RNS Representations with Redundant Residues

(Invited Paper)

Behrooz Parhami

Dept. Electrical and Computer Enginnering
University of California
Santa Barbara, CA 93106, USA

parhami@ece.ucsb.edu

Abstract

Residue number system (RNS) representations that contain redundant moduli have been extensively studied with regard to their error checking properties. A second form of redundancy in RNS, that of redundant residues, has been applied in certain application contexts to gain speed and cost benefits. Such applications are developed in an ad hoc manner and there does not exist a theoretical framework for the latter variety of redundant RNS. We develop a theory of RNS representations with redundant residues and show how representation parameters affect the speed and complexity of various arithmetic operations. The theory parallels that of redundant signed-digit representations in that at issue are choice of 'residue sets' (akin to digit sets), encoding of residue sets, conversions between residue sets with different degrees of redundancy, including none, and algorithms for arithmetic on redundant-residue operands.

Keywords: *Carry-free addition, Computer arithmetic, Modular reduction, Number system, Parallel processing, Redundant representation, Residue number system, VLSI.*

Guide to Notation and Terminology

Note: The modulus index i is omitted when specifying how a single residue is represented and/or processed. For abbreviations used, see Table I and Example 1.

$[\bullet, \bullet)$	Half-open interval, upper endpoint excluded
$[\bullet, \bullet]$	Closed interval, both endpoints included
$\langle\bullet\rangle_m$	Residue of a value or an expression modulo m
ρ_i	Redundancy index of $[a_i, b_i]$, def. as $b_i + 1 - a_i - m_i$
a_i, b_i	Endpoints of the ith, mod-m_i, residue set $R_i = [a_i, b_i]$
h	Bit width of a standard mod-m residue $= \lceil \log_2 m \rceil$
m_i	The ith modulus of an RNS (residue number system)
M	Dynamic range of an RNS $= \prod_{i \in [0, k-1]} m_i$
M_i	Dynamic range, with m_i exluded; i.e., M/m_i
n	Smallest integer such that $2^n = 1 \bmod m$
p	Position sum of residues or pseudoresidues
t	Comparison threshold for modular reduction
x_i, y_i	Residue modulo m_i (lowercase variable)
X_i, Y_i	Pseudoresidue modulo m_i (uppercase variable)

1. Introduction

Using representational redundancy is an established method for improving the performance of arithmetic circuits. A key reason is that redundancy provides some tolerance to imprecision, so that certain decisions can be made based on local information or truncated versions of operands and partial results [Atki75]. Modern arithmetic circuits make extensive use of redundancy: in multipliers, multiply-add units, dividers, square rooters, etc. [Parh00].

Residue number system (RNS) representations have a long history that has been traced back to ancient China [Jenk93]. In modern times, Svoboda [Svob59] and Garner [Garn59] are credited with independently suggesting the usefulness of this method for faster arithmetic in digital computers. RNS uses mod-m residues in $[0, m)$, which is the standard *residue set*, or, more generally, in $[a, m + a)$, as is the case in a symmetric residue set where $a = -(m - 1)/2$ for odd m. Three kinds of redundancy can be envisaged in RNS:

1. Extra residues (moduli) beyond those needed to provide a desired dynamic range.
2. Redundant representation of conventional residues to render arithmetic on them faster.
3. Redundant residues with a range exceeding the m values in $[a, m + a)$ for some a.

The first kind has been extensively studied and is relatively well-understood (see, e.g., [Etze80]). The extra residues allow cross-checking for error detection or correction using standard base extension methods. The second kind, though viable, has not been applied in practice, primarily because residues are often small numbers that are easily handled by means of conventional arithmetic circuits or table lookup. Our focus in this paper is redundant RNS of the third kind. This kind of redundancy has been used in many different, and seemingly unrelated, contexts and has never been explicitly recognized as a type of redundant representation. We seek to remedy this shortcoming so that results in this area can be related to, and benefit from, work on other classes of redundant representations.

