Simple, Fast and Exact RNS Scaler for The Three-Moduli Set $\{2^n-1, 2^n, 2^n+1\}$

Chip-Hong Chang, *Senior Member, IEEE* and Jeremy Yung Shern Low

***Abstract*— Scaling in RNS has always been conceived as a performance bottleneck similar to the residue-to-binary conversion problem due to the inefficient inter-modulo operation. In this paper, a simple and fast scaling algorithm for the three-moduli set $\{2^n-1, 2^n, 2^n+1\}$ RNS is proposed. The complexity of inter-modulo operation has been resolved by a new formulation of scaling an integer in RNS domain by one of its moduli. By elegant exploitation of the Chinese Remainder Theorem and the number theoretic properties for this moduli set, the design can be readily implemented by a standard cell based design methodology. The low cost VLSI architecture without any read-only memory (ROM) makes it easier to fuse into and pipeline with other residue arithmetic operations of a RNS-based processor to increase the throughput rate. The proposed RNS scaler possesses zero scaling error and has a critical path delay of only $2\lceil log_2 n \rceil + 9$ units in unit-gate model. Besides the scaled residue numbers, the scaled integer in normal binary representation is also produced as a byproduct of this process, which saves the residue-to-binary converter when the binary representation of scaled integer is also required. Our experimental results show that the proposed RNS scaler is smaller and faster than the most area-efficient adder-based design and the fastest ROM-based design besides being the most power efficient among all scalers evaluated for the same three-moduli set.**

***Index Terms*— Chinese Remainder Theorem, full-adder-based architecture, inner product step processor, residue number system, scaling.**

I. Introduction

THE new generation of portable electronic devices, such as camcorders, digital cameras, 3G cell phone, etc. have incorporated many sophisticated real-time application-specific digital signal processing (DSP) functions. It has become increasingly difficult to keep up with the logical culmination of the demands for higher performance and lower power budget with the increased versatility of electronic products. Residue Number System (RNS) springs up as an alternative number system of special advantage for mapping complex DSP algorithms into application-specific hardware accelerators. This is due to the fact that long integer addition, subtraction and multiplication operations can be decomposed into smaller carry-free modulo operations for parallel processing. By introducing subword-level parallelism into an algorithm, RNS can tolerate a larger reduction in supply voltage than the corresponding weighted binary number system architecture for a given delay specification. The supply voltage reduction results in a quadratic power reduction and facilitates more efficient exploitation of multi-supply and multi-threshold voltage scaling techniques to further reduce the total static and dynamic power dissipation [1] – [2]. The usefulness of RNS has been demonstrated especially in the design of inner product step processor (IPSP) like architectures [3]-[7]. The most recent study [7] based on the two's complement and the mixed RNS-binary architecture implementations of an adaptive channel equalization filter with a large number of taps for wireless communications showed that the latter offer remarkable savings in area and power consumption without any degradation in performance. The reason for using a hybrid-RNS in [7] instead of a true RNS system is to avoid the use of complex and expensive scaling circuits.

Reverse conversion of an integer from its residue number representation [8]–[13], [43] and scaling of an integer number by a known constant [14]–[25] are two of the most important problems in RNS. Scaling is of particular significance for implementing IPSP and iterative digital processing systems dominated by a large number of additions and multiplications because it ensures the computed results of the preceding stages do not exceed the dynamic range of the system. Being inefficient in inter-modular operations, implementing scaling operation in RNS domain entices considerably long delay and high hardware complexity, as in the reverse conversion. Unlike the reverse conversion, which is required only when the final result is needed to be communicated with a normal binary number system, scaling operations are executed repeatedly in IPSP-like architectures to avoid the severe consequence of overflow in modulo arithmetic. In [7], adaptation is carried out in the binary domain to avoid the speed bottleneck of RNS scaling, but this is viable provided that the channel variations in time are slower with respect to the data rate.

To avoid overflow error in iterative computations, a RNS scaler should be placed after each RNS computation stage (each stage may involve one or more arithmetic operations), as shown in Fig. 1(a). RNS scaling problem is typically formulated based on modulo reduction of the least integer function of division or by Chinese Remainder Theorem (CRT). The former problem is generally solved by computationally intensive base extension methods [15], [17], [19], [23]-[25] and the outputs are free from

Manuscript received October 14, 2010; Revised December 26, 2010 and March 8, 2011. The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore (e-mail: {echchang, jere0017}@ntu.edu.sg).

scaling error. On the other hand, CRT-based approaches [14], [16], [20]-[22] usually produce scaled integer with some fractional error. The error could escalate and might not be tolerable for critical applications. All the aforementioned RNS scaling algorithms, except [19], [24] and [25], are implemented using ROMs. ROM-based approaches have low throughput. Although ROM matrices can be replaced with modulo multipliers, the cost is unacceptably high, especially for large moduli. In the absence of an efficient RNS scaler, a binary scaler may be used in a hybrid RNS but it must be preceded by an expensive reverse converter, as shown in Fig. 1(b). It is therefore imperative to have a high-speed scaler that operates directly in the RNS domain with lower complexity than a RNS-to-binary converter. Given that scaling in RNS is an inherently inter-modular operation whereby knowledge of all of the residue digits is essential to produce the correct final result [15], [20], such a problem is better tackled with a moduli set that possesses good number theoretic properties.

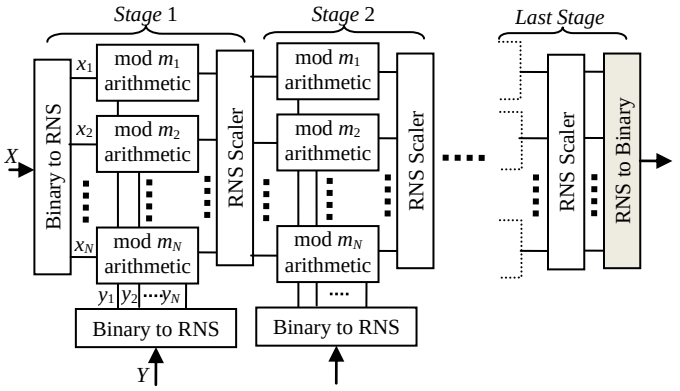


Figure 1: (a) IPSP with true RNS scalers (b) IPSP with binary scalers in hybrid RNS

This paper presents a new and simple adder-based RNS scaler for the three-moduli set $\{2^n-1, 2^n, 2^n+1\}$. The choice of this RNS for overcoming the scaling problem is well justified by the relative simplicity and efficiency in implementing modulo addition, multiplication and shifter circuits for these three moduli proven by the humongous amount of research publications [26]–[34]. The conversion from the normal binary representation to residues for moduli of types 2^n and $2^n \pm 1$ is straightforward and the reverse conversion problem for this special three-moduli set has been well solved [8], [9]. Most importantly, the inputs to our proposed RNS scaler are in residue form, and the scaled output can be produced efficiently in both normal binary and residue forms. This has an additional advantage of eliminating the reverse converter in the last stage of the architecture shown in Fig. 1(a).

The paper is organized as follows: recent works on RNS scaling operation are reviewed in Section II, followed by the derivation of the new RNS scaling algorithm and its error analysis in Section III. The architecture of the proposed scaler is presented in Section IV. Section V analyzes the area and time complexity of its implementation, and compares the synthesis results with other scaler designs. The paper is concluded in Section VI.

II. Related Work

The earliest research on RNS scaling problem dated back to the sixties whence scaling an integer by a product of any subset of the moduli in RNS can be performed in n clock cycles [10]. Since then, many new techniques have been proposed in the literature. This section reviews some recent developments on RNS scaling algorithms that have an impact on hardware implementation.

Shenoy and Kumaresan [14] proposed a scaling technique based on the CRT. In their work, a redundant residue digit is introduced to enable the parallel computation of the scaled residues, and a novel decomposition of CRT is applied to reduce the maximum scaling error to unity. In [21], Griffin *et al.* first used the least integer function $\lfloor \cdot \rfloor$ in CRT to produce the scaled output with error bounded by the number of modulus L used in the RNS (hence, it is known as L-CRT). Later, a generalization of L-CRT, called $L(\varepsilon+\delta)$-CRT [20] is proposed to provide the flexibility in choosing the scaling factor but the new reduced system modulus comes at a cost of potentially large errors. The catastrophic error band may be avoided with carefully selected reduced modulus for a pre-specified scaling factor. Another scaling algorithm that utilizes CRT was put forward by Ulman *et al.* [22]. The latter scaling method could be considered as an improved version of the design in [10], as it eliminates the use of redundant modulus without imposing any restrictions on the system moduli and on the scaling factor. Besides, the scaling error of this algorithm is not greater than 1.5. The aforementioned scaling algorithms use CRT in such a way that the residue representation of an integer in RNS is partially converted to its binary representation before producing the scaled output. Furthermore, they are ROM-based design which cannot be easily pipelined for high-throughput applications.

More recent works on scaling an integer, X by a factor k for the moduli set $\{m_1, m_2, \cdots, m_N\}$ is based on the scaling operation defined in [17]. It states that if k is the scaling constant and Y is the result of scaling X by k, then $X = kY + |X|_k$, where $|X|_k$ denotes the remainder of X divided by k. Thus,

$$|Y|_{m_i} = \left\| \frac{X}{k} \right\|_{m_i} = \left| \frac{X - |X|_k}{k} \right|_{m_i} \quad (1)$$

Barsi and Pinotti [15] derived a scaling algorithm by applying the base extension to (1) without approximation. The latter work has once again manipulated the CRT so that their design can perform both base extension and exact scaling without any system redundancy. Using almost similar scaling algorithm, Garcia and Lloris [17] had proposed a different look-up scheme, which leads to a faster and simpler implementation of the RNS scaling operation than [15]. The drawback is its restriction on the number of moduli in the system – significantly large amount of memory is required for a large number of moduli. Similarly, these two designs also use memory for implementation. All these memory-based designs suffer from poor pipelinability and dramatic increase in hardware cost with the number of system moduli. Recently, Meyer-Base and Stouraitis [18] had come out with a scaling algorithm for small word length applications. It does not rely on a conversion back to a weighted number system. Instead, it extended the *Division Remainder Zero Theorem* of Szabo and Tanaka [10] to perform scaling directly on the residue digits.

