision RNS multiplier.

Moduli set	$\{7, 8, 9\}$
Residues,$\{x_1, x_2, x_3\}$,$\{y_1, y_2, y_3\}$	$\{1, 0, 1\}$,$\{5, 4, 1\}$
$\left\lfloor \frac{X}{m_3} \right\rfloor = \{X_{3,1}, X_{3,2}, X_{3,3}\}$	$\{0, 7, 7\}$
$\left\lfloor \frac{Y}{m_1} \right\rfloor = \{Y_{1,1}, Y_{1,2}, Y_{1,3}\}$	$\{2, 1, 2\}$
$\left\lfloor \frac{X}{m_2 m_3} \right\rfloor$	0
$\left\lfloor \frac{Y}{m_1 m_2} \right\rfloor$	8
$\left\lfloor \frac{Y}{m_1 m_2} \right\rfloor \left\lfloor \frac{X}{m_3} \right\rfloor$	$\{0, 0, 2\}$
$Y_{1,2} \left\lfloor \frac{X}{m_2 m_3} \right\rfloor$	$\{0, 0, 0\}$
$\left\lfloor \frac{Y}{m_1 m_2} \right\rfloor \frac{x_3}{m_2}$	1
$\left\lfloor \frac{X}{m_2 m_3} \right\rfloor \frac{y_1}{m_2}$	0
$\frac{Y_{1,2} X_{3,2}}{m_2}$	0
(29)	$\{0, 0, 2\} + 1 = \{1, 1, 3\} = 57$
calculation error	$64 \times 460 - 57 * 504 = 712$

4. Hardware Structures

For the RNS with moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, some numerical calculations and operations can be replaced by bit-wise logic operation to achieve an efficient hardware structure. Let the binary representation of an n-bit integer x be $x_{n-1} x_{n-2} \ldots x_0$ and $0 \leq x < 2^n - 1$, according to the properties of the modulo operations and the Boolean logic operation rules [14], we can easily get

$$\begin{aligned} \langle -x \rangle_{2^n - 1} &= \langle \overline{x} \rangle_{2^n - 1} \\ \langle 2^r x \rangle_{2^n - 1} &= \mathrm{CLS}(x, r)' \end{aligned} \tag{36}$$

where $\overline{x}$ represents bit-wise inversion of binary numbers, and $\mathrm{CLS}(x, r)$ indicates that integer x is shifted to the left by r bits.

The above two operations for modulo $2^n + 1$ will become a little bit complicated. For an $(n+1)$-bit integer with binary representation $x = x_n x_{n-1} ... x_0$ and $0 \leq x < 2^n + 1$, then

$$\begin{aligned} \langle -x \rangle_{2^n+1} &= \left\langle 2^n + 1 - \sum_{i=0}^{n} 2^i x_i \right\rangle_{2^n+1} \\ &= \left\langle 2 + 2^n - 1 + x_n - \sum_{i=0}^{n} 2^i x_i \right\rangle_{2^n+1} \\ &= \langle 2 + x_n + \overline{x_{n-1} x_{n-2} ... x_0} \rangle_{2^n+1}, \end{aligned} \tag{37}$$

and

$$\begin{aligned} \langle 2^r x \rangle_{2^n+1} &= \left\langle \sum_{i=0}^{n} 2^{i+r} x_i \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-r-1} 2^{i+r} x_i + \sum_{i=n-r}^{n} 2^{i+r} x_i \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-r-1} 2^{i+r} x_i + 2^n \sum_{i=n-r}^{n} 2^{i-n+r} x_i \right\rangle_{2^n+1} \\ &= \left\langle \sum_{i=0}^{n-r-1} 2^{i+r} x_i + (2^n - 1) - \sum_{i=n-r}^{n} 2^{i-n+r} x_i + 2 \right\rangle_{2^n+1} \\ &= \left\langle x_{n-r-1} ... x_1 x_0 \underbrace{000...0}_{r} + \underbrace{111...1}_{n-r-1} \overline{x_n x_{n-1} ... x_{n-r}} + 2 \right\rangle_{2^n+1}. \end{aligned} \tag{38}$$

The hardware structure of the scalers and multipliers in this paper are based on these operations.

4.1. Multiplier Hardware Structure

The implementation block diagrams of the proposed compensatory scaling RNS multipliers are shown in Figure 3.