0-7803-7147-X/01/$10.00©2001 IEEE

2. Residue Number Systems

An RNS is characterized by a set of k pairwise relatively prime moduli $m_{k-1} > \ldots > m_1 > m_0$ and a residue set R_i for the ith modulus, typically chosen to be $[0, m_i)$. Clearly, at most one of the moduli can be even and that one is usually taken to be a power of 2 to simplify the associated circuitry; all other moduli are odd. Roughly speaking, the set of moduli corresponds to the choice of a radix and the residue set to the digit set of conventional positional number representations. Just as the digit set of nonredundant radix-r system can be chosen to be $[a, r + a)$, $a < 0$, the residue set R_i associated with m_i can be selected to be $[a_i, m_i + a_i)$.

Such nonredundant RNS have been extensively studied and used, primarily, in signal processing applications. For a concise treatment of residue number systems and issues in implementing RNS arithmetic, see Chap. 4 in [Parh00]. State of the art in RNS theory and applications, circa the mid 1980s, is provided by [Sode86]. Some application details, including examples of custom chip implementations can be found in [Jenk93]. After a slowdown in reporting of research results on RNS, renewed interest is evident due to a combination of emerging, more efficient, algorithms for difficult operations such as division [Hung94] and potential power economy of RNS arithmetic [Frek97]. In what follows, we review only elementary properties of RNS to the extent needed for understanding the role of redundancy, as explained in the rest of this paper.

An RNS may have a handful of large moduli, typically chosen to be integers of the form 2^h or $2^h \pm 1$, or a larger number of small moduli, often chosen to be primes or powers of primes. The former category of RNS corresponds to very-high-radix number representation systems, in which digit manipulation circuits are quite complex but there are fewer such circuits, while the latter are akin to moderartely high radices such as 16 or 64. Issues in choosing the moduli, and the associated cost/speed tradeoffs, are beyond the scope of this paper. However, we emphasize that the techniques discussed here are applicable to either category of RNS, regardless of the choice of the moduli and the use of signed or unsigned residues.

The primary advantage of RNS is that addition, subtraction, and multiplication can be performed independently and in parallel on the various residues. When the residues are small, this results in very high speed, particularly for multiplication which is slow/expensive with conventional number representation. Figure 1 depicts two ways of implementing a residue adder for a modulus m. The simpler version (Fig. 1a), which is often described in the literature, adds two h-bit residues, x and y, to form their sum $u + 2^h c$. If either c or the carry-out d of the addition $u + 2^h - m$ is 1 (the latter implying, for $c = 0$, that $x + y \geq m$), then the output is $v = x + y - m$ instead of $x + y$. The somewhat faster implementation in Fig. 1b forms v directly through a carry-save addition and an ordinary addition.

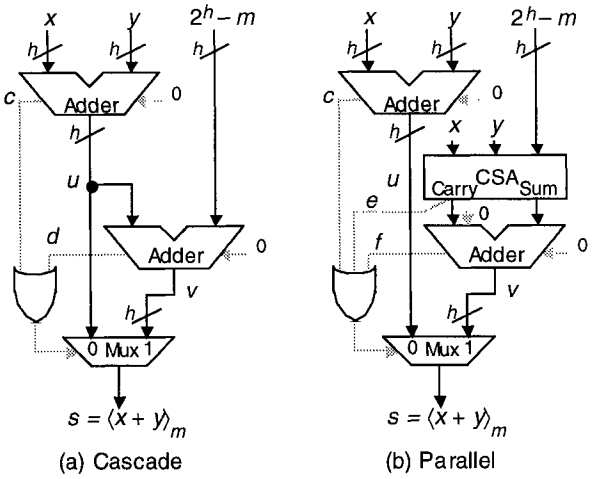

Fig. 1. Two mod-m residue adders.

