Scaling an RNS number using the core function

Neil Burgess,
Cardiff School of Engineering,
Cardiff University,
Queen's Buildings,
The Parade,
CARDIFF CF24 3TF
U.K.

Abstract

This paper introduces a method for extracting the core of a Residue Number System (RNS) number within the RNS, this affording a new method for scaling RNS numbers. Suppose an RNS comprises a set of co-prime moduli, m_i, with $\prod m_i = M$. This paper describes a method for approximately scaling such an RNS number by a subset of the moduli, $\prod m_j = M_J \approx \sqrt{M}$, with the characteristic that all computations are performed using the original moduli and one other non-maintained short wordlength modulus.

1. Background and Motivation

The Residue Number System (RNS) has great potential for accelerating arithmetic operations, achieved by breaking operands into several smaller residues and operating on the residues independently and in parallel. RNS implementations were studied extensively in the 1970's, particularly for DSP applications [1], and led to Inmos' production of an RNS 2-D convolver chip in 1989 [2]. However, wider take-up of RNS for DSP was limited because of a number of fundamental difficulties:

- Conversion to binary representation from RNS is difficult (the inverse operation is simple)
- Direct magnitude comparison and sign determination of RNS numbers is impossible
- Square root operations are not available, and division operations, although available [3], are not practical due to their complexity

These difficulties place major constraints on the possible applications of RNS arithmetic.

Recently, however, DSP chips using RNS have enjoyed something of a renaissance for a variety of reasons:

- They offer high-performance implementations of arithmetic-intensive applications at reduced power supply voltages, important for mobile and wearable computer and communication systems [4]
- They avoid lengthy on-chip interconnects, which now represent the major constraint on the realisation of high-performance digital VLSI circuits [5]
- They afford hardware-efficient complex multipliers ("QRNS multiplication") comprising two independent multiplications instead of four multiplications and two additions [1]
- The component arithmetic operations in an RNS implementation can, without exception, be reduced to short adders and small look-up tables [1]

All the items in the above list are applicable to custom VLSI implementations, and the last two also apply advantageously to FPGA implementations [6,7]. Recent industrial interest in RNS confirms the existence and scale of problems faced in implementing DSP algorithms in digital microelectronic fabrics at high clock rates but with low power consumption. For example, reference [8] describes an FIR filter in RNS designed by Texas Instruments because of its low-power capability, and reference [9] discusses a general-purpose DSP engine developed by Siemens that incorporates an RNS vector processor with a considerably higher data processing bandwidth than its binary counterpart.

The fundamental difficulties with RNS arithmetic listed earlier have been overcome to some extent by recent innovations in RNS theory. For example, the core function has been shown to be advantageous in converting an RNS number to binary [10], and for adding extra moduli to an RNS in order to increase its dynamic range ("base extension") [11]. The outstanding problem with RNS processing that prevents its wider take-up is reducing an RNS number's wordlength through scaling – that is, dividing – by a constant with low latency and minimal hardware cost. In binary arithmetic, the scaling constant is invariably set to a power of two so that wordlength reduction is achieved simply by truncating (or rounding) a number. There is no equivalent operation in an RNS with the consequence that the wordlength growth of an accumulated result through a sequence of multiplications, such as is encountered in a multiple-point FFT or in an IIR filter, is very difficult to manage.

A number of algorithms for scaling RNS numbers have been reported, but as yet none operates entirely within an RNS. Early attempts at scaling fell into two categories: scaling by one modulus, whereby the RNS number was adjusted to be divisible by one of the moduli, dividing by that modulus in all the other moduli in a single step, and finally base extending the scaled number back into the "scaling modulus" (e.g. [12]); or performing a truncated conversion to binary – that is, scaling by a power of two – followed by conversion back into RNS representation (e.g. [13]). However, these methods are

generally slow and require processing of longer word-length numbers outside the RNS.

A major advance was made by Shenoy and Kumaresan [14], who devised a novel decomposition of the Chinese Remainder Theorem that enabled scaling by the product of several moduli. However, their scheme was not optimal in that an extra modulus with a similar wordlength to the existing moduli outside the RNS was employed, requiring extra hardware (typically >10%) for its maintenance, and two redundant channels of residue computation were necessary in the scaling algorithm itself. The total hardware count for Shenoy and Kumaresan's RNS scaler operating on k moduli was $k\cdot(k+4)$ modulo arithmetic multiply-accumulates (MACs). Recent work has concentrated on removing the extra modulus in Shenoy and Kumaresan's scheme at the expense of increasing the logical depth of the scaler [15], or of reducing the accuracy of the scaler by limited use of binary arithmetic outside the RNS channels [16].

This paper introduces a novel technique for scaling an RNS number, based on the core function. The method consists simply of extracting the core of the RNS number within the RNS. All computations reduce to inner products within the moduli of the RNS, with one extra inner product using a modulus outside the RNS but not requiring maintenance of the corresponding residue. The paper is structured as follows: first some preliminaries regarding the core function are dealt with; then, the proposed scaling algorithm is introduced along with an example; next, difficulties with the proposed algorithm are identified and a workaround described; finally, the paper concludes with a brief discussion of possible further avenues of research.