In Figure 3, the multiplications in solid box represent the basic items of the proposed RNS multiplier, while the multiplications in dotted box represent the compensation items. The scaler blocks in Figure 3 are all proposed joint scalers, and the implementation details are shown in Figure 4. As shown in Figure 3, the final result is calculated by the product of two scalers without any other effort. Thus, this just costs a little extra hardware consumption.

4.2. Scaler Hardware Structure

We designed the scaling structures of the two joint scalers proposed in this paper. In addition, the implementation block diagram is shown in Figure 4.

In Figure 4, the proposed scaling structures of these four scaling factors are implemented using logic operations in (36) and modular adders proposed in [19]. The module in the two dotted line box can be used separately as scaler implementations of scaling factor $m_2 m_3$ and $m_1 m_2$, respectively. The overall implementation in Figure 4a can be used as a joint implementation for scaling factors $m_2 m_3$ and m_3, while Figure 4b can be used for scaling factors $m_1 m_2$ and m_1.

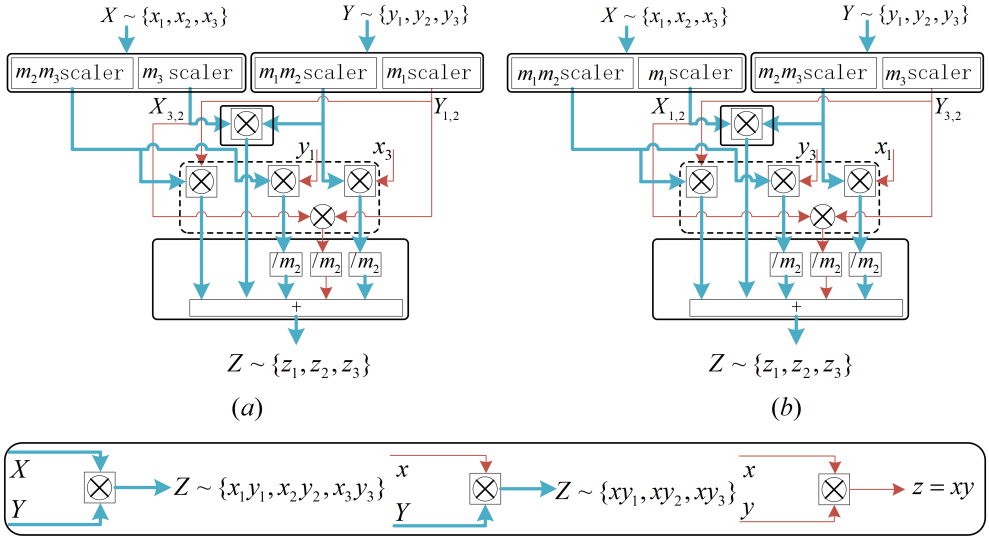

Figure 3. Compensatory scaling RNS multiplier structure. (**a**) multiplier structure corresponding to (29), (**b**) multiplier structure corresponding to (33).

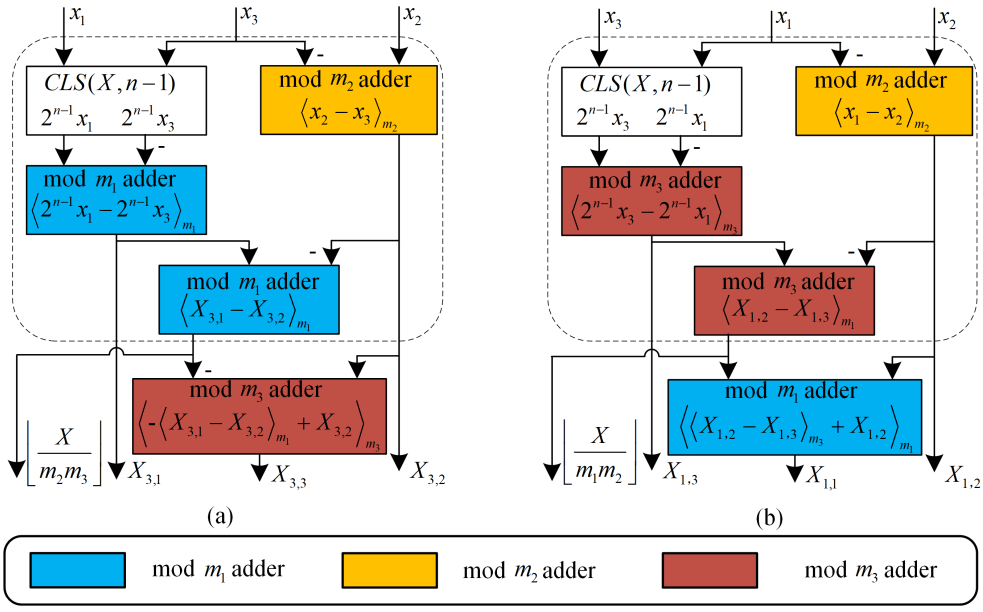

Figure 4. Scaling structures of four scaling factors. (**a**) scaling factor $2^n + 1$ and $2^n(2^n + 1)$ joint scaling structure; (**b**) scaling factor $2^n - 1$ and $2^n(2^n - 1)$ joint scaling structure.