The first and only full adder (FA) based exact scaling algorithm that operates directly on the residue digits was proposed by Dasygenis *et al.* [19]. It uses the axiom, $\left|X - |X|_k\right|_{m_i} = \left|x_i - \left\||X|_k\right|_{m_i}\right|_{m_i}$ to compute the scaled output, $y_i = \left|\left|x_i - \left\||X|_k\right|_{m_i}\right|_{m_i} \cdot \left|k^{-1}\right|_{m_i}\right|_{m_i}$ over each modulus independently, where $x_i = |X|_{m_i}$ is a residue of the RNS defined by the moduli set $\{m_1, m_2, \cdots, m_N\}$. The computation is divided into several major stages as shown in Fig. 2. Each stage can be implemented with only FAs, which makes the hardware cost remarkably lower and throughput rate significantly higher than all the previous designs. Unfortunately, this algorithm has ignored the fact that $\left\|X|_k\right|_{m_i}$ is an inter-modular operation and it cannot be simply resolved by treating X as an overflowed digit or a residue over each modulus independently. The authors of [19] also made a catastrophic assumption that the scaling constant, k can be chosen to be a product of various moduli, i.e., $k = \prod_{i=1}^{S} m_i$ with $S < N$, but it has been proven that multiplicative inverse of k for $m_i\ (1 \le i \le S)$ does not exist [15], [17]. Consequently, the output, $(y_1, y_2, \cdots, y_N)$ calculated by the algorithm of [19] is not a valid RNS representation of scaling X by k. Using the first example from [19], this error is illustrated by the numerical calculation in Table I. The actual residue digits of the scaled value (254, 204, 154) are different from those calculated (4, 3, 3) using the scaling method in [19]. In fact, the condition $(k, m_i) = 1$ for the existence of $\left|k^{-1}\right|_{m_i}$ is not met because of $(k, m_1) = (255, 255) = 255$. Besides, it may not be possible to access to the already overflowed digits because the modulo operators in RNS are usually designed such that modulo reductions are performed to keep the final results and even their intermediate computations within the legitimate ranges of the respective moduli.

In view of the above, the only valid adder-based RNS scaler design technique that utilizes (1) was proposed in [24] and [25]. The scaler of [24] is dedicated to the three moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ with scaling constant 2^n while [25] has extended it to general moduli sets with the proviso that the moduli are coprime with the scaling constant. The generalization has restricted its hardware simplification, leading to increased hardware cost and reduced performance over those of [24].

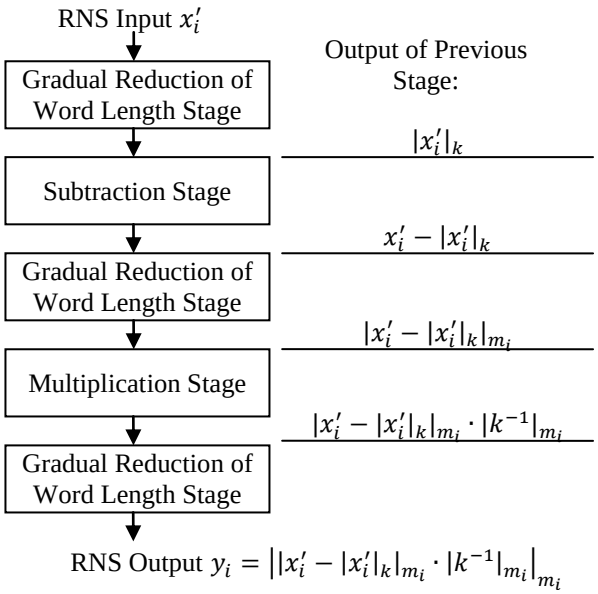


Figure 2: Block diagram of RNS scaler design in [19].

TABLE I: COMPUTATION OF ACTUAL RESIDUE DIGITS OF SCALED NUMBER

Modulus, m_i	255	256	257
Dynamic range, M	16776960		
M_i	65792	65535	65280
M_i^{-1}	128	255	129
$M_i \cdot M_i^{-1}$	8421376	16711425	8421120
overflowed $x_i = x_i'$	1100	900	800
$x_i = \|x_i'\|_{m_i}$	80	132	29
$x_i \cdot M_i \cdot M_i^{-1}$	673710080	2205908100	244212480
Integer X	3316100		
Scaler k	255		
$Y = \lfloor X/k \rfloor$	13004		
y_i	*254*	*204*	*154*

III. PROPOSED SCALING ALGORITHM

A. Preliminaries

In RNS, an integer X is represented by an N-tuple $(x_1, x_2, \cdots, x_N)$ with respect to a set of pairwise relatively prime numbers $\{m_1, m_2, \cdots, m_N\}$, where $x_i = |X|_{m_i}$, i = 1, 2, …, N and $|X|_{m_i}$ is defined as X *mod* m_i. The dynamic range of a selected

moduli set $\{m_1, m_2, \cdots, m_N\}$ is given by

$$M = \prod_{i=1}^{N} m_i \tag{2}$$

Based on CRT [14], X is related to its residue digits by

$$X = \left| \sum_{i=1}^{N} M_i \left| M_i^{-1} \right|_{m_i} x_i \right|_M \tag{3}$$

where $M_i = \dfrac{M}{m_i}$ and $\left|M_i^{-1}\right|_m$ is the multiplicative inverse of $\left|M_i^{-1}\right|_m$.

Our proposed scaling algorithm is designed for the three-moduli set $\{2^n-1, 2^n, 2^n+1\}$, so $N = 3$ and (3) becomes

$$X = \left| m_2 m_3 \left| M_1^{-1} \right|_{m_1} x_1 + m_1 m_3 \left| M_2^{-1} \right|_{m_2} x_2 + m_1 m_2 \left| M_3^{-1} \right|_{m_3} x_3 \right|_M \tag{4}$$

The following axioms and theorems are used for the derivation of our scaling algorithm.

Axiom 1: $A|X|_B = |AX|_{AB}$

Axiom 2: $|X \pm Y|_m = \left| |X|_m \pm |Y|_m \right|_m$

Axiom 3: $|X \cdot Y|_m = \left| |X|_m \cdot |Y|_m \right|_m$

Theorem 1: Given $p = kq$, where k is an integer, $\left| |X|_p \right|_q = |X|_q$.

Proof: Based on the definition of modulo operation [14],

$$|X|_p = X - \varepsilon p, \ \varepsilon = 0, 1, 2, \cdots \tag{5}$$

From *Axioms 2* and *3*,

$$\left| |X|_p \right|_q = |X - \varepsilon p|_q = \left| |X|_q - |\varepsilon|_q \cdot |p|_q \right|_q \tag{6}$$

Since p is divisible by q, $|p|_q = 0$,

$$\left| |X|_p \right|_q = |X|_q \tag{7}$$

Theorem 2[42]: Given the moduli set of $\{m_1, m_2, m_3\}$ with $m_1 = 2^n - 1$, $m_2 = 2^n$ and $m_3 = 2^n + 1$, the followings hold true.

$$\left|M_1^{-1}\right|_{m_1} = 2^{n-1} \tag{8}$$

$$\left|M_2^{-1}\right|_{m_2} = 2^n - 1 \tag{9}$$

$$\left|M_3^{-1}\right|_{m_3} = 2^{n-1} + 1 \tag{10}$$

B. New Formulation of RNS Scaling

Based on the above preliminaries, the following theorem is proposed to reduce the complexity of inter-modular operation of RNS scaling.

Theorem 3: Let $x_i = |X|_{m_i}$ for $i = 1, 2, 3$ and $m_1 = 2^n - 1$, $m_2 = 2^n$ and $m_3 = 2^n + 1$. If $k = m_2 = 2^n$, then

$$\left\| \frac{X}{k} \right\|_{m_1} = |x_1 - x_2|_{m_1} \tag{11}$$

$$\left\| \frac{X}{k} \right\|_{m_2} = \left| \left(2^{2n-1} + 2^{n-1}\right) x_1 - 2^n x_2 + \left(2^{2n-1} + 2^{n-1} - 1\right) x_3 \right|_{m_1 m_3} \Big|_{m_2} \tag{12}$$

$$\left\| \frac{X}{k} \right\|_{m_3} = \left| x_2 + 2^n x_3 \right|_{m_3} \tag{13}$$

The formulation of this theorem and its proof is given as follows.

By definition, scaling an integer variable X by a constant integer k can be obtained by dividing both sides of (4) by k and then taking its floor value. Let Y be the integer result of the scaling operation, using *Axiom 1*, we have

$$\begin{aligned} Y &= \left\lfloor \frac{X}{k} \right\rfloor \\ &= \left\lfloor \frac{1}{k} \left| m_2 m_3 \left| M_1^{-1} \right|_{m_1} x_1 + m_1 m_3 \left| M_2^{-1} \right|_{m_2} x_2 + m_1 m_2 \left| M_3^{-1} \right|_{m_3} x_3 \right|_M \right\rfloor \\ &= \left\lfloor \left| \frac{m_2 m_3}{k} \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{k} \left| M_2^{-1} \right|_{m_2} x_2 + \frac{m_1 m_2}{k} \left| M_3^{-1} \right|_{m_3} x_3 \right|_{\frac{M}{k}} \right\rfloor \end{aligned} \tag{14}$$

where $\lfloor \bullet \rfloor$ is the least integer function.

k is chosen to be equal to $m_2 = 2^n$ (The choice of k is crucial in ensuring that the truncation error is negligible and this will be discussed shortly), which produces

$$\left\lfloor \frac{X}{k} \right\rfloor = \left\lfloor \left| m_3 \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{m_1 m_3} \right\rfloor \tag{15}$$

Using (15), the scaled integer can be computed directly from the RNS representation of X.