2 Scaling an RNS number using the Core Function

2.1 The Core Function

The core function is defined for an integer, n, as:

$$C(n) = \sum_i w_i \cdot \left\lfloor \frac{n}{m_i} \right\rfloor = \sum_i (n - n_i) \cdot \frac{w_i}{m_i} \qquad \text{—— (1)}$$

where n_i denotes n mod m_i and w_i denotes the i^{th} weight. Setting $n = M$ in (1) gives:

$$C(M) = \sum_i M \cdot \frac{w_i}{m_i} \qquad \text{—— (2)}$$

so that:

$$\frac{C(M)}{M} = \sum_i \frac{w_i}{m_i} \qquad \text{—— (3)}$$

This implies some values of w_i must be negative to obtain small values of $C(M)$. Substituting (3) into (1) gives:

$$C(n) = n \cdot \frac{C(M)}{M} - \sum_i n_i \cdot \frac{w_i}{m_i} \qquad \text{—— (4)}$$

Equation (4) indicates that a plot of $C(n)$ against n should reveal a straight line with slope $C(M)/M$ with some "furriness" due to the superimposed summation term. The magnitude of the furriness is set by the magnitude of the weights, in turn related to the particular value of $C(M)$ for a given RNS modulus set.

The weights, w_i, are determined by re-arranging (3) and reducing both sides modulo m_j:

$$\langle C(M) \rangle_{m_j} = \left\langle \sum_i \frac{w_i \cdot M \cdot}{m_i} \right\rangle_{m_j} = \left\langle w_j \cdot M_j^* \right\rangle_{m_j} \qquad \text{—— (5)}$$

(M_i^* mod $m_j = 0$ for all m_i except m_j.) Re-arranging (5):

$$w_j = \left\langle C(M) \cdot M_j^{*-1} \right\rangle_{m_j} \qquad \text{—— (6)}$$

so the weights may be derived once $C(M)$ has been chosen, but with the proviso that (2) is satisfied, implying that some of the weights must be negative to ensure $C(M) << M$. Note that if $C(M)$ is a multiple of m_j, the corresponding weight, $w_j = 0$. Finally, from [10], the range of a core function, $G(M)$ is given as:

$$G(M) = C(M) + \sum_i |w_i| \qquad \text{—— (7)}$$

Now, the Chinese Remainder Theorem for converting RNS numbers back to positional (i.e. decimal or binary) representation can be expressed as:

$$n = \sum_i n_i \cdot B_i - R(n) \cdot M \qquad \text{—— (8)}$$

where $R(n)$ is known as the rank function, and B_i denotes the i^{th} base of the RNS:

$$B_i = \frac{M}{m_i} \cdot \left\langle \left(\frac{M}{m_i} \right)^{-1} \right\rangle_{m_i} = M_i^* \left\langle M_i^{*-1} \right\rangle_{m_i} \qquad \text{—— (9)}$$

Substituting (8) into (4) gives:

$$C(n) = \frac{C(M)}{M} \cdot \left\{ \sum_i n_i \cdot B_i - R(n) \cdot M \right\} - \sum_i n_i \cdot \frac{w_i}{m_i} \qquad \text{—— (10)}$$

Simplifying and re-arranging (10) yields:

$$C(n) = \sum_i n_i \cdot \left(\frac{B_i \cdot C(M)}{M} - \frac{w_i}{m_i} \right) - R(n) \cdot C(M) \qquad \text{—— (11)}$$

Setting $n = B_i$ in (4):

$$C(B_i) = B_i \cdot \frac{C(M)}{M} - \frac{w_i}{m_i} = \frac{C(M) \cdot \left\langle M_i^{*-1} \right\rangle_{m_i} - w_i}{m_i} \quad \text{—— (12)}$$

Whence:

$$C(n) = \cdot \sum_i n_i \cdot C(B_i) - R(n) \cdot C(M) \quad \text{—— (13)}$$

which is known as the Chinese Remainder Theorem of Core Functions. However, owing to the unfeasibility of computing $R(n)$ independently, the preferred form of the Chinese Remainder Theorem for Core Functions is:

$$\left\langle C(n) \right\rangle_{C(M)} = \left\langle \sum_i n_i \cdot C(B_i) \right\rangle_{C(M)} \quad \text{—— (14)}$$

An example should help make things clearer. Consider an RNS with the modulus set, m_i = {2, 3, 5, 7, 11, 13}, giving M = 30,030, and M_i^{*-1} = {1, 2, 1, 6, 6, 3}. Next, choose $C(M)$ = 165. Then, from (5), the weights are found as:

$w_1 = \langle 165 \times 1 \rangle_2 = +1$ or -1
$w_2 = \langle 165 \times 2 \rangle_3 = 0$
$w_3 = \langle 165 \times 1 \rangle_5 = 0$
$w_4 = \langle 165 \times 6 \rangle_7 = +3$ or -4
$w_5 = \langle 165 \times 6 \rangle_{11} = 0$
$w_6 = \langle 165 \times 3 \rangle_{13} = +1$ or -12

In order to minimise the "furriness" in the core function, weights with small magnitudes should be chosen. In this example, the weight set w_i = {-1, 0, 0, 3, 0, 1} is chosen, and its legitimacy can be checked against equation (2):

$C(M) = -1 \times 15015 + 3 \times 4290 + 1 \times 2310 = 165$

This core function is plotted in Figure 1. An alternative weight set w_i = {1, 0, 0, -4, 0, 1} is available that has the useful property $C(n) \geq 0$ if $n \geq 0$. The legitimacy of this weight set can also be checked as follows:

$C(M) = 1 \times 15015 + -4 \times 4290 + 1 \times 2310 = 165$

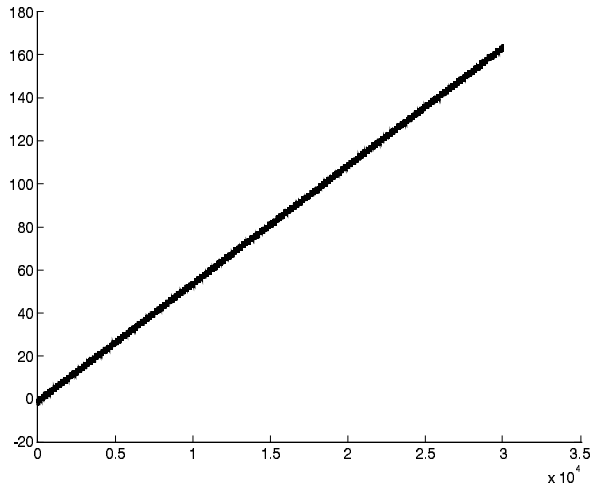

Figure 1 Plot of typical core function: M = 30,030; $C(M)$ = 165

2.2 RNS Scaling Method

This paper proposes an RNS scaling method that consists of extracting the core of a number within the RNS. From equation (4):

$$C(n) = n \cdot \frac{C(M)}{M} - \sum_i n_i \cdot \frac{w_i}{m_i} \quad \text{—— (4)}$$

Thus, if $C(n)$ could be computed within the RNS, an approximate scaled version of n is obtained. This can be achieved by splitting a modulus set into two sub-sets, M_J and M_K, such that $M_J \cdot M_K = M$ and $M_J / M_K \approx 1$. Then, it is possible to perform scaling by either M_J or M_K (in other words, extract $C(n)$ with $C(M) = M_J$ or $C(M) = M_K$) as follows.

First, set $C_J(M) = M_J = \prod m_j$. Then, from (14):

$$\left\langle C_J(n) \right\rangle_{C_J(M)} = \left\langle \sum_i n_i \cdot C_J(B_i) \right\rangle_{C_J(M)} \quad \text{—— (15)}$$

for the sub-set of moduli that make up M_J. But, m_j is a factor of $C_J(M)$, so that:

$$\left\langle \left\langle C_J(n) \right\rangle_{C_J(M)} \right\rangle_{m_j} = \left\langle C_J(n) \right\rangle_{m_j} = \left\langle \sum_i n_i \cdot C_J(B_i) \right\rangle_{m_j} \quad \text{—— (16)}$$

However, for the for the sub-set of moduli that make up M_K - namely the moduli, m_k, that do not divide M_J - the same simplification is not possible:

$$\left\langle \left\langle C_J(n) \right\rangle_{C_J(M)} \right\rangle_{m_k} = \left\langle \left\langle \sum_i n_i \cdot C_J(B_i) \right\rangle_{C_J(M)} \right\rangle_{m_k} \quad \text{—— (17)}$$

Thus, equation (15) may be computed within the subset of moduli, m_j, but (16) may not. Similarly, if $C_K(M) = M_K = \prod m_k$:

$$\langle C_K(n)\rangle_{m_k} = \left\langle \sum_i n_i \cdot C_K(B_i) \right\rangle_{m_k} \quad \text{—— (18)}$$

which is computable within the subset of moduli, m_k. Hence, $C_J(n) \approx n/M_K$ may be calculated within the moduli sub-set, M_J, but not the sub-set, M_K; also, $C_K(n) \approx n/M_J$ may be calculated within the moduli sub-set, M_K, but not the sub-set, M_J. However, if the difference between the cores $\Delta C(n) = C_J(n) - C_K(n)$ can be calculated, $C_K(n)$ modulo the subset M_K can be extended into $C_J(n)$ modulo the subset M_K. In other words, by adding $\Delta C(n)$ to (or subtracting it from) the values of one sub-set of scaled residues either $C_J(n)$ or $C_K(n)$ is available across all the residues, and a scaled value of n is obtained within the RNS. A simple expression for $\Delta C(n)$ is obtained from (13) as:

$$\Delta C(n) = \sum_i n_i \cdot C_J(B_i) - R(n) \cdot C_J(M) - \left\{ \sum_i n_i \cdot C_K(B_i) - R(n) \cdot C_K(M) \right\} \quad \text{—— (19)}$$

which may be simplified to read:

$$\Delta C(n) = \sum_i n_i \cdot \Delta C(B_i) - R(n) \cdot \Delta C(M) \quad \text{—— (20)}$$

where $\Delta C(B_i) = C_J(B_i) - C_K(B_i)$, and $\Delta C(M) = C_J(M) - C_K(M)$. However, given the difficulty of determining the value of $R(n)$ from the residues, a more useful form of equation (20) is:

$$\langle \Delta C(n)\rangle_{\Delta C(M)} = \left\langle \sum_i n_i \cdot \Delta C(B_i) \right\rangle_{\Delta C(M)} \quad \text{—— (21)}$$

This expression will be most conveniently evaluated if $\Delta C(M)$ is of similar wordlength to the moduli, m_i (or a not-so-small power of two).

2.3 Worked example of proposed RNS scaling algorithm

Suppose an RNS has the moduli set, {7, 11, 13, 17, 19, 23}: then M = 7,436,429. The moduli set is split into two groups, $M_J = 7\times17\times23 = 2737$ and $M_K = 11\times13\times19 = 2717$, to give $\Delta C(M) = 20$. Scaling a residue number by either 2737 or 2717 is practical because $\Delta C(M)$ has a similar wordlength to the moduli.