Figure 4a can be further optimized, since the cascade modulo m_1 can be replaced by a Carry Save Adder (CSA) and a modulo m_1 adder. This optimized structure is shown in Figure 5. It is worth noting that we use this optimized structure in the implementation of Figure 4a in addition to scaling factor m_2m_3, in order to get a better hardware performance.

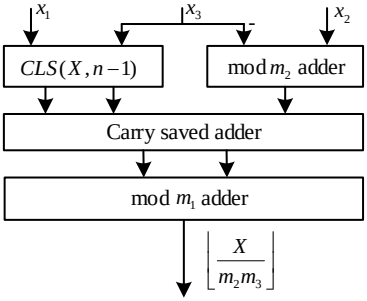

Figure 5. Optimized scaling structure for scaling factor $m_2 m_3$.

5. Performance Analysis

5.1. Calculation Performance of the Proposed Multiplier

In order to verify the calculation performance of the proposed compensatory scaling multiplier, we firstly analyze the calculation precision of the proposed multiplier compared with the TCS multiplier and the basic scaling RNS multiplier (see Figure 1a. Two metrics, MSE (mean-square-error), and SNR (Signal-to-Noise) are used to evaluate the calculation precision of three multipliers included. The MSE is calculated by:

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=0}^{N-1} \left(X_{real,i,norm} - X_{result,i,norm} \right)^2, \tag{39}$$

while SNR is defined as:

$$\mathrm{SNR(dB)} = 10 \log_{10} \left(\frac{\sum_{i=0}^{N-1} {X_{real,i}}^2}{\sum_{i=0}^{N-1} \left(X_{real,i} - X_{result,i} \right)^2} \right). \tag{40}$$

In (39) and (40), $X_{real,i}$ represents real results of ith calculation, $X_{result,i}$ represents ith result calculated by the three multipliers, while $X_{real,i,norm}$ and $X_{result,i,norm}$ represent the unit normalization of $X_{real,i}$ and $X_{result,i}$, respectively. For each i, the inputs of the multiplier are randomly selected from RNS's dynamic range. N is the total number of samples calculated for each n, in this paper, $N = 10{,}000$. The dynamic range of RNS is changed by n from 5 to 12. For comparison, the bit-width k of TCS is selected to guarantee that the dynamic range of TCS is slightly bigger than that of RNS. For example, for $n = 5$, the moduli set of the RNS is $31, 32, 33$; then, $M = 32{,}736$, and we choose $k = floor(log_2 M + 1) = 15$ bit in order to compare with TCS. The simulation results are shown in Figures 6 and 7.

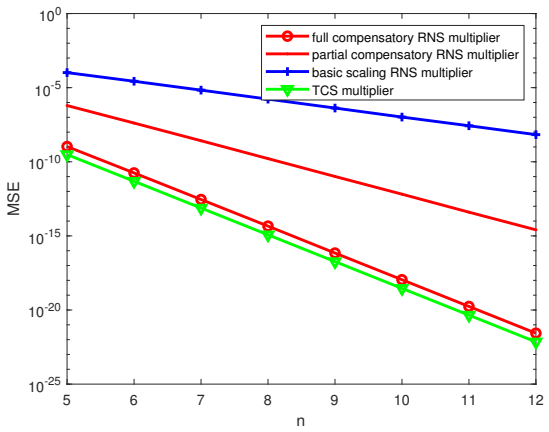

Figure 6. MSE of three kinds of multipliers.