A residue multiplier is somewhat more complex, but still considerably simpler and faster than an ordinary binary multiplier. A bit-at-a-time implementation would use a modular adder to derive the h-bit mod-m product in h cycles. It is also possible to use an $h \times h$ combinational (tree or array) multiplier, followed by a mod-m reduction circuit for the $2h$-bit product. Alternatively, modular reduction can be fused with combining of partial products for speed/economy, particularly if one takes advantage of the fact that some bit patterns are unused [Pali01].

Here, we say nothing about implementation of difficult RNS operations such as sign test, overflow detection, magnitude comparison, or general division [Parh00]. The methods to be discussed do not cure such difficulties but are meant as mechanisms for making the already efficient RNS operations even simpler and more efficient. To the extent that the difficult operations use the simpler ones as building blocks, the added efficiency benefits those as well.

3. Redundant Residues or Pseudoresidues

Consider a modulus m satisfying $2^{h-1} < m \leq 2^h$. Conventional binary residues associated with this modulus are h-bit unsigned numbers in the range $[0, m)$; we refer to these as *single-range unsigned* (SRU) residues. Similarly, *single-range signed* (SRS) residues have a symmetric or almost symmetric range $[-\lfloor m/2 \rfloor, \lceil m/2 \rceil)$, depending on whether m is odd or even. In both of these cases, the residues are nonredundant in the sense that each mod-m equivalence class has a single representative in the residue set.

In a manner similar to the use of digit sets with more than r values for radix-r positional representations, we can envisage the use of more than m values in a residue set. Using multiple representatives from some or all mod-m equivalence classes does not cause any problem in arithmetic because all operations are modulo m.

We refer to residues from redundant sets as *pseudoresidues* and to the resulting RNS as *redundant-residue RNS*. This is to distinguish our systems from *redundant-modulus RNS* in which extra moduli, beyond the minimum set required for providing a desired dynamic range, are employed for the sake of error detection and/or correction.

Table I shows the nonredundant SRU and SRS residue sets along with several examples of redundant residue sets. The *redundancy index* associated with a modulus m having pseudoresidues in $[a, b]$, or $[a, b + 1)$, is defined as:

$$\rho = b + 1 - a - m$$

The two single-width redundant residue sets use the entire range of values offered by an h-bit representation as valid pseudoresidues, in unsigned (SWU) or 2's-complement signed (SWS) format. Doublerange pseudoresidues can assume twice as many values as a nonredundant residue, again in unsigned (DRU) or signed (DRS) form. A useful encoding of DRU is the carry-save format in which the pseudoresidue is the sum of two h-bit unsigned numbers that are separately represented. Squared- or quadratic-range unsigned (QRU) residues are capable of holding the product of two SRU residues and have thus been used in inner-product computations of the type found in digital filters. Doublewidth unsigned residues can be used in lieu of QRU for implementation simplicity. End-around residue sets allow an overflow of weight 2^n to be reinserted at the least-significant position, given the property $2^n = 1 \bmod m$. For example, mod-9 residues can be accumulated by using 6-bit pseudoresidues with end-around carry ($2^6 = 1 \bmod 9$).

Table I. Some choices for mod-m residue set.

Code	Type of residue	Residue set	ρ
SRU	Single-range unsigned	$[0, m)$	0
SRS	Single-range signed	$[-\lfloor m/2 \rfloor, \lceil m/2 \rceil)$	0
SWU	Single-width unsigned	$[0, 2^h)$	$2^h - m$
SWS	Single-width signed	$[-2^{h-1}, 2^{h-1})$	$2^h - m$
DRU	Doublerange unsigned	$[0, 2m)$	m
DRU_{cs}	DRU, carry-save variant	$[0, 2m - 2]$	$m - 1$
DRS	Doublerange signed	$[-m, m)$	m
QRU	Squared-range unsigned	$[0, (m-1)^2]$	$m^2 - 3m + 2$
DWU	Doublewidth unsigned	$[0, 2^{2h})$	$2^{2h} - m$
EAU	End-around unsigned	$[0, 2^n)$	$2^n - m$

Such redundant residue sets have been used in an ad hoc fashion as tools for speeding up or simplifying circuit designs [Bhar98], [Burg98], [Ibra94], [Koc98], [Parh96], [Pies94], [Pour97]. They are not explicitly recognized as redundant residues and, thus, have not enjoyed a uniform treatment. Our contribution is to derive a general theory of redundant-residue RNS representations that allows us to obtain such implementation strategies as special cases, thus facilitiating design tradeoffs and fostering the portability of techniques and results among different application contexts.