The values of M_i* and M_i*$^{-1}$ are {1,062,347, 676,039, 572,033, 437,437, 391,391, 323,323} and {6, 1, 2, 12, 2, 2} respectively. The weight set for $C_J(M) = 2737$ is w_i = {0, -2, 1, 0, 2, 0) and for $C_K(M) = 2717$, w_i ={-1, 0, 0, -2, 0, 6}. The two sets of $C(B_i)$ then follow from (12): $C_J(B_i)$ = {2346, 249, 421, 1932, 288, 238}; $C_K(B_i)$ = {2329, 247, 418, 1918, 286, 236}; finally, $\Delta C(B_i)$ = {17, 2, 3, 14, 2, 2}.

The number n = 1,859,107 is to be approximately scaled by 2717 to yield ≈ 684. That is, we will compute $C_J(n)$ wholly within the RNS. n is represented as (5, 8, 3, 4, 14, 17) by this set of moduli. First, compute $C_J(n)$ moduli 7, 17, and 23, and $C_K(n)$ moduli 11, 13, and 19 using (15) and (17):

$C_J(n)$ mod 7 = (5×2346 + 8×249 + 3×421 + 4×1932 + 14×288 + 17×238) mod 7 = (5×1 + 8×4 + 3×1 + 4×0 + 14×1 + 17×0) mod 7 = 5

$C_J(n)$ mod 17 = (5×2346 + 8×249 + 3×421 + 4×1932 + 14×288 + 17×238) mod 17 = (5×0 + 8×11 + 3×13 + 4×11 + 14×16 + 17×0) mod 17 = 4

$C_J(n)$ mod 23 = (5×2346 + 8×249 + 3×421 + 4×1932 + 14×288 + 17×238) mod 23 = (5×0 + 8×19 + 3×7 + 4×0 + 14×12 + 17×8) mod 23 = 17

$C_K(n)$ mod 11 = (5×2329 + 8×247 + 3×418 + 4×1918 + 14×286 + 17×236) mod 11 = (5×8 + 8×5 + 3×0 + 4×4 + 14×0 + 17×5) mod 11 = 5

$C_K(n)$ mod 13 = (5×2329 + 8×247 + 3×418 + 4×1918 + 14×286 + 17×236) mod 13 = (5×2 + 8×0 + 3×2 + 4×7 + 14×0 + 17×2) mod 13 = 0

$C_K(n)$ mod 19 = (5×2329 + 8×247 + 3×418 + 4×1918 + 14×286 + 17×236) mod 19 = (5×11 + 8×0 + 3×0 + 4×18 + 14×1 + 17×8) mod 19 = 11

In parallel, calculate $\langle \Delta C(n)\rangle_{\Delta C(M)}$ using (21):

$\Delta C(n)$ mod 20 = (5×17 + 8×2 + 3×3 + 4×14 + 14×2 + 17×2) mod 20 = 8

Finally, add $\Delta C(n)$ to the $C_K(n)$ values to obtain the remaining scaled moduli:

$C_J(n)$ mod 11 = $C_K(n)$ + $\Delta C(n)$ mod 11 = 5+8 mod 11 = 2

$C_J(n)$ mod 13 = $C_K(n)$ + $\Delta C(n)$ mod 13 = 0+8 mod 13 = 8

$C_J(n)$ mod 19 = $C_K(n)$ + $\Delta C(n)$ mod 19 =11+8 mod 19 = 0

Hence, the RNS value of n = 1,859,107 (or (5, 8, 3, 4, 14, 17) in RNS format) after being approximately scaled by 2717 is $C_J(n)$ = (5, 2, 8, 4, 0, 17).

We can check the result by converting $C_J(n)$ back to decimal using the Chinese Remainder Theorem:

$C_J(n) = \Sigma n_i \times B_i \bmod M$ = 5×1062347×6 + 2×676039×1 + 8×572033×2 + 4×437437×12 + 0×391391×2 + 17×323323×2 mod 7436429 = 31870410 + 1352078 + 9152528 + 20996976 + 0 + 10992982 = 74364974 mod 7436429 = 684.

A block diagram of this calculation method, emphasising the consistent use of short wordlength arithmetic and making explicit the degree of available parallelism, is presented in Figure 2.