It is desired to have the output of the RNS scaler to be generated directly in the residue form so that it can be directly processed by the arithmetic unit in each residue channel. The residue digit in each channel can be computed by taking the corresponding modulo operation on both sides of (15) as follows:

$$\left\| \frac{X}{k} \right\|_{m_i} = \left| \left| m_3 \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{m_1 m_3} \right|_{m_i} \tag{16}$$

where $i = 1$, 2 and 3.

For m_1 and m_3 channels, according to *Theorem 1*, (16) can be simplified to

$$\left\| \frac{X}{k} \right\|_{m_i} = \left| m_3 \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{m_i} \tag{17}$$

where $i = 1, 3$.

Each independently scaled residue of (17) can be further reduced to

$$\left\| \frac{X}{k} \right\|_{m_1} = \left| m_3 \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 \right|_{m_1} \tag{18}$$

and

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_3} = \left| \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{m_3} \quad (19)$$

For modulus m_2,

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_2} = \left| \left| m_3 \left| M_1^{-1} \right|_{m_1} x_1 + \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} x_2 + m_1 \left| M_3^{-1} \right|_{m_3} x_3 \right|_{m_1 m_3} \right|_{m_2} \quad (20)$$

As modulo 2^n of a variable is the same as keeping only the n least significant bits of it, the scaled residue digit, y_2 for the modulus m_2 can be obtained by simply hardwiring the n least significant bits of the result computed from the right hand side of (20). There is no additional logic and delay cost incurred by direct hardwiring.

It is observed that $\frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2}$ is a common term in (18), (19) and (20). Substituting the expressions of m_1, m_2, m_3 and $\left| M_2^{-1} \right|_{m_2}$ for this common term, we have

$$\begin{aligned} \left\lfloor \frac{m_1 m_3}{m_2} \left| M_2^{-1} \right|_{m_2} \right\rfloor &= \left\lfloor \frac{(2^n - 1)(2^n + 1)(2^n - 1)}{2^n} \right\rfloor \\ &= \left\lfloor \frac{2^{3n} - 2^{2n+1} + 2^{2n} + 2^n - 2^{n+1} + 1}{2^n} \right\rfloor \\ &\cong 2^{2n} - 2^{n+1} + 2^n - 1 \end{aligned} \quad (21)$$

The least integer function $\lfloor \cdot \rfloor$ of the scaling operation in RNS justifies the truncation of (21). The effect of this approximation will be analyzed in Section C.

By substituting (21) into (18), (19) and (20), we have

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_1} = \left| \left(2^{2n-1} + 2^{n-1}\right) x_1 + \left(2^{2n} - 2^{n+1} + 2^n - 1\right) x_2 \right|_{m_1} \quad (22)$$

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_2} = \left\| \left(2^{2n-1} + 2^{n-1}\right) x_1 + \left(2^{2n} - 2^{n+1} + 2^n - 1\right) x_2 + \left(2^{2n-1} + 2^n - 2^{n-1} - 1\right) x_3 \right|_{m_1 m_3} \Big|_{m_2} \quad (23)$$

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_3} = \left| \left(2^{2n} - 2^{n+1} + 2^n - 1\right) x_2 + \left(2^{2n-1} + 2^n - 2^{n-1} - 1\right) x_3 \right|_{m_3} \quad (24)$$

Since $\left| 2^n \right|_{2^n - 1} = \left| 1 \right|_{2^n - 1}$, as a corollary of *Axioms 2* and *3*, the following periodic property for modulus m_1 can be defined [35], [36].

Corollary 1: For any non-negative integer, a

$$\left| 2^{an+b} \right|_{m_1} = \left| 2^b \right|_{m_1} \quad (25)$$

By applying *Axioms 2* and *3* and *Corollary 1*, and using $m_1 m_3 = (2^n - 1)(2^n + 1) = 2^{2n} - 1$, (22), (23) and (24) can be reduced eventually to the following highly simplified expressions in *Theorem 3*.

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_1} = \left| x_1 - x_2 \right|_{m_1}$$

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_2} = \left\| \left(2^{2n-1} + 2^{n-1}\right) x_1 - 2^n x_2 + \left(2^{2n-1} + 2^{n-1} - 1\right) x_3 \right|_{m_1 m_3} \Big|_{m_2}$$

$$\left\| \left\lfloor \frac{X}{k} \right\rfloor \right\|_{m_3} = \left| x_2 + 2^n x_3 \right|_{m_3}$$

Example 1: Consider the same integer X = 3316100 from Table 1 with the RNS representation of (80, 132, 29) for the moduli set {255, 256, 257}. Using (11), (12) and (13), the scaled output in RNS representation is calculated in Table II. It can be verified that the RNS number (203, 153, 103) is equivalent to the integer, 12953 computed directly from $\lfloor 3316100 / 256 \rfloor$. Note that the binary representation of 12953 = 0011001010011001_2 is a byproduct in the computation of the scaled residue y_2 before the final modulo of (12) is taken. Its eight least significant bits are 10011001_2 = 153.

TABLE II: COMPUTATION TRACES OF SCALING (80, 132, 29) BY K =256 IN RNS {255, 256, 257}

$\left\| \lfloor X/k \rfloor \right\|_{m_1}$	$\left\| 80 - 132 \right\|_{255} = 203$
$\left\| \lfloor X/k \rfloor \right\|_{m_2}$	$\left\| 32896 \times 80 - 256 \times 132 + 32895 \times 29 \right\|_{65535} \Big\|_{256}$ $= \left\| 12953 \right\|_{256} = 153$
$\left\| \lfloor X/k \rfloor \right\|_{m_3}$	$\left\| 132 + 256 \times 29 \right\|_{257} = 103$

C. Error Analysis

The second term of the exact scaling equation of (15) is truncated to produce the approximation of

$$\left\lfloor \frac{X}{k} \right\rfloor' = \left\lfloor \left| \begin{aligned} &\left(2^{2n-1} + 2^{n-1}\right) x_1 + \left(2^{2n} - 2^{n+1} + 2^n - 1\right) x_2 \\ &+ \left(2^{2n-1} + 2^n - 2^{n-1} - 1\right) x_3 \end{aligned} \right|_{m_1 m_3} \right\rfloor \quad (26)$$

Let $k_1 = 2^{2n-1} + 2^{n-1}$, $k_2 = 2^{2n} - 2^{n+1} + 2^n - 1$, $k_3 = 2^{2n-1} + 2^n - 2^{n-1} - 1$, and $p = m_1 m_3$, (26) can be rewritten as

$$\left\lfloor \frac{X}{k} \right\rfloor' = \left\lfloor \left| k_1 x_1 + k_2 x_2 + k_3 x_3 \right|_p \right\rfloor \quad (27)$$

The integer resulting from the above modulo operation can be expressed using the definition of (5).

$$\left\lfloor \frac{X}{k} \right\rfloor' = \left\lfloor k_1 x_1 + k_2 x_2 + k_3 x_3 - \varepsilon p \right\rfloor \quad (28)$$

where ε is a non-negative integer.

Without truncation, the exact equation, (15), is given by

$$\left\lfloor \frac{X}{k} \right\rfloor = \left\lfloor \begin{aligned} &\left| \left(2^{2n-1}+2^{n-1}\right)x_1 + \left(2^{2n}-2^{n+1}+2^n-1+\frac{1}{2^n}\right)x_2 \right| \\ &+\left(2^{2n-1}+2^n-2^{n-1}-1\right)x_3\Big|_{m_1 m_3} \end{aligned} \right\rfloor$$
$$= \left\lfloor \left| k_1x_1 + \left(k_2+\frac{1}{2^n}\right)x_2 + k_3x_3 \right|_p \right\rfloor$$
$$= \left\lfloor \left| k_1x_1 + k_2x_2 + k_3x_3 + \frac{x_2}{2^n} \right|_p \right\rfloor \quad (29)$$
$$= \left\lfloor \left| k_1x_1 + k_2x_2 + k_3x_3 + c \right|_p \right\rfloor$$
$$= \left\lfloor k_1x_1 + k_2x_2 + k_3x_3 + c - \varepsilon p \right\rfloor$$

where $c = \frac{x_2}{2^n}$.

Since $\left|k_1x_1 + k_1x_2 + k_3x_3\right|_p = k_1x_1 + k_2x_2 + k_3x_3 - \varepsilon p$ is an integer, based on the definition of the least integer function, (28) and (29) can be rewritten as:

$$\left\lfloor \frac{X}{k} \right\rfloor' = k_1x_1 + k_2x_2 + k_3x_3 - \varepsilon p \quad (30)$$

$$\left\lfloor \frac{X}{k} \right\rfloor = k_1x_1 + k_2x_2 + k_3x_3 - \varepsilon p + \lfloor c \rfloor \quad (31)$$

Since $0 \le x_2 < 2^n$, $0 \le c < 1$ and $\lfloor c \rfloor = 0$. Therefore,

$$\left\lfloor \frac{X}{k} \right\rfloor = k_1x_1 + k_2x_2 + k_3x_3 - \varepsilon p = \left\lfloor \frac{X}{k} \right\rfloor' \quad (32)$$

This proves that the proposed RNS scaling algorithm is exact and does not introduce any scaling error.

IV. Hardware Implementation

From (11), since $\left|-x_2\right|_{2^n-1} = \left|2^n-1-x_2\right|_{2^n-1} = \bar{x}_2$, the generation of the scaled RNS residue, y_1 requires a two-operand modulo 2^n-1 adder, which can be implemented using a Ling parallel-prefix modulo 2^n-1 adder [37].