4. Addition with Pseudoresidues

Two kinds of 2-operand addition involving pseudoresidues might be envisaged:

Pseudoresidue + Residue → Pseudoresidue

Pseudoresidue + Pseudoresidue → Pseudoresidue

The first kind is akin to carry-save addition in that it might be used to accumulate the sum of a set of numbers (represented with ordinary residues) while taking advantage of speed/ease of arithmetic with pseudoresidues; the second kind is of the same flavor as adding redundant operands in (generalized) signed-digit arithmetic. We will take ordinary residues to lie in $[0, m)$; similar results can be developed for symmetric or arbitrary nonredundant residue sets.

Symbolically, we write the first type of modulo-m addition as the following combination of residue sets:

$$[a, a + m + \rho) + [0, m) \rightarrow [a, a + m + \rho)$$

Because the sum on the left lies in $[a, a + 2m + \rho - 1)$, the addition process requires that the raw sum be reduced by m whenever it is greater than $a + m + \rho - 1$ (or $\geq a + m + \rho$). In fact, the comparison threshold t_{-m} for determining if the position sum p should be reduced by m via testing $p \geq ? t_{-m}$, can lie anywhere in $[a + m, a + m + \rho]$. Whenever this range includes a power of 2, the comparison can be simplified by choosing it as t_{-m}. Here are some examples:

DRU + SRU → DRU, $t_{-m} = 2^h$

DRU_{CS} + SRU → DRU_{CS}, $t_{-m} = 2^h$

SWS + SRU → SWS, $t_{-m} = 2^{h-1}$

SWU + SRU → SWU, $t_{-m} = 2^h$

The adder hardware for the last example above is depicted in Fig. 2a, where the subtraction of m is performed by discarding the carry c and adding $c(2^h - m)$; this latter term is formed by fanning out the signal c into all positions where $2^h - m$ has 1s. A faster version of this adder can be developed in a manner similar to Fig. 1b.

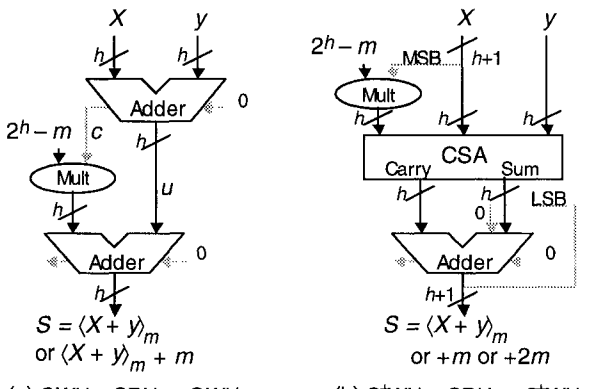

Fig. 2. Pseudoresidue/residue addition.

More importantly, when ρ is large enough, the comparison threshold can be chosen as $t_{-m} = c + y$ for some constant c. If $c + y$ is in $[a + m, a + m + \rho]$ for all y, one can compare X to c instead of comparing $p = X + y$ to $c + y$. This results in significant circuit simplification and speedup and leads us to a novel redundant-residue representation.