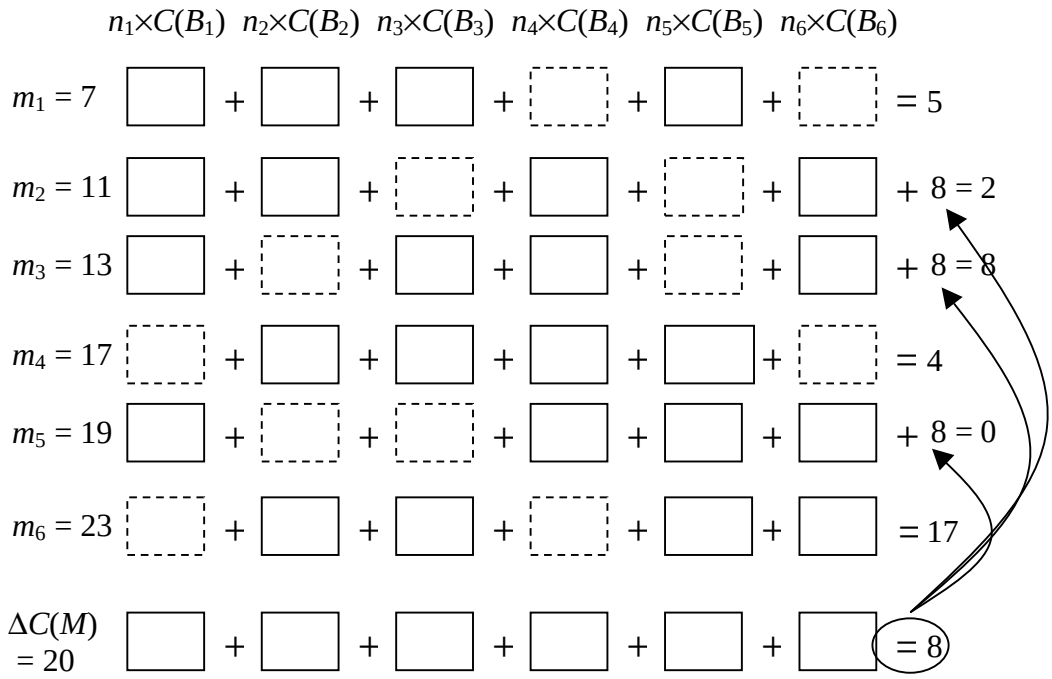

Figure 2 Example RNS scaling calculation

Each box in Figure 2 represents a ROM look-up table storing the possible multiples of $n_p \times C(B_p)$ mod m_q, where p denotes the column number, and q the row number. Note that some of the ROM's in Figure 2 could be removed because the corresponding $C(B_i)$ coefficients are 0 - these ROM's are indicated by dotted lines. The total number of MAC operations is $k\cdot(k/2 + 1) + k + k/2 = k\cdot(k+5)/2$, almost half that of Shenoy and Kumaresan's method.

2.4 Error analysis of the scaling algorithm

The error, ε, that is incurred by employing the core function to represent a scaled number rather than performing a rounded division can be estimated as follows:

$$\varepsilon_J = rnd(n/C_K(M)) - C_J(n) \qquad \text{—— (22)}$$

where ε_J is some integer. Substituting in equation (4) yields:

$$\varepsilon_J = rnd(n/C_K(M)) - (n/C_K(M) - \Sigma n_i \cdot w_i/m_i) \qquad \text{—— (23)}$$

The remainder of the division $n/C_K(M)$ is given by:

$$rem_K = n/C_K(M) - rnd(n/C_K(M)) \qquad \text{—— (24)}$$

where $-1 \le rem_K < 1$. Then,

$$\varepsilon_J = \Sigma n_i \cdot w_i/m_i - rem_K \qquad \text{—— (25)}$$

Now, the largest positive error in ε_J will occur when all the residues of moduli with positive weights are at their largest:

$$\varepsilon_{J(max+)} = \Sigma w_i^+ \cdot (m_i - 1)/m_i - rem_K \approx \Sigma w_i^+ \qquad \text{—— (26)}$$

where w_i^+ denotes the positive weights. Similarly, the largest negative error in ε_J will occur when all the residues of moduli with negative weights are at their largest:

$$\varepsilon_{J(max-)} = \Sigma w_i^- \cdot (m_i - 1)/m_i - rem_K \approx \Sigma w_i^- \qquad \text{—— (27)}$$

where w_i^- denotes the negative weights. Thus, to minimise the error in the scaled number, the core function with the smaller total magnitude weight set (i.e. smaller value of $\Sigma|w_i|$) should be chosen.. In the above example, where the weight set for $C_J(M) = 2737$ is {0, -2, 1, 0, 2, 0) and for $C_K(M) = 2717$ is {-1, 0, 0, -2, 0, 6}, that would be $C_J(M) = 2737$.

3 Ambiguity in core function extraction

A number of difficulties related to the "furriness" of the core function exist that if left unresolved may restrict the technique:

- $C_J(n)$, $C_K(n)$ or $\Delta C(n)$ may be negative for some values of n.

- $C_J(n)$ or $C_K(n)$ may exceed $C_J(M)$ or $C_K(M)$ respectively
- $\Delta C(n)$ may exceed $\Delta C(M)$.

These occurrences can give rise to difficulties because equations (14) and (21) are both computed over finite fields, so that out-of-range results alias onto in-range but erroneous results. For example, if $C_J(n)$, which is implicitly calculated mod $C_J(M)$, is negative, $C_J(n) + C_J(M)$ is incorrectly returned. Similarly, if $C_J(n) > C_J(M)$, $C_J(n) - C_J(M)$ is incorrectly returned.

3.1 Examples of ambiguity

By way of illustration of these issues, suppose an RNS has the moduli set comprised of the six smallest prime numbers: m_i = {2, 3, 5, 7, 11, 13}, giving M = 30,030. The moduli set is split into two groups, M_J = 3×5×11 = 165 and M_K = 2×7×13 = 182, to give $\Delta C(M)$ = 17. Scaling a residue number by either 165 or 182 is practical because $\Delta C(M)$ has a similar wordlength to the moduli.