The generation of the scaled RNS residue, y_2 by (12) needs some attention. The following two extensively used special properties of modulo 2^n-1 arithmetic, proposed by Szabo and Tanaka [10] are exploited here to simplify the implementation.

Property 1: Multiplying an *n*-bit binary number *x* by *r* power of 2 in modulo 2^n-1 is equivalent to a circular left shift operation,

$$\left|2^r x\right|_{2^n-1} = CLS_n(x,r), \quad (33)$$

where $CLS_n(x, r)$ denotes a circular shift of the *n*-bit binary number *x* by *r* bits to the left.

Property 2:

$$\left|-2^r x\right|_{2^n-1} = \left|2^r\left(2^n-1-x\right)\right|_{2^n-1} = \left|2^r\bar{x}\right|_{2^n-1} = CLS_n(\bar{x},r) \quad (34)$$

where $\bar{x}$ is the 1's complement of the *n*-bit binary number *x*.

Let the *j*-th bit of a residue x_i be represented as $x_{i,j}$. The three residues to be scaled can then be represented in their binary forms as follows: $x_1 = \left(x_{1,n-1}x_{1,n-2}\cdots x_{1,0}\right)_2$, $x_2 = \left(x_{2,n-1}x_{2,n-2}\cdots x_{2,0}\right)_2$ and $x_3 = \left(x_{3,n}x_{3,n-1}\cdots x_{3,0}\right)_2$.

By applying *Properties 1* and *2* to each term of (12), we have

$$P_1 = \left|2^{2n-1}x_1\right|_{2^{2n}-1} = CLS_{2n}\left(x_1, 2n-1\right) = \left(x_{1,0}\underbrace{00\cdots0}_{n}x_{1,n-1}x_{1,n-2}\cdots x_{1,1}\right)_2 \quad (35)$$

$$P_2 = \left|2^{n-1}x_1\right|_{2^{2n}-1} = CLS_{2n}\left(x_1, n-1\right) = \left(0x_{1,n-1}x_{1,n-2}\cdots x_{1,0}\underbrace{00\cdots0}_{n-1}\right)_2 \quad (36)$$

$$P_3 = \left|-2^n x_2\right|_{2^{2n}-1} = CLS_{2n}\left(\overline{x_2}, n\right) = \left(\bar{x}_{2,n-1}\bar{x}_{2,n-2}\cdots\bar{x}_{2,0}\underbrace{11\cdots1}_{n}\right)_2 \quad (37)$$

$$P_4 = \left|2^{2n-1}x_3\right|_{2^{2n}-1} = CLS_{2n}\left(x_3, 2n-1\right) = \left(x_{3,0}\underbrace{00\cdots0}_{n-1}x_{3,n}x_{3,n-1}\cdots x_{3,1}\right)_2 \quad (38)$$

$$P_5 = \left|2^{n-1}x_3\right|_{2^{2n}-1} = CLS_{2n}\left(x_3, n-1\right) = \left(x_{3,n}x_{3,n-1}\cdots x_{3,0}\underbrace{00\cdots0}_{n-1}\right)_2 \quad (39)$$

$$P_6 = \left|-x_3\right|_{2^{2n}-1} = \left(\underbrace{11\cdots1}_{n-1}\bar{x}_{3,n}\bar{x}_{3,n-1}\cdots\bar{x}_{3,0}\right)_2 \quad (40)$$

The sum of the binary vectors $P_1 + P_2$ can be expressed as a single term $Q_1 = \left(x_{1,0}x_{1,n-1}x_{1,n-2}\cdots x_{1,0}x_{1,n-1}x_{1,n-2}\cdots x_{1,1}\right)_2$. The remaining four addends can be reduced to three binary vectors by reordering the bits of different addends at the same position followed by logic minimization. As shown in Fig. 3(a), the 1-string in P_3 is first swapped with $\bar{x}_{3,n}\bar{x}_{3,n-1}\cdots\bar{x}_{3,0}$ in P_6. Then, the 0-string in P_5 is swapped with the corresponding 1-string that has just been transferred to P_6. After that, $x_{3,n}$ in P_4 is swapped with $x_{3,0}$ in P_5. Finally, the 0-string in P_4, $x_{3,n-1}x_{3,n-2}\cdots x_{3,1}$ in P_5 and the 1-string in P_6 are interchanged. The strings to be swapped are boxed and annotated with its order of execution in Fig. 3(a). The vectors after bit reordering are shown in Fig. 3(b). The last two vectors (in the dotted box) of Fig. 3(b) can be merged into a single constant vector of $\left(11\cdots1\right)_2$ when $x_{3,n} = 0$ and $(1011\cdots\underset{n-1}{1}1011\cdots\underset{n-1}{1})_2$ when $x_{3,n} = 1$. Since $\left|(1011\cdots\underset{n-1}{1}1011\cdots\underset{n-1}{1})_2\right|_{2^{2n}-1} = (01\cdots\underset{n}{1}10\cdots\underset{n-1}{0})_2$, the merged term is represented by $Q_4 = (\bar{x}_{3,n}\underset{n}{1\cdots1}\underbrace{\bar{x}_{3,n}\cdots\bar{x}_{3,n}}_{n-1})_2$ in Fig. 3(c).

Since $x_3 \le 2^n$, $x_{3,n} = 1$ if $x_3 = 2^n$. Also, when $x_3 = 2^n$, $x_{3,i} = 0$ $\forall i \le n-1$, the *n* lower order bits of Q_2 becomes '1's, Q_3

becomes an all-zero string and $Q_4 = "(01\cdots110\cdots00)_2$. The three nonzero binary vectors, Q_1, Q_2 and Q_4, can be summed using a 2*n*-bit carry save adder (CSA) with end-around carry (EAC). On the other hand, when $x_{3,n} = 0$, Q_4 is an all-one string. If Q_1, Q_2 and Q_3 are summed using a 2*n*-bit CSA with EAC, the EAC from the CSA will propagate all the way to the carry out, and wrap around to the LSB. Hence, no additional 2*n*-bit CSA is needed to sum the all-one vector, Q_4. The same 2*n*-bit CSA can therefore be reused by replacing Q_3 by Q_4 when $x_{3,n} = 1$. The *n* multiplexed addend bits to the CSA can be implemented by $x_{3,n} + x_{3,i}$, where i = 0, 1, ..., n–1 and '+' denotes logical *OR* operation. Consequently, the four binary vectors, Q_1 to Q_4 can be added using only one 2*n*-bit modulo $2^{2n}-1$ CSA with EAC and *n* two-input *OR* gates, followed by a parallel-prefix modulo $2^{2n}-1$ adder. The four input vectors are formed by routing the binary bits from the input residues without using any hardware logic. It is important to note that the maximum delay of the CSA tree is kept constant at one FA delay regardless of the value of *n*. The residue y_2 of the scaled integer output, *Y* for the modulus m_2 can be taken directly from the least significant *n* bits of the $2^{2n}-1$ adder.

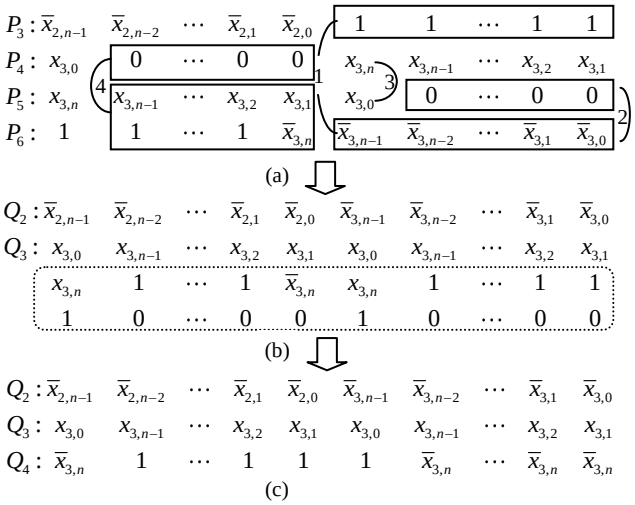


Figure 3: Reduction of addends.

To produce y_3 from (13), the following two axioms [32] are needed.

Axiom 4: $\left|2^{n+i}a\right|_{2^n+1} = \left|2^i\bar{a} + 2^{n+i}\right|_{2^n+1}$

Axiom 5: $\left|2^{2n}a\right|_{2^n+1} = \left|a\right|_{2^n+1}$

where $a \in \{0,1\}$ and i = 0, 1, .., n–1.