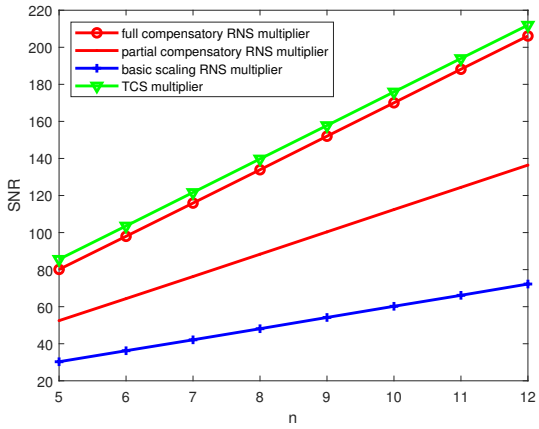

Figure 7. SNR of three kinds of multipliers.

From Figure 6, for all n, the proposed RNS multiplier achieves almost the same MSE with a TCS multiplier, while, for the basic scaling RNS multiplier, it suffers a relatively big MSE when $n < 8$. This means that the proposed RNS multiplier has a better calculation precision when the bit-width is not large enough.

As shown in Figure 7, for each n, the proposed RNS multiplier outperforms the traditional way by about 50–140 dB when n increases from 5 to 12, which reveals that the proposed RNS multiplier can greatly improve the calculation precision of the multiplication in RNS and avoid the overflow at the same time. In addition, the proposed multiplier achieves an SNR curve similar to that of the TCS multiplier, with an SNR loss of about 5 dB. This is because the dynamic range of TCS is chosen to be slightly larger than that of the RNS. Moreover, as mentioned before, the hardware consumption of proposed RNS multiplier can be saved by abandoning the third add item in (29) and (33). Although this can lead to a loss of calculation precision, the result shown in Figure 7 suggests that our partial compensatory RNS multiplier still outperforms the basic scaling one.

5.2. Synthesis Results of RNS Multiplier Based on Design Compiler

In order to evaluate the hardware performance of the proposed RNS multipliers, we designed them by VHDL and compiled them with Synopsys Design Compiler (DC) under the SMIC 65 nm process, respectively. The partial compensatory RNS multiplier with only one complementary item was also in consideration. We synthesized three mentioned multipliers in a clock timing constraint approach which uses the smallest clock period. The results are shown in Table 3.

In Table 3, Structure 1 means the structure in (29) and its simplified version, and Structure 2 means the structure in (33) and its simplified version. Although structures of two multipliers are symmetrical, they use different numbers of modulo m_1 and m_3 adders. Moreover, two cascaded modulo m_1 adders can benefit from the optimal hardware structure in Figure 5, so the hardware consumption of $S1$ is slightly larger than that of $S2$ for each n. BS means basic scaling RNS multiplier, PC means partial compensatory RNS multiplier and FC means full compensatory RNS multiplier. We use the AT of basic scaling RNS multiplier with structure 1 as a basis to calculate the AT ratio for each n. From Table 3, for each n, although the proposed multiplier needs about 2–2.3 times hardware consumption compared with the basic scaling multiplier, it can get about 2.6–3 times SNR than that of the basic scaling multiplier. To further evaluate the hardware efficiency of the proposed RNS multiplier, we calculate the SNR/AT of three multipliers. The results are shown in Figure 8. We can see that the proposed partial compensatory RNS multiplier has the biggest SNR/AT for each n, which means it has the highest hardware efficiency. In addition, the proposed RNS multiplier has bigger SNR/AT than that of the basic scaling one. This indicates that our proposed RNS multiplier still outperforms basic scaling RNS multiplier in hardware efficiency.

Table 3. Synthesis results of multipliers.

			A (μm^2)	T (ns)	$A * T$ ($\mu m^2 * ns$)	AT Ratio
$n = 5$	Structure 1	BS	4500.36	2.19	9889.31	1
		PC	5197.68	2.39	12,458.73	1.26
		FC	6280.92	2.69	16,942.90	1.71
	Structure 2	BS	3651.84	2.29	8399.15	0.85
		PC	4745.52	2.49	11,848.47	1.20
		FC	7913.88	2.59	20,570.07	2.08
$n = 8$	Structure 1	BS	7743.96	2.69	20,906.37	1
		PC	10,293.12	2.89	29,823.28	1.43
		FC	13,224.60	3.09	40,949.84	1.96
	Structure 2	BS	7428.24	2.69	20,021.11	0.96
		PC	10,254.96	2.89	23,453.74	1.12
		FC	16,380.36	3.03	49,757.47	2.38
$n = 11$	Structure 1	BS	12,677.76	2.89	36,704.90	1
		PC	17,025.48	3.09	52,766.21	1.43
		FC	23,253.84	3.29	76,688.60	2.09
	Structure 2	BS	12,537.72	2.99	37,606.26	1.02
		PC	15,290.28	3.19	48,888.83	1.33
		FC	26,150.76	3.29	86,222.45	2.35
$n = 14$	Structure 1	BS	16,674.12	3.18	53,148.76	1
		PC	26,294.76	3.39	90,232.63	1.70
		FC	31,066.20	3.59	111,744.81	2.10
	Structure 2	BS	15,747.12	3.29	51,894.63	0.98
		PC	26,294.76	3.39	89,279.12	1.68
		FC	32,858.64	3.59	118,271.72	2.23