The values of M_i^* and M_i^{*-1} are {15015, 10010, 6006, 4290, 2730, 2310} and {1, 2, 1, 6, 6, 3} respectively. The weight set for $C_J(M)$ = 165 is {-1, 0, 0, 3, 0, 1} and for $C_K(M)$ = 182 is {0, -2, 2, 0, 3, 0}. The two sets of $C(B_i)$ then follow from (11): $C_J(B_i)$ = {83, 110, 33, 141, 90, 38}; $C_K(B_i)$ = {91, 122, 36, 156, 99, 42}; finally, $\Delta C(B_i)$ = {8, 12, 3, 15, 9, 4}. Figure 1 (presented earlier in Section 2) is a plot of $C_J(n)$ against n calculated using (1): that is, outside the RNS, where ambiguity cannot occur.

Figure 3 is a plot of the same function but now computed modulo $C_J(M)$, in accordance with equation 14. Note the two small regions of ambiguity at either end of the x-axis. In many RNS applications, these areas could be avoided by selecting a modulus set with a greater dynamic range than that of the application.

However, in this use of the core function, ambiguity is avoided for values of $n \approx M$ (i.e. as $C(n) \approx C(M)$) because the core is being extracted effectively over modulo M, not modulo $C(M)$. Consequently, there is no ambiguity due to aliassing arising from equation (4). However, aliassing can occur for values of $n \approx 0$ (i.e. as $C(n) \approx 0$), because $C(n)$ could be negative. Two possible solutions to this are (i) select a weight set that prevents $C(n)$ from being negative; (ii) add a small bias after the scaling technique equivalent to the most negative value that $C(n)$ could take.

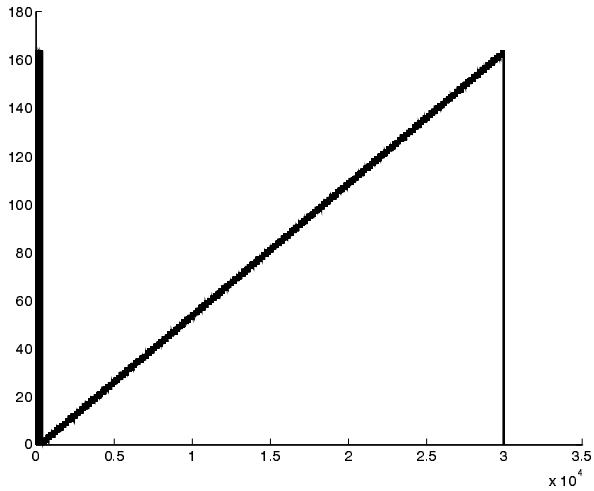

Figure 3 Plot of the core function $\langle C_J(n) \rangle_{165}$

Figure 4 is a plot of $\Delta C(n)$ for the previous example.

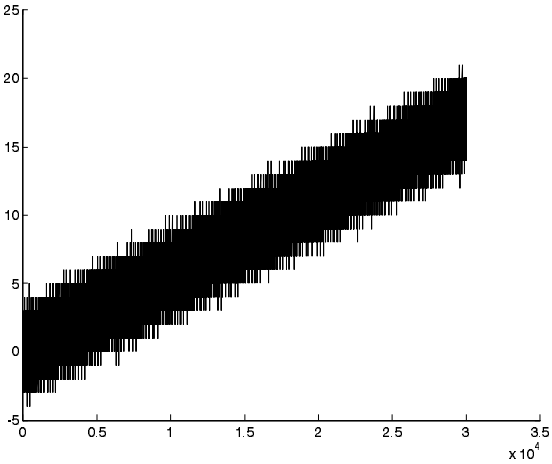

Figure 4 Plot of $\Delta C(n) = C_K(n) - C_J(n)$

Figure 5 is also a plot of $\Delta C(n)$ but computed modulo $\Delta C(M)$, as in the scaling algorithm. Note how the ambiguity region is much greater than in the core function plot. This is because the "furriness" of the plot, which is approximately given by the sums of the magnitudes of the weights [10], is a much greater proportion of the modulus. That is, $\Delta C(M) << C_J(M), C_K(M)$. The proposed scaling algorithm aims to reproduce the core function of Figure 1 within the RNS. However, the ambiguity illustrated in Figure 5 prevents this from occurring over much of the range of n. Thus, the major obstacle to RNS scaling using the core function lies in the ambiguity in computing $\Delta C(n)$.

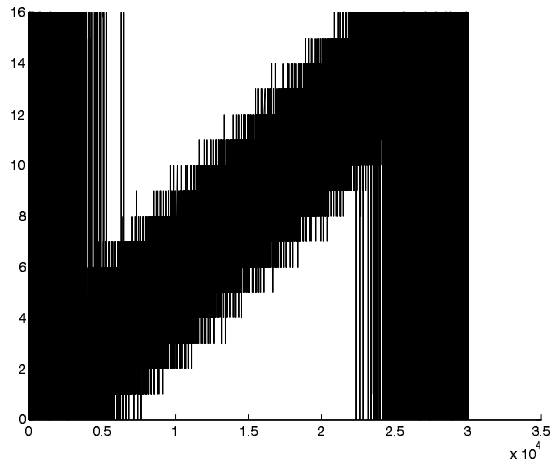

Figure 5 Plot of $\Delta C(n)$ mod $\Delta C(M)$

3.2 Removing ambiguity from core function difference calculations

In [10], two methods for overcoming ambiguity in core function computations were proposed both of which were based on retaining a parity bit at all stages of an RNS processing system. This paper proposes a similar idea for overcoming the ambiguity in computing $\Delta C(n)$, by employing the parity bit to afford calculation of $\Delta C(n)$ over the finite field $2{\cdot}\Delta C(M)$.