By applying *Axioms 4* and *5* to the second term of (13), the variable bits of this term are wrapped to the *n* least significant bit positions and its expression becomes:

$$\left|2^n x_3\right|_{2^n+1} = \left|\sum_{i=0}^{n} 2^{n+i} x_{3,i}\right|_{2^n+1} = \left|x_{3,n} + \sum_{i=0}^{n-1} 2^i \overline{x_{3,i}} + \sum_{i=n}^{2n-1} 2^i\right|_{2^n+1} \quad (41)$$

The addition of x_2 to (41) can be reduced by a CSA tree before a diminished-one mod 2^n+1 adder [28] is used to produce the final output. A one needs to be subtracted from each of the sum and carry outputs of the CSA tree to convert them into diminished-one representation before the diminished-one adder, and the output of the diminished-one adder needs to be incremented by one to obtain the actual output. This aggregate offset of –1 can be added upfront in the CSA tree. According to *Axiom 4*, the carry generated out of the most significant bit of each *n*-bit CSA will be complemented and added to the least significant bit of the next level of CSA together with a constant addition of 2^n. By assuming a three-level CSA tree, an offset of 3×2^n will be generated. Together with the compensation for diminished one addition and the last term of (41), the total correction constant, *C* to be added in the CSA tree is given by:

$$\begin{aligned} C &= \left|\sum_{n}^{2n-1} 2^i - 1 + 3\times2^n\right|_{2^n+1} = \left|2^{2n} - 2^n - 1 + 3\times2^n\right|_{2^n+1} \\ &= \left|1-(-1)-1+3\times(-1)\right|_{2^n-1} = \left|-2\right|_{2^n+1} \\ &= \left|2^n+1-2\right|_{2^n+1} = \left|2^n-1\right|_{2^n+1} \end{aligned} \quad (42)$$

The addends to be summed in the CSA tree are shown in Fig. 4(a). The first *n*-bit CSA adds only the last two addends to produce the results of Fig. 4(b), with the complementary end-around carry (CEAC) printed bold. Since $1+a = 2a + \bar{a}$ for any binary variable *a*, no new signal variable is introduced. The result can be obtained by direct hardwiring without the need for the CSA. The second *n*-bit CSA adds the last three addends of Fig. 4(b) to produce the results of Fig. 4(c), with the CEAC printed bold. Based on the property of $1+a = 2a + \bar{a}$, only two modified full adders (MFAs), instead of a complete *n*-bit CSA, are required to produce the unknown variables, s_0, c_0, s_1 and c_1. Finally, an *n*-bit CSA is required to reduce Fig. 4(c) to two addends to be fed into the final diminished-one mod 2^n+1 adder. Although we started with a three-level CSA addition, most intermediate results can be directly hardwired and only one level of *n*-bit CSA with CEAC and two MFAs are needed eventually, which are highlighted by the shaded boxes of Fig. 4.

$x_{2,n-1}$	$x_{2,n-2}$	...	$x_{2,2}$	$x_{2,1}$	$x_{2,0}$
$\overline{x_{3,n-1}}$	$\overline{x_{3,n-2}}$	...	$\overline{x_{3,2}}$	$\overline{x_{3,1}}$	$\overline{x_{3,0}}$
1	1	...	1	1	1
0	0	...	0	0	$x_{3,n}$

(a) Level 1 CSA with CEAC

$x_{2,n-1}$	$x_{2,n-2}$	...	$x_{2,2}$	$x_{2,1}$	$x_{2,0}$
$\overline{x_{3,n-1}}$	$\overline{x_{3,n-2}}$	...	$\overline{x_{3,2}}$	$\overline{x_{3,1}}$	$\overline{x_{3,0}}$
1	1	...	1	1	$\overline{x_{3,n}}$
0	0	...	0	$x_{3,n}$	**1**

(b) Level 2 CSA with CEAC

$x_{2,n-1}$	$x_{2,n-2}$	...	$x_{2,2}$	$x_{2,1}$	$x_{2,0}$
$x_{3,n-1}$	$x_{3,n-2}$	...	$x_{3,2}$	s_1	s_0
$\overline{x_{3,n-2}}$	$\overline{x_{3,n-3}}$	...	c_1	c_0	$\boldsymbol{x_{3,n-1}}$

(c) Level 3 CSA with CEAC

Figure 4: Simplification of the CSA tree for the computation of y_3

The complete RNS scaling architecture is shown in Fig. 5. Since the inner modulo $2^{2n}-1$ operation in (12) has to be computed any way to obtain the scaled output, y_2, the scaled integer, Y is also available as a byproduct of this computation. Thus, without introducing any hardware overhead, the scaled output is available in both RNS and binary forms. This flexibility to manipulate the scaled output in two domains has advantage in interfacing the scaled output to computational units in either domain without the need for the expensive RNS-to-binary converter. For example, the bottleneck reverse converter in the last stage of Fig. 1(a) can be eliminated to increase the throughput of the RNS IPSP.

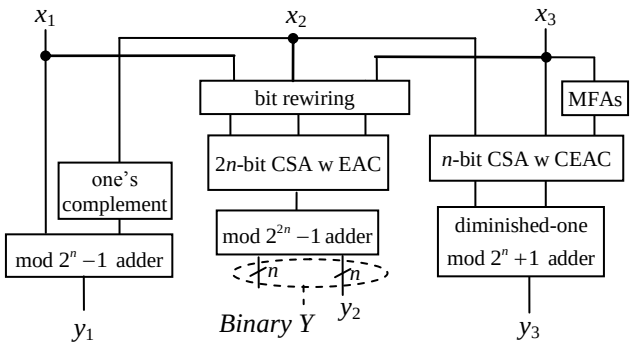


Figure 5: Architecture of the proposed RNS scaler.

Example 2: Consider the RNS representation of X = (80, 132, 29) for the moduli set {255, 256, 257} from Example 1. x_1 = 01010000_2, x_2 = 10000100_2 and x_3 = 000011101_2. Q_1 = 00101000 00101000_2, Q_2 = 01111011 11100010_2, Q_3 = 10001110 10001110_2 and Q_4 = 11111111 11111111_2. The modulo 2^n-1 addition of x_1 and $\overline{x}_2$, carry save addition with end-around carry of the four $2n$-bit binary vectors, Q_1 to Q_3, and diminished-one modulo 2^n+1 addition of x_2 and $\overline{x}_3$ are illustrated in Fig. 6. From Fig. 6, Y = 0011001010011001_2 = 12953, y_1 = 11001011_2 = 203, y_2 = 10011001_2 = 153 and y_3 = 01100111_2 = 103 tally with the correct outputs from Table II.

In the final part of this section, we show that the final modulo $2^{2n}-1$ adder in m_2 channel can be replaced by a much simpler modulo 2^n adder if only the residue output y_2 is needed.

From [37],

$$\left|A+B\right|_{2^{2n}-1}=\begin{cases}\left|A+B\right|_{2^{2n}}+1 & \text{if } A+B\geq 2^{2n}\\ \left|A+B\right|_{2^{2n}} & \text{Otherwise}\end{cases} \tag{43}$$

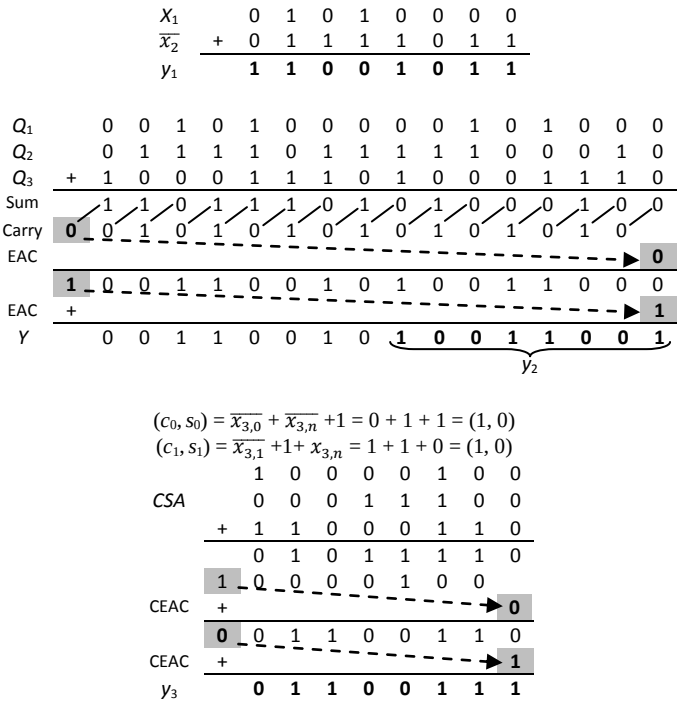
It follows that

$$\left|\left|A+B\right|_{2^{2n}-1}\right|_{2^n}=\begin{cases}\left|\left|A+B\right|_{2^{2n}}+1\right|_{2^n} & \text{if } A+B\geq 2^{2n}\\ \left|\left|A+B\right|_{2^{2n}}\right|_{2^n} & \text{Otherwise}\end{cases} \tag{44}$$

Using *Theorem I*,

$$\left|\left|A+B\right|_{2^{2n}-1}\right|_{2^n}=\begin{cases}\left|A+B\right|_{2^n}+1 & \text{if } A+B\geq 2^{2n}\\ \left|A+B\right|_{2^n} & \text{Otherwise}\end{cases} \tag{45}$$

Only a few additional logic gates are needed to compute the carry-in to the modulo 2^n adder based on the Ling's carry definition [37].



$(c_0, s_0) = \overline{x_{3,0}} + \overline{x_{3,n}} + 1 = 0 + 1 + 1 = (1, 0)$
$(c_1, s_1) = \overline{x_{3,1}} + 1 + x_{3,n} = 1 + 1 + 0 = (1, 0)$

Figure 6: Calculation of scaled RNS output, y_1, y_2 and y_3 for Example 2.

V. Evaluation and Comparison

In this section, the total delay and hardware area of the proposed design are modeled and simulated in order to analyze its hardware implementation cost and complexity. In order to benchmark our proposed RNS scaler architecture against the FA based scaler architecture [19] and other RNS scaler architectures compared in [19], the unit-gate model used in [28] is adopted for our analysis. In this analytical model, a two-input monotonic gate, such as *AND* or *NAND* gate, is said to have one unit of area and one unit of delay. An *XOR* gate is deemed to consume two units of area and have two units of delay. The area and delay of an inverter is a negligible fraction of a unit and hence it is assumed to have zero unit of area and delay. Based on the unit-gate model, a FA has seven units of area and 4 units of delay.

Based on the architecture (See Fig. 5) described in Section IV, the area and delay for each residue channel can be independently evaluated by studying the area and time complexity of the logic gate implementation of the carry save adder tree, if any, and the carry propagating adder (CPA).