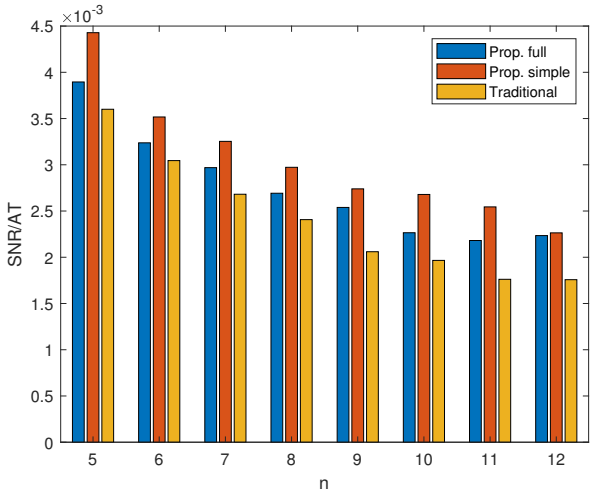

Figure 8. Hardware efficiency of two kinds of multipliers (higher is better).

Hisat proposes a few efficient RNS scalers for moduli set $\{2^n-1, 2^{n+p}, 2^n+1\}$ with scaling factor 2^n and 2^{n+p} [15]. This work is a recent development in RNS scaler design. The hardware efficiency of this work is obviously better than that of ours. As far as I know, our work in this paper is the first to discuss the problem of overall accuracy of the RNS multiplier, instead of the modular multiplier. In our design, we use the scaling results scaled by 2^n-1, $2^n(2^n-1)$, 2^n+1, and $2^n(2^n+1)$, instead of 2^n. Our focus in this work is not on optimizing the scaler. Any optimized scaler can be used in the proposed multiplier as long as it meets the requirements of scaling factor.

6. Conclusions

This work presented a high precision overflow-free multiplier design for three-moduli set $\{2^n-1, 2^n, 2^n+1\}$. The proposed RNS multiplier avoids overflow based on the scaling approach and achieves high precision by adding several compensation items to compensate the precision loss caused by scaling. Our RNS multiplier can get almost the same calculation precision as the TCS multiplier, which outperforms the basic scaling RNS multiplier about 2.6–3 times in SNR. In addition, the compensation items can be flexibly selected to make a trade-off between hardware resource and calculation precision. Synthesis results suggest that both our full compensatory RNS multiplier and partial compensatory RNS multiplier outperform traditional basic scaling RNS multiplier in hardware efficiency.