Equation (21), which is used to calculate $\Delta C(n)$ was given earlier as:

$$\Delta C(n) = \left\langle \sum_i n_i \cdot \Delta C(B_i) \right\rangle_{\Delta C(M)} \qquad \text{—— (21)}$$

Now, ambiguity occurs in this equation because the range of $\Delta C(n)$ exceeds $\Delta C(M)$. Earlier, a simple expression for the range of a core function, $G(M)$, was shown to be:

$$G(M) = C(M) + \sum_i |w_i| \qquad \text{—— (7)}$$

Hence, an expression for the maximum range of $\Delta C(n)$ is:

$$\Delta G(M) = C_K(M) + \sum_k |w_k| - \left(C_J(M) + \sum_j |w_j| \right) \qquad \text{—— (28)}$$
$$\approx \Delta C(M) + \sum |w_k| + \sum |w_j|$$

This follows because the two weight sets are orthogonal: for each modulus in the RNS, one of the weights in the two core functions must be zero. This implies that $\Delta C(n)$ should be calculated to a number greater than modulus $\Delta G(M)$ to avoid ambiguity.

A simple way to achieve this is to calculate $\Delta C(n)$ modulus $2{\cdot}\Delta C(M)$:

$$\Delta C(n) = \left\langle \sum_i n_i \cdot \Delta C(B_i) - R(n) \cdot \Delta C(M) \right\rangle_{2\Delta C(M)} \qquad \text{—— (29)}$$

This method avoids ambiguity provided $2{\cdot}\Delta C(M) > \Delta G(M)$, or equivalently provided $\Delta C(M) > \Sigma |w_i|$, which is readily achievable in practice. Now, equation (29) apparently requires $R(n)$ to be calculated; however, the expression can be rewritten such that only the parity of $R(n)$ is needed:

$$\Delta C(n) = \left\langle \sum_i n_i \cdot \Delta C(B_i) - \langle R(n) \rangle_2 \cdot \Delta C(M) \right\rangle_{2\Delta C(M)} \qquad \text{—— (30)}$$

The rank function, $R(n)$, is defined by the Chinese Remainder Function from (8) as:

$$R(n) = \frac{\sum_i n_i \cdot B_i - n}{M} = \sum_i \frac{n_i \cdot \left\langle M_i^{*-1} \right\rangle_{m_i}}{m_i} - \frac{n}{M} \qquad \text{—— (31)}$$

Hence, the parity of the rank function is given by:

$$\langle R(n) \rangle_2 = \left\langle \sum_i \frac{n_i \cdot \left\langle M_i^{*-1} \right\rangle_{m_i}}{m_i} - \frac{n}{M} \right\rangle_2 \qquad \text{—— (32)}$$
$$= \left\langle \sum_i n_i \cdot \left\langle M_i^{*-1} \right\rangle_{m_i} - p \right\rangle_2$$

where p denotes the parity of n, and M and hence all the moduli are assumed to be odd. (If M is even, equation (32) is undefined.) This calculation of the rank function parity can occur in parallel with the proposed scaling algorithm, so that $\Delta C(M)$ (now computed over the finite field $2{\cdot}\Delta C(M)$) appears at the same juncture in the algorithm as before. An example should help make things clearer.

3.3 Worked example of unambiguous scaling method

The number n = 6,432,750 is to be scaled by 2717 to yield ≈ 2368, again using an RNS with the moduli set, {7, 11, 13, 17, 19, 23}, but this time also using $p = n \bmod 2 = 0$. The values of M_i^{*-1} are {6, 1, 2, 12, 2, 2}, as before. In the RNS, n is represented by (2, 5, 12, 1, 15, 18). First, compute $C_J(n)$ moduli 7, 17, and 23, and $C_K(n)$ moduli 11, 13, and 19 using (15) and (17):

$C_J(n) \bmod 7 = (2{\times}1 + 5{\times}4 + 12{\times}1 + 1{\times}0 + 15{\times}1 + 18{\times}0) \bmod 7 = 0$

$C_J(n)$ mod 17 = (2×0 + 5×11 + 12×13 + 1×11 + 15×16 + 18×0) mod 17 = 3

$C_J(n)$ mod 23 = (2×0 + 5×19 + 12×7 + 1×0 + 15×12 + 18×8) mod 23 = 20

$C_K(n)$ mod 11 = (2×8 + 5×5 + 12×0 + 1×4 + 15×0 + 18×5) mod 11 = 3

$C_K(n)$ mod 13 = (2×2 + 5×0 + 12×2 + 1×7 + 15×0 + 18×2) mod 13 = 6

$C_K(n)$ mod 19 = (2×11 + 5×0 + 12×0 + 1×18 + 15×1 + 18×8) mod 19 = 9

In parallel, calculate $\langle \Delta C(n) \rangle_{\Delta 2C(M)}$ using equations (30) and (32):

$\langle R(n) \rangle_2$ = (2×6 + 5×1 + 12×2 + 1×12 + 15×2 + 18×2 – 0) mod 2 = (0×0 + 1×1 + 0×0 + 1×0 + 1×0 + 0×0 – 0) mod 2 = 1.