Based on Fig. 6 of [37], the area and delay for the Ling modulo 2^n-1 adder are estimated to be $3n\lceil log_2 n-1\rceil+12n$ and $2\lceil log_2 n-1\rceil+3$ units, respectively. From Table 2 of [28], the area and time complexity for the diminished-one mod 2^n+1 adder are $4.5n\lceil log_2 n\rceil+0.5n+6$ and $2\lceil log_2 n\rceil+3$ units, respectively. For the m_1 channel, we need n inverters for the one-complement of an operand before the CPA. A $2n$-bit CSA with EAC and n *OR* gates are required in the m_2 channel. $2n$ FAs are required to implement the CSA. Finally, from Fig. 4, two MFAs and n FAs are required to implement the CSA with CEAC for the m_3 channel. The decomposition of area and

delay for each channel is tabulated in Tables III and IV, respectively.

TABLE III: ESTIMATION OF UNIT-GATE AREA OF PROPOSED DESIGN

Modulus	A_{CPA}	A_{other}	A_{Total}
m_1	$3n\lceil log_2 n-1\rceil+12n$	≈ 0	$3n\lceil log_2 n-1\rceil+12n$
m_2	$6n\lceil log_2 n\rceil+24n$	$15n$	$6n\lceil log_2 n\rceil+39n$
m_3	$4.5n\lceil log_2 n\rceil+0.5n+6$	$7n+6$	$4.5n\lceil log_2 n\rceil+7.5n+12$

TABLE IV: ESTIMATION OF UNIT-GATE DELAY OF PROPOSED DESIGN

Modulus	D_{CPA}	D_{other}	D_{Total}
m_1	$2\lceil log_2 n-1\rceil+3$	≈ 0	$2\lceil log_2 n-1\rceil+3$
m_2	$2\lceil log_2 n\rceil+3$	5	$2\lceil log_2 n\rceil+8$
m_3	$2\lceil log_2 n\rceil+3$	6	$2\lceil log_2 n\rceil+9$

It is worth noting here that the delays of the non-CPA logic in all three modulus channels are independent of the word length, *n*. In addition, the delay of the carry propagating addition is only logarithmically dependent on the word length *n*. This means that the rate of increase in delay actually slows down as *n* grows larger. In fact, the speed is merely dependent on the performance of the modulo 2^n+1 and 2^n-1 adders. In other words, the proposed scaling algorithm has successfully circumvented the complexity of RNS scaling problem for $\{2^n-1, 2^n, 2^n+1\}$ by reducing it into an operation as simple as addition or constant multiplication in the same RNS.

We also benchmark the area and time complexity of our proposed RNS scaler against other designs reported in the RNS scaling architecture [19], including [19] itself. For ease of reference, the algorithms and scaling errors of these designs are listed in Table V. All the implementations are based on ROMs except for the proposed scaler, Dasygenis08 [19] and Garcia99 [17]. The latter is based partly on ROM-based arithmetic and partly on FAs. Since Garcia99 has significantly improved performance over Garcia99B from the same authors, Garcia99B in [19] is not listed here for comparison. From Table V, it is noted that only the proposed design and the two-lookup cycle design have no scaling error. The other scaling methods introduce some non-zero scaling error. Even though the error may be small, it could escalate to an unacceptable magnitude after several iterations of computation as shown in Fig. 1. As discussed in Section II, Dasygenis08 [19] is not able to produce the correct scaled output and it is not meaningful to analyze its scaling error.

TABLE V: ALGORITHMS AND SCALING ERRORS OF SCALERS IN COMPARISON

Reference	Algorithm	Scaling error
This	FA based	0
Dasygenis08 [19]	FA based	Not Applicable
Garcia99 [17]	Two lookup cycle	0
Ulman93 [22]	CRT based	1
Shenoy89 [14]	CRT decomposition	1.5
Griffin88 [21]	L-CRT	*N*

For consistency, we adopt the same method of evaluation as [19] so that the hardware area and time complexity of the contending designs can be excerpted directly from Table IV of [19] for comparison. Since the hardware complexity is expressed in terms of the number of transistors for all designs, the transistor count of our proposed design can be estimated from its unit-gate area by reasonably and conservatively assumed that a two-input monotonic gate can be implemented with six transistors using a classical CMOS implementation. In addition, we consider an inverter as equivalent to two transistors. The conversion of the unit gate area from Table III to the equivalent number of transistors for our proposed design is shown in Table VI.

The same unit gate delay model is used in [19] for the comparison of time complexity except that a classic FA is assumed to have only 2 units of delay in [19], which is equivalent to that of an *XOR* gate. This estimate is optimistic because the delay of a classic 28T FA implemented in static CMOS technology [39]-[41] is at least 3 units. In other words, the total delay of Dasygenis08 [19] has been underestimated.

TABLE VI: ESTIMATED NUMBER OF TRANSISTOR OF PROPOSED DESIGN

Modulus	A_{CPA}	A_{other}	A_{Total}
m_1	$18n\lceil log_2 n-1\rceil+72n$	$2n$	$18n\lceil log_2 n-1\rceil+74n$
m_2	$36n\lceil log_2 n\rceil+144n$	$90n$	$36n\lceil log_2 n\rceil+234n$
m_3	$27n\lceil log_2 n\rceil+3n+36$	$42n+36$	$27n\lceil log_2 n\rceil+45n+72$

As the use of RNS scaler is to ensure that the computed result represents the real value instead of its residue modulo *M*, it is fairer to compare different RNS scalers based on the same dynamic range. As this is not always possible due to the odd values of moduli selected by [19], the value of *n* for our proposed design has been selected such that its dynamic range is at least several time larger than the dynamic range of other designs so that the comparison is always in favor of the contenders. Tables VII and VIII show the comparisons of the transistor counts and the delay, respectively, of six different RNS scaling architectures for four different dynamic ranges.

TABLE VII: COMPARISON OF ESTIMATED NUMBER OF TRANSISTORS (VALUE IN PARENTHESIS INDICATES THE PERCENTAGE AREA REDUCTION OF PROPOSED DESIGN OVER ITS CONTENDER)

Proposed $\{m_1, m_2, m_3\}$ DR	[19] $\{m_1, m_2\}$ DR	Number of transistors ([contender−this]/contender×100%)					
		This	[19]	[17]	[22]	[14]	[21]
{7,8,9} 504	{5. 17} 85	1563	7476 (79.09)	22588 (93.08)	148862 (98.95)	20812 (92.49)	8618 (81.86)
{15, 16, 17} 4080	{17, 113} 1921	2060	10624 (80.61)	485684 (99.58)	320942 (99.36)	416816 (99.51)	87132 (97.64)
{31, 32, 33} 32736	{97, 113} 10961	2962	11940 (75.19)	928152 (99.68)	323438 (99.08)	1216816 (99.76)	89196 (96.68)
{255, 256, 257} 16776960		4696	11940 (60.67)	928152 (99.49)	323438 (98.55)	1216816 (99.61)	89196 (94.74)

It is obvious from Tables VII and VIII that the proposed design is significantly faster and has much lower hardware complexity than all the other designs. The proposed design reduces 79.09%, 80.61%, 75.19% and 60.67% of the transistor counts from the design of [19] for an extended DR of 504 (≈ six

times), 4080 (≈ twice), 32736 (≈ triple) and 16776960 (> 1500 times), respectively. Although both implementations are FA based, many more number of stages of modulo additions are required by [19] (See Fig. 2) in each channel to produce the residue digits of the scaled integer, which apparently also causes it to be the slowest design. Furthermore, the proposed design has also shorten the delay of the fastest two-lookup cycle design of [17] by respectively 23.53%, 38.10% and 28.57% for approximately six times, twice and triple the dynamic range of its contender. Owing to the logarithmically dependence on the input bit width, the delay of the proposed design increases relatively slower with dynamic range than the two-lookup cycle design. In fact, the delay of the proposed design will remain constant at 15 units for up to n = 8 (equivalent to a DR of approximately 2^{24}).

TABLE VIII: COMPARISON OF ESTIMATED UNIT-GATE DELAY (VALUE IN PARENTHESIS INDICATES % DELAY REDUCTION OF PROPOSED DESIGN OVER ITS CONTENDERS)

Proposed $\{m_1, m_2, m_3\}$ DR	[19] $\{m_1, m_2\}$ DR	Unit gate delay ([contender−this]/contender×100%)					
		This	[19]	[17]	[22]	[14]	[21]
{7,8,9} 504	{5. 17} 85	13	167 (92.22)	17 (23.53)	42 (69.05)	22 (40.91)	42 (69.05)
{15, 16, 17} 4080	{17, 113} 1921	13	307 (95.77)	21 (38.10)	50 (74)	22 (40.91)	54 (75.93)
{31, 32, 33} 32736	{97, 113} 10961	15	131 (88.55)	21 (28.57)	50 (70)	26 (42.31)	56 (73.21)
{255, 256, 257} 16776960		15	131 (88.55)	21 (28.57)	50 (70)	26 (42.31)	56 (73.21)

Although Tables VII and VIII provide a good insight on the relative competitiveness of various designs based on the same method of evaluation as [19], the moduli sets used are different from ours. For a more accurate evaluation and fairer comparison, the fastest ROM-based design [17] from Table VIII has been selected for implementation using the same three moduli set $\{2^n-1, 2^n, 2^n+1\}$ and scaling factor 2^n. In addition, another fast and precise base-extension ROM-based design [15] and the current-art adder-based design [24] that were not compared in [19] are also implemented using the same moduli set and scaling factor. According to the original algorithms, the scaling constant of [15] can be of any subset of the moduli set. However, the use of a product of lowest order moduli as scaling constant in the example of [15] may not necessarily lead to more efficient hardware implementation. It is worth noting that the [15] and [24] have zero scaling error. Unit gate area and delay of these designs are also analyzed. For this theoretical analysis, the design [24] without the sign number handling is evaluated against our design without the scaled integer output.