Author Contributions: The contributions made by the authors are as follows: Conceptualization, S.M. and S.H.; Methodology, S.M. and S.H.; Software, Z.Y., X.W., M.L., and J.H. Validation, X.W. and M.L. Formal analysis, Z.Y., X.W., M.L., and J.H. Investigation, S.M. and S.H. Data curation, S.M. and S.H. Writing—original draft preparation, S.M. and S.H. Writing—review and editing, S.M. and S.H. Visualization, S.M. and S.H. Supervision, S.M. and S.H. Project administration, S.M. and S.H. Funding acquisition, S.M. and S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China under Grant No. 61571083.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Schoinianakis, D. Residue arithmetic systems in cryptography: A survey on modern security applications. *J. Cryptogr. Eng.* **2020**, *10*, 249–267. [CrossRef]
2. Schinianakis, D.M.; Fournaris, A.P.; Kakarountas, A.P.; Stouraitis, T. An RNS Architecture of an Fp Elliptic Curve Point Multiplier. In Proceedings of the IEEE International Symposium on Circuits and Systems, Kos, Greece, 21–24 May 2006; Volume 56, pp. 1202–1213.
3. Bajard, J.C.; Eynard, J.; Merkiche, N.; Plantard, T. RNS Arithmetic Approach in Lattice-based Cryptography Accelerating the "Rounding-off" Core Procedure. In Proceedings of the 2015 IEEE 22nd Symposium on Computer Arithmetic, Lyon, France, 22–24 June 2015.
4. Jenkins, W.; Leon, B. The use of residue number systems in the design of finite impulse response digital filters. *IEEE Trans. Circuits Syst.* **1977**, *24*, 191–201. [CrossRef]
5. Re, A.D.; Nannarelli, A.; Re, M. Implementation of digital filters in carry-save residue number system. In Proceedings of the Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 4–7 November 2001; Volume 2, pp. 1309–1313.
6. Bernocchi, G.L.; Cardarilli, G.C.; Del Re, A.; Nannarelli, A.; Re, M. Low-power adaptive filter based on RNS components. In Proceedings of the IEEE International Symposium on Circuits and Systems, New Orleans, LA, USA, 27–30 May 2007; pp. 3211–3214.
7. Fernández, P.; García, A.; Ramírez, J.; Parrilla, L.; Lloris, A. A new implementation of the discrete cosine transform in the residue number system. In Proceedings of the Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 24–27 October 1999; Volume 2, pp. 1302–1306.
8. Arai, Y.; Agui, T.; Nakajima, M. A fast DCT-SQ scheme for images. *IEICE Trans.* **1988**, *71*, 1095–1097.
9. Tseng, B.D.; Jullien, G.A.; Miller, W.C. Implementation of FFT structures using the residue number system. *IEEE Trans. Comput.* **1979**, *100*, 831–845. [CrossRef]
10. Hiasat, A. Sign detector for the extended four-moduli set $\{2^n-1, 2^n+1, 2^{2n}+1, 2^{n+k}\}$. *IET Comput. Digit. Tech.* **2017**, *12*, 39–43. [CrossRef]

11. Xu, M.; Yao, R.; Luo, F. Low-Complexity Sign Detection Algorithm for RNS $\{2^n-1, 2^n, 2^n+1\}$. *IEICE Trans. Electron.* **2012**, *95*, 1552–1556. [CrossRef]
12. Sousa, L. Efficient method for magnitude comparison in RNS based on two pairs of conjugate moduli. In Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH'07), Montpellier, France, 25–27 June 2007; pp. 240–250.
13. Hiasat, A. A residue-to-binary converter with an adjustable structure for an extended RNS three-moduli set. *J. Circuits Syst. Comput.* **2019**, *28*, 1950126. [CrossRef]
14. Chang, C.H.; Low, J.Y.S. Simple, Fast, and Exact RNS Scaler for the Three-Moduli Set $\{2^n-1, 2^n, 2^n+1\}$. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2011**, *58*, 2686–2697. [CrossRef]
15. Hiasat, A. Efficient RNS scalers for the extended three-moduli set $\{2^n-1, 2^{n+p}, 2^n+1\}$. *IEEE Trans. Comput.* **2017**, *66*, 1253–1260. [CrossRef]
16. Taylor, F.J. An Overflow-Free Residue Multiplier. *IEEE Trans. Comput.* **1983**, *32*, 501–504. [CrossRef]
17. Chen, J.W.; Yao, R.H.; Wu, W.J. Efficient Modulo 2^n+1 Multipliers. *IEEE Trans. Very Large Scale Integr. Syst.* **2011**, *19*, 2149–2157. [CrossRef]
18. Muralidharan, R.; Chang, C.H. Radix-8 Booth Encoded Modulo 2^n-1 Multipliers with Adaptive Delay for High Dynamic Range Residue Number System. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2011**, *58*, 982–993. [CrossRef]
19. Zimmermann, R. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. In Proceedings of the IEEE Symposium on Computer Arithmetic, Adelaide, SA, Australia, 14–16 April 1999; p. 158.
20. Hiasat, A.A. New efficient structure for a modular multiplier for RNS. *IEEE Trans. Comput.* **2000**, *49*, 170–174. [CrossRef]
21. Omondi, A.R.; Premkumar, A.B. *Residue Number Systems: Theory and Implementation*; World Scientific: Singapore, 2007; Volume 2.
22. Low, J.Y.S.; Chang, C.H. A new RNS scaler for $\{2^n-1, 2^n, 2^n+1\}$. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Rio de Janeiro, Brazil, 15–18 May 2011; pp. 1431–1434.
23. Low, J.Y.S.; Tay, T.F.; Chang, C.H. A unified $\{2^n-1, 2^n, 2^n+1\}$ RNS scaler with dual scaling constants. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Kaohsiung, Taiwan, 2–5 December 2012.