$\Delta C(n)$ mod 40 = (2×17 + 5×2 + 12×3 + 1×14 + 15×2 + 18×2 + 1×20) mod 40 = 20

Finally, add $\Delta C(n)$ to the $C_K(n)$ values to obtain the remaining scaled moduli:

$C_J(n)$ mod 11 = $C_K(n)$ + $\Delta C(n)$ mod 11 = 3+20 mod 11 =1

$C_J(n)$ mod 13 = $C_K(n)$ + $\Delta C(n)$ mod 13 = 6+20 mod 13 =0

$C_J(n)$ mod 19 = $C_K(n)$ + $\Delta C(n)$ mod 19=9+20 mod 19 =10

Hence, the RNS value of n = 6,432,750 (or (2, 5, 12, 1, 15, 18) in RNS format) after being scaled by 2717 is $C_J(n)$ = (0, 1, 0, 3, 10, 20). We can check the result by converting $C_J(n)$ back to decimal using the Chinese Remainder Theorem:

$C_J(n) = \Sigma n_i \times B_i \mod M$ = 0×1062347×6 + 1×676039×1 + 0×572033×2 + 3×437437×12 + 10×391391×2 + 20×323323×2 mod 7436429 = 37184511 mod 7436429 = 2366.

In the previous worked example, $\Delta C(n)$ would have been calculated as 0 because it equalled $\Delta C(M)$, thus leading to an error in the final scaled result. In order to maintain $p = \langle n \rangle_2$, only one extra XOR gate or one extra AND gate is needed for each addition or multiplication respectively in the rest of the RNS hardware. This obviously represents a tiny overhead.

4 Summary and future work

This paper has introduced a new method for scaling RNS numbers by extracting the core of an RNS number within an RNS. In fact two core functions are extracted, and the difference between them used to help provide the scaled result. A simple technique requiring the maintenance of a parity bit was shown to be effective in removing errors due to ambiguity in computing the difference between the pair of core functions.

Future work could be undertaken to try and make the scaling method more flexible: it may prove possible to scale by M_K / m_i for example. Alternatively, scaling algorithms using more than two core functions might be tried. Work exploring other uses of computing a core function within an RNS would be of interest: for example, base extension is much simpler if $C(n)$ is known. Finally, it would be of interest to use this method in a typical application (DSP, or long wordlength cryptography) to assess performance benefits.

5 References

[1] "*RNS arithmetic: Modern applications in DSP*", ed. M. Soderstrom *et al*, IEEE Press, 1986

[2] S.R. Barraclough *et al*: "The Design and Implementation of the IMS A110 Image and Signal Processor", Proc. IEEE CICC, San Diego, May 1989, pp. 24.5/1-4

[3] A.A. Hiasat and H.S. Zohdy, "Design and implementation of an RNS division algorithm", Proc. 13th IEEE Symposium on Computer Arithmetic, Asilomar CA, June 1997, pp. 240-249

[4] A.P. Preethy and D. Radhakrishnan, "A 36-bit balanced moduli MAC architecture", Proc. 42nd IEEE Midwest Symp. Circuits & Systems, Las Cruces NM, August 1999, pp. 380-383

[5] H.A. Al-Twaijry and M. J. Flynn, "Technology scaling effects in multipliers", *IEEE Trans.*, vol. 47, pp. 1201-1215 (Nov. 1998)

[6] L. Maltar *et al*, "Implementation of RNS addition and RNS multiplication into FPGAs", Proc. IEEE FCCM, Napa Valley CA, April 1998, pp. 331-332

[7] M. Re, A. Nannarelli, G.C. Cardarilli, R. Lojacono, "FPGA realization of RNS to binary signed conversion architecture", Proc. IEEE International Symposium on Circuits and Systems (ISCAS), Sydney, May 2001, pp. 350-353

[8] M.N Mahesh and M. Mehendale, "Low power realization of residue number system based FIR filters", 13th Int. Conf. on VLSI Design, Bangalore, India, January 2000, pp. 30-33

[9] M. Bhardwaj and B. Ljusanin, "The Renaissance – a residue number system based vector co-processor", Proc. 32nd Asilomar Conference on Signals, Systems & Computers, Asilomar CA, November 1998, pp. 202-207

[10] N. Burgess, "Scaled and unscaled residue number system to binary conversion techniques using the core function", Proc. 13th IEEE Symposium on Computer Arithmetic, Asilomar CA, June 1997, pp. 250-257

[11] D.D. Miller *et al*: "Analysis of the residue class core function of Akushskii, Burcev and Pak", in "*RNS arithmetic: Modern applications in DSP*", *op. cit.*

[12] F.J. Taylor and C. Huang, "An autoscale residue multiplier", *IEEE Trans. Comp.*, vol. C-31, pp. 321-325 (April 1982)

[13] G. Jullien, "Residue number scaling and other operations using ROM arrays", *IEEE Trans.*, vol. C-27, pp. 325-336 (April 1978)

[14] A. Shenoy and R. Kumaseran, "A fast and accurate RNS scaling technique for high speed signal processing", *IEEE Trans.*, vol. ASSP-37, pp. 929-937 (June 1989)

[15] F. Barsi and M.C. Pinotti, "Fast base extension and precise scaling in RNS for look-up table implementations", *IEEE Trans.*, vol. SP-43, pp.2427-2430 (Oct. 1995)

[16] Z.D. Ulman and M. Czyzak, "Highly parallel, fast scaling of numbers in nonredundant residue arithmetic", *IEEE Trans. Sig. Proc.*, vol. SP-46, pp.487-496 (Feb. 1998)