Example 1: Consider $(h + 1)$-bit residues in $[0, 2^h + m)$, with $\rho = 2^h$. Because this representation requires one more bit than SWU, we call it single$^+$-width unsigned (S$^+$WU). Addition of such pseudoresidues to nonredundant residues in $[0, m)$ is quite efficient (Fig. 2b). Per our results above, the comparison threshold must be in $[0, 2^h + m]$ and thus can be chosen as $2^h + y$, or 2^h before the addition of y. ■

The second type of mod-m addition may involve two or three different residue sets. For brevity, we only discuss the case of uniform residue sets for the operands/sum in some detail, providing just one example of the more general case. Symbolically, this type of modulo-m addition can be written as the following combination of residue sets:

$$[a, a + m + \rho) + [a, a + m + \rho) \rightarrow [a, a + m + \rho)$$

The position sum on the left lies in $[2a, 2a + 2m + 2\rho - 1)$. Reducing this sum mod m may require several operations that are best dealt with by considering various cases.

Case 1 ($a = 0$): In this case, which covers a majority of practically useful examples listed in Table I, the position sum is in $[0, 2m + 2\rho - 1)$ and must be reduced to lie in the original residue set $[0, m + \rho)$. The excess magnitude over the maximum pseudoresidue can be as high as $m + \rho - 1$. If $\rho = 1$, at most m needs to be subtracted from the position sum, with the comparison constant being $t_{-m} = m$ or $m + 1$. This may occur, e.g., for SWU with $m = 2^h - 1$, for which the comparison constant $t_{-m} = 2^h$ would be a natural choice. In this case, the redundancy is too low to allow lookahead in determining the subtracted value. For $1 < \rho \leq m + 1$, which covers all but the last three examples in Table I, the amount to subtract is 0, m, or $2m$, with comparison constant ranges being $[m, m + \rho]$ for t_{-m} and $[2m, 2m + \rho]$ for t_{-2m}. Here are a couple of examples, assuming $m \leq 3 \times 2^{h-2}$:

$$\text{DRU} + \text{DRU} \rightarrow \text{DRU},\ t_{-m} = 2^h,\ t_{-2m} = 3 \times 2^{h-1}$$

$$\text{SWU} + \text{SWU} \rightarrow \text{SWU},\ t_{-m} = 2^h,\ t_{-2m} = 3 \times 2^{h-1}$$

In either example, two MSBs of the position sum are used for deciding what value to subtract. The SWU + SWU case is shown in Fig. 3a. The scheme for DRU + DRU is quite similar to the above. Again, when ρ is large enough, the computation becomes simpler and faster.

Example 2: Consider the $(h + 1)$-bit S$^+$WU pseudoresidues introduced in Example 1. Addition of such numbers is quite efficient (Fig. 3b). The comparison thresholds t_{-m} and t_{-2m} must be in $[m, 2^h + m]$ and $[2m, 2^h + 2m]$, respectively. Thus, the subtraction decision can be made based only on the MSBs of X and Y. The small box to the right of the multiplexor produces the "enable" and "select" signals by ORing and ANDing its two inputs, respectively. ■

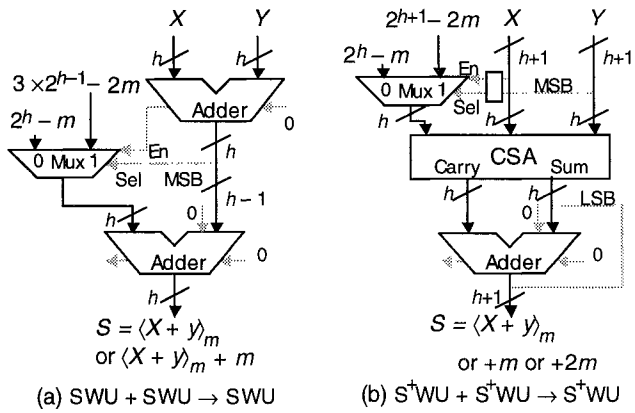

Fig. 3. Pseudoresidue/pseudoresidue addition.