Since the designs, [15] and [17], are implemented using ROMs, a ROM model based on [16] is used to estimate the number of transistors, A_{ROM} and the unit gate delay, T_{ROM}. For a $2^n \times p$ ROM [16], $A_{ROM(2^n \times p)} = 2^{\lceil n/2 \rceil}(\lceil n/2 \rceil + 1) + 2^n p + 2^{\lfloor n/2 \rfloor} p(\lfloor n/2 \rfloor + 2) + p(2^{\lfloor p/2 \rfloor} + 1)$ and $T_{ROM(2^n \times p)} = (1 + \lceil \log_2 n \rceil + \lceil n/2 \rceil) \cdot t_{NAND}$. It is stated in [17] that, for three moduli set with one of the modulus being the scaling constant, the design requires three double-input ROMs. Therefore, the total number of transistors of the design [17] is estimated to be $A_{ROM(2^{2n} \times n)} + A_{ROM(2^{2n+1} \times \{n+1\})} + A_{ROM(2^{2n+1} \times n)}$, where n is the bit width of the moduli. Also, its total unit gate delay is approximate to be ($T_{ROM(2^{2n+1} \times \{n+1\})} + T_{ROM(2^{2n+1} \times n)}$) units. Based on Fig. 2 of [15], additional block of hardware should be included in each of m_1 and m_3 channels of [15] to perform the base extension operation of $\left\| X \right|_k \Big|_{m_i}$, for i = 1 and 3. In this special case where $k = 2^n$ and $m_3 > m_1$, only one ROM is required in the m_1 channel. As a result, the total hardware of the design [15] is estimated to be $A_{ROM(2^{2n} \times n)} + A_{ROM(2^{2n+1} \times \{n+1\})} + A_{ROM(2^{2n+1} \times n)} + A_{ROM(2^n \times n)}$ and its total unit gate delay is $T_{ROM(2^{2n} \times n)} + T_{ROM(2^{2n+1} \times n)} + T_{ROM(2^n \times n)}$ units. For the design [24], the main difference lies in the m_2 channel. As depicted in Fig. 1 of [24], the m_2 channel consists of one modulo 2^n–1 adder and one modulo 2^n adder excluding the sign number handling. The former adder has been analyzed at the beginning of this section while the unit gate area and delay of the latter adder are approximately $1.5n\lceil \log n \rceil + 5n$ and $2\lceil \log n \rceil + 3$, respectively [28]. For our proposed design without the additional scaled integer output feature, the modulo 2^{2n}–1 adder is replaced with a modulo 2^n adder and some glue logic for the Ling's carry. The estimated total number of transistors and unit gate delay are tabulated in Table IX.

TABLE IX: COMPARISON OF ESTIMATED TOTAL NUMBER OF TRANSISTORS (A) AND UNIT GATE DELAY (T) FOR THE SAME THREE MODULI SETS

$\{m_1, m_2, m_3,\}$	This		[24]		[17]		[15]	
	A	T	A	T	A	T	A	T
{31, 32, 33}	2095	17	2077	31	32424	22	32721	29
{63, 64, 65}	2478	17	2478	31	143467	24	144177	31
{127, 128, 129}	2861	17	2879	31	637446	26	638765	34
{255, 256, 257}	3244	17	3280	31	2829481	30	2832513	38

All these full-feature designs (with sign handling as reported in [24] and with both scaled residue and integer outputs for ours) are coded using Verilog HDL and functionally verified using ModelSim. The designs are synthesized and mapped to STM 90nm standard cell library using Synopsys Design Compiler. Each design is optimized for speed to obtain their minimum achievable delay. The synthesized area in μm^2 and critical path delay in ns of these $\{2^n-1, 2^n, 2^n+1\}$ RNS scalers for n = 5, 7 and 8 are shown in Table IX. Due to the memory quota, the two LUT-based scalers are implemented only for n = 5 and 7.

TABLE X: COMPARISON OF THE SYNTHESIZED AREA AND DELAY OF $\{2^n-1, 2^n, 2^n+1\}$ RNS SCALERS FOR n = 5, 7 AND 8

Design	n = 5		n = 7		n = 8	
	Area	*Delay*	*Area*	*Delay*	*Area*	*Delay*
This	3168.8	0.87	4590.2	0.88	4869.0	0.93
[24]	4166.5	1.47	6786.5	1.58	7498.8	1.66
[17]	12315.0	0.98	13782.6	1.67	-	-
[15]	10147.3	1.22	15645.2	1.87	-	-

From Table X, it is obvious that our proposed RNS scaler consumes the least area for all values of *n*. Comparing with the next most area efficient design [24], our proposed design consumes 23.9%, 32.4% and 35.1% less area for n = 5, 7 and 8, respectively. Besides, our proposed design is faster than it by 40.8%, 44.3% and 44.0% for n = 5, 7 and 8, respectively. The significant performance improvement over [24] is attributable to the different approaches to the formulation of scaling problem although both scaling algorithms are designed based on adders. From the synthesis results, it is observed that the critical path delay of our proposed design remains relatively constant for three different values of *n*. This is consistent with the inference from the unit gate delay estimation in Table IV. Comparing with the fastest design [17] from Table VIII, our design is faster than it by 11.2% and 47.3% for n = 5 and 7 respectively.

The power consumption of the above designs were also simulated using PrimeTime PX with the same technology library. The total power consumption as well as the leakage power for n = 5, 7 and 8 are listed in Table XI. The results show that our proposed design dissipates the least total power as well as leakage power for all values of *n*. It consumes 39.3%, 50.2% and 43.4% less total power and 37.0%, 44.1%, 43.1% less leakage power than the next most power-efficient design [24] for n = 5, 7 and 8, respectively.

TABLE XI: COMPARISON OF TOTAL POWER AND LEAKAGE POWER IN *mW* (VALUE IN PARENTHESIS IS THE PERCENTAGE OF LEAKAGE POWER) FOR n = 5, 7 AND 8.

Design	n = 5		n = 7		n = 8	
	Total	*Leakage*	*Total*	*Leakage*	*Total*	*Leakage*
This	3.796	0.670 (17.7%)	4.955	0.955 (19.3%)	5.821	1.020 (17.5%)
[24]	6.258	1.063 (17.0%)	9.948	1.708 (17.2%)	10.287	1.793 (17.4%)
[17]	13.761	3.140 (22.8%)	16.607	3.144 (18.9%)	-	-
[15]	21.083	2.473 (11.7%)	19.313	3.943 (20.4%)	-	-

VI. CONCLUSION

In this paper, a novel area-efficient, high-speed and precise RNS scaler for the three-moduli set RNS $\{2^n-1, 2^n, 2^n+1\}$ is proposed. The new scaling algorithm is formulated based on the Chinese New Remainder Theorem and it uniquely exploits the number theoretic properties of this moduli set and the scaling factor of 2^n to overcome the complexity associated with the hardware implementation of inter-modulo operation. The proposed RNS scaler has an area complexity of $O(n\log_2 n)$ and a time complexity of $O(\log_2 n)$. The integer scaled output in normal binary number representation is also generated as a byproduct of this new formulation. Hence, the expensive residue-to-binary converter can be saved if the result after scaling is also required by a normal binary number system. Our synthesis results based on 90nm standard-cell based implementation show that the proposed RNS scaler is approximately 30.5% more area efficient and 43.0% faster than the state-of-the-art adder-based scaler and at least 66% more area efficient and 11.2% faster than the fastest ROM-based scaler design. On average, the total power consumption and leakage power of our proposed design are respectively 44.3% and 41.4% smaller than the most power-efficient adder-based scaler design.

REFERENCES

[1] M. N. Mahesh, and M. Mehendale, "Low power realization of residue number system based FIR filters," in *Proc, of 13th Int. Conf. on VLSI Design*, Jan. 2000, pp. 30-33.

[2] G. C. Cardarilli, A. D. Re, A. Nannarelli, and M. Re, "Low power and low leakage implementation of RNS FIR filters," in *Proc. 39th Asilomar Conf. on Signals, Syst. and Computer*, 2005, pp. 1620-1624.

[3] G. C. Cardarilli, A. Nannarelli, and M. Re, "Reducing power dissipation in FIR Filters using the residue number system," in *Proc. of the 43rd IEEE Midwest Symp. on Circuits and Syst.*, 2000, pp. 320-323.

[4] R. Conway, and J. Nelson, "Improved RNS FIR filter architectures," *IEEE Trans. on Circuits and Syst. II*, vol. 51, no. 1, pp. 26-28, Jan. 2004.

[5] A. Nannarelli, M. Re, and G. C. Cardarilli, "Tradeoffs between residue number system and traditional FIR filters," in *Proc. 2001 IEEE Int. Symp. on Circuits and Syst. (ISCAS2001)*, Sydney, Australia, May, 2001, pp. 305-308.

[6] A. D. Re, A. Nannarelli, and M. Re, "Implementation of digital filters in carry-save residue number system," in *Proc. Conf. Rec. of the Asilomar Conf. on Signals, Syst. and Computers*, Pacific Grove, California, USA, Nov. 2001, pp. 1309-1313.

[7] G. L. Bernocchi, G. C. Cardarili, A. Nannarelli and M. Re, "Low power adaptive filter based on RNS components," in *Proc. 2007 IEEE Int. Symp. on Circuits and Syst.*, New Orleans, USA, pp. 3211-3214, May 2007.

[8] Y. Wang, X. Song, M. Aboulhamid, and H. Shen, "Adder based residue to binary number converters for $(2^n-1, 2^n, 2^n+1)$," *IEEE Trans. on Signal Processing*, vol. 50, no. 7, pp. 1772-1779, Jul. 2002.

[9] B. Vinnakota, and V. V. Bapeswara Rao, "Fast conversion technique for binary-residue number system," *IEEE Trans, on Circuits and Syst. I*, vol. 41, no. 12, pp. 927-929, Dec. 1994.

[10] N. Szabo, and R. Tanaka, *Residue Arithmetic and Its Application to Computer Technology*. New York: McGraw-Hill, 1967.

[11] B. Cao, C. H. Chang and T. Srikanthan, "A residue-to-binary converter for a new five-moduli set," *IEEE Trans. on Circuits and Syst.–I*, vol. 54, no. 5, pp. 1041-1049, May 2007.