Case 2 ($-m \leq a < 0$): In this case, the position sum range $[2a, 2a + 2m + 2\rho - 1)$ includes values that are too small and thus must be adjusted upward. In the following examples, the value m is added when the position sum is less than t_{+m}:

$$\text{DRS} + \text{DRS} \rightarrow \text{DRS},\ t_{-m} = 2^{h+1},\ t_{+m} = 0$$

$$\text{SWS} + \text{SWS} \rightarrow \text{SWS},\ t_{-m} = 2^{h-1},\ t_{+m} = -2^{h-1}$$

Hardware realization of the first example above is quite similar to Fig. 3b, except that the mux inputs are $2^h - m$ (0) and m (1), the mux "enable" signal is obtained as the XNOR of the MSBs (sign bits), and the "select" signal is tied to either sign bit; thus, when both inputs are positive (negative), $2^h - m$ (m) is added to the position sum.

Other cases: Cases with greater redundancy (smaller values of a or larger values of ρ) can be handled similarly.

As an example of the second type of mod-m addition with nonuniform residue sets, consider

$$\text{DWU} + \text{QRU} \rightarrow \text{DWU}$$

This might be useful for inner product computations that are of interest, for example, in digital filters [Ibra94], [Parh96]. The QRU component might be the $2h$-bit output of an $h \times h$ multiplier, with the sum accumulated in DWU form. When the carry-out of the addition on the left-hand side is 1 (worth 2^{2h}), the position sum can be reduced by $2^h m$, requiring an h-bit adder. The result is guaranteed to be nonnegative and no greater than $2^{2h} - 1 + (m - 1)^2 - 2^h m$. Hence, the result will be in $[0, 2^{2h})$.

One can also envisage

$$\text{DWU} + \text{DWU} + \text{QRU} \rightarrow \text{DWU} + \text{DWU}$$

where the notation means that a doublewidth unsigned pseudoresidue in carry-save form (DWU + DWU) is added to a QRU pseudoresidue, with the modular sum produced in carry-save form. The MSBs of the three inputs can be used to predict whether a carry-out will be produced; if so, then $2^h(2^h - m)$ is added to the inputs in a 4-to-2 carry-save adder tree with no carry propagation.

5. Pseudoresidue Conversions

Converting from a source residue set $[a, b]$ to a target set $[a', b']$ involves the addition or subtraction of suitable multiples of m. If the target residue set is redundant, there is room for error and the multiple can be chosen based on examining a subset of the bits in the input. Otherwise, full precision is required, as in all redundant to nonredundant conversions. Here are examples with redundant target sets:

1. SWU → DRU
2. DRU_{cs} → SWU
3. QRU → DRU
4. DWU → DRU

The first of these requires no conversion at all. The second conversion can be performed by using methods similar to those depicted in Figs. 1-3. One possibility for the last two conversions is to reduce the upper and lower h bits of the input modulo m via table lookup, with the residues then added to form an $(h + 1)$-bit value in $[0, 2m - 2]$ or used directly if DRU_{cs} is the desired format.

If the target residue set is nonredundant (e.g., SRU or SRS), then a true modular reduction is required. For low-redundancy source residue sets, circuits similar to that at the lower half of Fig. 1a might do. The greater the redundancy of the source residue set, the more difficult the conversion to a nonredundant residue set. For tabular realization, a stepwise refinement approcah might help keep the size of the tables in check [Parh97].

6. Multiplication with Pseudoresidues

One may anticipate a need for multiplying a pseudoresidue by another pseudoresidue (same or different type) or by an ordinary residue. When table lookup methods are used, low-redundancy pseudoresidues create no special difficulty, except perhaps to make the tables somewhat larger. With high redundancy pseudo residues or when the moduli are large, circuit techniques are called for.

As in ordinary modular multipliers, the required modular reduction operations can be interlaced with partial-product accumulation steps to keep the intermediate results from growing in width. Because of redundancy in intermediate and final values, modular reduction steps are simplified. Implementation details are being worked out.

7. Conclusion

Unifying theories are desirable because they lead to better understanding of existing techniques and also pave the way for new developments and porting of methods from one application domain to another. We have sown the seeds of a unifying theory of redundant-residue RNS representations. Further work may focus on the detailed specification of arithmetic operations other than addition, cataloging of other useful instances of redundant-residue systems, and quantifying the speed/cost/power benefits of various forms of redundancy and residue-set encodings.

References

[Atki75] Atkins, D.E., "An Introduction to the Role of Redundancy in Computer Arithmetic," *IEEE Computer*, Vol. 8, pp. 74-76, June 1975.

[Bhar98] Bhardwaj, M. and B. Ljusanin, "The Renaissance – A Residue Number System Based Vector Co-Processor for DSP Dominated Embedded ASICs," *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 202-207, Nov. 1998.

[Burg98] Burgess, N., "Efficient RNS to Binary Conversion Using High-Radix SRT Division," *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 1240-1243, Nov. 1998.

[Etze80] Etzel, M.H. and W.K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 28, pp. 538-545, Oct. 1980.

[Frek97] Freking, W.L. and K.K. Parhi, "Low-Power FIR Digital Filters Using Residue Arithmetic," *Proc. 31st Asilomar Conf. Signals, Systems and Computers*, pp. 739-743, Nov. 1997.

[Garn59] Garner, H., "The Residue Number System," *IEEE Trans. Electronic Computers*, Vol. 8, pp. 140-147, June 1959.

[Hung94] Hung, C.Y. and B. Parhami, "An Approximate Sign Detection Method for Residue Numbers and Its Application to RNS Division," *Computers & Mathematics with Applications*, Vol. 27, pp. 23-35, 1994.

[Ibra94] Ibrahim, M.K., "Novel Digital Filter Implementations Using Hybrid RNS-Binary Arithmetic," *Signal Processing*, Vol. 40, pp. 287-294, 1994.

[Jenk93] Jenkins, W.K., "Finite Arithmetic Concepts," Chap. 9 in *Handbook for Digital Signal Processing*, ed. by S.K. Mitra and J.F. Kaiser, Wiley, 1993, pp. 611-675.

[Koc98] Koc, C.K. and C.Y. Hung, "Fast Algorithms for Modular Reduction," *IEE Proc. – Computers and Digital Techniques*, Vol. 145, No. 4, pp. 265-271, July 1998.

[Pali01] Paliouras, V., K. Karagianni, and T. Stouraitis, "A Low-Complexity Combinatorial RNS Multiplier," *IEEE Trans. Circuits and Systems II*, Vol. 48, pp. 675-683, July 2001.

[Parh94] Parhami, B. and C.Y. Hung, "Optimal Table Lookup Schemes for VLSI Implementation of Input/Output Conversions and Other Residue Number Operations," *VLSI Signal Processing VII* (Proc. of IEEE Workshop), pp. 470-481, Oct. 1994.

[Parh96] Parhami, B., "A Note on Digital Filter Implementation Using Hybrid RNS-Binary Arithmetic," *Signal Processing*, Vol. 51, pp. 65-67, 1996.

[Parh97] Parhami, B., "Modular Reduction by Multi-Level Table Lookup," *Proc. 40th Midwest Symp. Circuits and Systems*, pp. 381-384, Aug. 1997.

[Parh00] Parhami, B., *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2000.

[Pies94] Piestrak, S.J., "Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders," *IEEE Trans. Computers*, Vol. 43, pp. 68-77, Jan. 1994.

[Pour97] Pourbigharaz, F. and H.M. Yassine, "A Signed-Digit Architecture for Residue to Binary Conversion," *IEEE Trans. Computers*, Vol. 46, pp. 1146-1150, Oct. 1997.

[Sode86] Soderstrand, M.A., W.K. Jenkins, G.A. Jullien, and F.J. Taylor (Eds.), *Residue Number System Arithmetic*, IEEE Press, 1986.

[Svob59] Svoboda, A., "The Numerical System of Residual Classes in Mathematical Machines," in *Information Processing* (Proc. UNESCO Conf., 1959), pp. 419-422, 1960.