[12] B. Cao, T. Srikanthan and C. H. Chang, "Efficient reverse converters for the four-moduli sets $\{2^n-1, 2^n, 2^n+1, 2^{n+1}-1\}$ and $\{2^n-1, 2^n, 2^n+1, 2^{n-1}-1\}$," *IEE Proc. Computers and Digital Tech.*, vol. 152, no. 5, pp. 687-696, Sept. 2005.

[13] B. Cao, C. H. Chang and T. Srikanthan, "An efficient reverse converter for the 4-moduli set $\{2^n-1, 2^n, 2^n+1, 2^{2n}+1\}$ based on the New Chinese Remainder Theorem," *IEEE Trans. on Circuits and Syst.–I*, vol. 50, no. 10, pp. 1296-1303, Oct. 2003.

[14] M. A. P. Shenoy, and R. Kumaresan, "A fast and accurate RNS scaling technique for high speed signal processing," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 37, no. 6, pp. 929-937, Jun. 1989.

[15] F. Barsi, and M.C. Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations," *IEEE Trans. on Signal Processing*, vol. 43, no. 10, pp. 2427-2430, Oct. 1995.

[16] Z.D. Ulman, and M. Czyzak, "Highly parallel, fast scaling of numbers in nonredundant residue arithmetic," *IEEE Trans. on Signal Processing*, vol. 46, no. 2, pp. 487-496, Feb. 1998.

[17] A. Garcia, and A. Lloris, "A look-up scheme for scaling in the RNS," *IEEE Trans. on Computers*, vol. 48, no. 7, pp. 748-751, Jul. 1999.

[18] U. Meyer-Base, T. Stouraitis, "New power-of-2 RNS scaling scheme for cell-based IC design," *IEEE Trans. on VLSI Syst.*, vol. 11, no. 2, pp. 280-283, Apr. 2003.

[19] M. Dasygenis, K. Mitroglou, D. Soudris, and A. Thanailakis, "A full-adder-based methodology for the design of scaling operation in Residue Number System," *IEEE Trans. on Circuits and Syst. I*, vol. 55, no. 2, pp. 546-558, Mar. 2008.

[20] M. Griffin, M. Sousa, and F. Taylor, "Efficient scaling in the residue number system," in *Proc. 1989 Int. Conf. on Acoustics, Speech and Signal Processing*, Glasgow, Scotland, UK, May. 1989, pp. 1075-1078.

[21] M. Griffin, F. Taylor, and M. Sousa, "New scaling algorithm for the Chinese Remainder Theorem," in *Proc. Conf. Rec. Twenty-Second Asilomar Conf. on Signals, Syst. and Computers*, 1988, pp. 375-378.
[22] Z. D. Ulman, M. Czyzak, and J. M. Zurada, "Effective RNS scaling algorithm with the Chinese Remainder Theorem decomposition," in *Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, Victoria, BC, Canada, May, 1993, pp. 528-531.
[23] K. Yinan, and B. Phillips, "Fast scaling in the residue number system," *IEEE Trans. on VLSI Syst.*, vol. 17, no. 3, pp. 443-447, Mar. 2009.
[24] Y. Ye, S. Ma, and J. Hu, "An efficient 2^n RNS scaler for moduli set $\{2^n-1, 2^n, 2^n+1\}$," in *IEEE Symp. on Information Science and Engineering (ISISE 2008)*, Shanghai, China, Dec. 2008, pp. 511-515.
[25] S. Ma, J. Hu, Y. Ye, L. Zhang, and X. Ling, "A 2^n scaling scheme for signed RNS integers and its VLSI implementation," Science China, Information Sciences, vol. 53, no. 1, pp. 203-212, Jan. 2010.
[26] H. T. Vergos, D. Bakalis, and C. Efstathiou, "Fast modulo 2^n+1 multi-operand adders and residue generators," *Integration, the VLSI J.*, vol. 43, no. 1, pp. 42-48, Jan. 2010.
[27] H. T. Vergos, and C. Efstathiou, "Efficient modulo 2^n+1 adder architectures," *Integration, the VLSI J.*, vol. 42, no. 2, pp. 149-157, Feb. 2009.
[28] H. T. Vergos, C. Efstathiou, and D. Nikolos, "Diminished-one modulo 2^n+1 adder design," *IEEE Trans. on Computers*, vol. 51, no. 12, pp. 1389-1399, Dec. 2002.
[29] S. H. Lin, M. H. Sheu, K. H. Wang, J. J. Zhu, and S. Y. Chen, "Efficient VLSI design of modulo 2^n+1 adder using hybrid carry selection," in *Proc. IEEE Workshop on Signal Processing Syst., SiPS: Design and Implementation*, Shanghai, China, Oct. 2007, pp. 141-145.
[30] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Modulo $2^n\pm1$ adder design using select-prefix blocks," *IEEE Trans. on Computers*, vol. 52, no. 11, pp. 1399-1406, Nov. 2003.
[31] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-time efficient modulo 2^n-1 adder design," *IEEE Trans. on Circuits and Syst. II*, vol. 41, no. 7, pp. 463-467, Jul. 1994.
[32] R. Muralidharan, C. H. Chang, and C. C. Jong, "A low complexity modulo 2^n+1 squarer design," in *Proc. IEEE Asia Pacific Conf. on Circuits and Syst.*, Macao, China, Dec. 2008, pp. 1296-1299.
[33] S. Menon, and C. H. Chang, "A reconfigurable multi-modulus modulo multiplier," in *Proc. IEEE Asia Pacific Conf. on Circuits and Syst.*, Singapore, Dec. 2006, pp. 1168-1171.
[34] D. Bakalis, and H. T. Vergos, "Shifter circuits for 2^n+1, 2^n, 2^n-1 RNS," *Electronics Letters*, vol. 45, no. 1, pp. 27-29, Jan. 2009.
[35] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Trans. on Computers*, vol. 43, no. 1, pp. 68-77, Jan. 1994.
[36] S. J. Piestrak, "Design of squarers modulo *A* with low-level pipelining," *IEEE Trans. on Circuits and Systems – II*, vol. 49, no. 1, pp. 31-41, Jan 2002.
[37] G. Dimitrakopoulos, D. G. Nikolos, H. T. Vergos, D. Nikolos, and C. Efstathiou, "New architectures for modulo 2^n-1 adders," in *Proc. IEEE Int. Conf. on Electronics, Circuits, and Syst.*, Gammarth, Tunisia, Dec. 2005, pp. 1-4.
[38] A. A. Hiasat, "High-speed and reduced-area modular adder structure for RNS," *IEEE Trans. on Computers*, vol. 51, no. 1, pp. 84-89, Jan. 2002.
[39] C. H. Chang, J. Gu and M. Zhang, "A review of 0.18μm full adder performances for tree structured arithmetic circuits," *IEEE Trans. on VLSI Syst.*, vol. 13, no. 6, pp. 686-695, Jun. 2005.
[40] C. H. Chang, J. Gu and M. Zhang, "Ultra low voltage, low power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits," *IEEE Trans. on Circuits and Syst.-I*, vol. 51, no. 10, pp. 1985-1997, Oct. 2004.
[41] A. Shams, T. Darwish and M. Bayoumi, "Performance analysis of low-power 1-bit CMOS full adder cells," *IEEE Trans. on VLSI Syst.*, vol. 10, no. 1, pp. 20-29, Feb 2002.
[42] R. Conway, and J. Nelson, "Fast converter for 3 moduli RNS using new property of CRT," *IEEE Trans. on Computers*, vol. 48, no. 8, pp. 852-860, Aug. 1999.
[43] W. Wang, M. N. S. Swamy, M. O. Ahmad, and Y. Wang, "A study of the residue-to-binary converters for the three-moduli sets," *IEEE Trans. on Cir. and Syst.-I*, vol. 50, no. 2, pp. 235-243, Feb. 2003.

Chip-Hong Chang (S'92–M'98–SM'03) received the B.Eng. (Hons.) degree from the National University of Singapore, Singapore, in 1989, and the M. Eng. and Ph.D. degrees from Nanyang Technological University (NTU), Singapore, in 1993 and 1998, respectively. He served as a Technical Consultant in industry prior to joining the School of Electrical and Electronic Engineering (EEE), NTU, in 1999, where he is currently an Associate Professor. He holds joint appointments with the university as Assistant Chair of Alumni of the School of EEE since June 2008, Deputy Director of the Center for High Performance Embedded Systems since 2000, and the Program Director of the Center for Integrated Circuits and Systems from 2003 to 2009. He has coedited one book, published three book chapters and more than 170 research papers in refereed international journals and conferences. His current research interests include low power arithmetic circuits, residue number system, digital filter design, and digital watermarking for VLSI intellectual property protection. He was the co-recipient of Gold Leaf and Silver Leaf awards at PrimeAisa 2010.

Dr. Chang serves as Associate Editor of the IEEE Transactions on Circuits and Systems-I in 2010-2011 and the IEEE Transactions on Very Large Scale Integration (VLSI) Systems since 2011, as well as the Guest Editor for the February 2011 special issue on Green Circuits and Systems of the *Journal of Circuits, Systems and Computers*. He is the Editorial Advisory Board Member of the *Open Electrical and Electronic Engineering Journal* since 2007 and the Editorial Board Member of the *Journal of Electrical and Computer Engineering* since 2008. He has also served in many international conference advisory and technical program committees. He is a Fellow of the IET.

Jeremy Yung Shern Low received his B.Eng. (Hons) degree in Electrical and Electronics Engineering from Nanyang Technological University (NTU), Singapore, in 2009. He is currently pursuing his Ph.D. degree in the division of Circuits and Systems, School of EEE, NTU, Singapore. His area of research is high speed, low power and fault tolerant computer arithmetic circuits.