

Arithmetic Circuits Combining Residue and Signed-Digit Representations

Anders Lindström¹, Michael Nordseth¹, Lars Bengtsson¹, and Amos Omondi²

¹ Department of Computer Engineering, Chalmers University of Technology, Sweden
{e8call, e8mn}@etek.chalmers.se, labe@ce.chalmers.se

² School of Informatics and Engineering, Flinders University, Australia
amos@infoeng.flinders.edu.au

Abstract. This paper discusses the use of signed-digit representations in the implementation of fast and efficient residue-arithmetic units. Improvements to existing signed-digit modulo adders and multipliers are suggested and new converters for the residue signed-digit number system are described for the moduli $\{2^n - 1, 2^n, 2^n + 1\}$. By extending an existing efficient signed-digit adder design to handle modulo operations, we are able to implement high performance modulo addition. The hardware complexity of signed-digit modulo multipliers is reduced by using a more efficient algorithm for calculating partial products. Finally, the novel converters presented makes it possible to integrate this residue signed-digit number system with conventional binary circuits.

1 Introduction

The residue number system (RNS) is often used when high speed computation is needed and where the use of the conventional binary system would limit the speed of the arithmetic circuits due to carry propagation. RNS divides an integer into a number of smaller integers (i.e. with a shorter binary representation) that can be processed in parallel independently of each other and thus reducing carry propagation. However, the remaining carry propagation can still be a limiting factor in real-time applications.

The use of the signed-digit (SD) system has recently been suggested as a way of eliminating the remaining carry propagation in RNS arithmetic [1][2]. The SD representation is a redundant number system and therefore facilitates carry-free addition [3], SD numbers can be added in constant time independent of operand-widths. Combining the carry-free properties of SD with RNS arithmetic helps simplify the implementation of crucial RNS arithmetic operations. Our work makes two main contributions: the improvements of circuits that have previously been proposed for residue signed-digit (RNS+SD) arithmetic; and the design of novel converters for the RNS+SD representation.

The improvements on existing work consist of designs for faster and smaller SD modulo adders than are currently known and in improvements, by reduction of the number of partial products, in the modulo multiplication algorithm (and corresponding

implementations). In addition, we present completely new designs for RNS+SD converters based on the moduli-set $\{2^n - 1, 2^n, 2^n + 1\}$.

The rest of the paper consists of four sections, as follows. Section 2 gives a brief overview of the mathematics and algorithms used in the RNS and SD representations. Section 3 presents the improved arithmetic units and the new converters, and Section 4 discuss the results obtained from implementing these circuits in VLSI. The last section is a concluding summa.

2 Background

2.1 The Residue Number System

A residue number system is defined by a set of relative prime numbers, $\{m_1, m_2, \dots, m_r\}$, called the moduli. In such a system, an integer X is represented by an ordered set of r residues, $\{x_1, x_2, \dots, x_r\}$, where $x_i = X \bmod m_i$. If only positive numbers are permitted, then any integer in the range $[0, M)$, where $M = m_1 \cdot m_2 \cdot \dots \cdot m_r$, can be uniquely represented. If negative numbers are also allowed, then it is usual to let the dynamic range be $[-M/2, M/2)$.

The choice of moduli is crucial to the representational efficiency and to the complexity and delay of the arithmetic unit. The moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ is used throughout this paper. This is a popular moduli-set, as the restriction to powers of two (± 1) in the set makes it relatively easy to implement efficient arithmetic units and to produce generalized designs that are parameterized by operand word length.

2.2 The Signed Digit Number System

The radix-2 signed-digit number system has the digit-set $\{\bar{1}, 0, 1\}$, where $\bar{1}$ denotes -1 . An n-digit SD integer $Y = [y_{n-1} \dots y_0]_{SD}$, $y_i \in \{\bar{1}, 0, 1\}$, has the value

$$\sum_{i=0}^{n-1} y_i \times 2^i$$

which is the same as for an unsigned binary integer except that y_i also can be -1 . This makes it possible to represent an integer in more than one way. For example, the integer "6" can be represented as $[0110]_{SD}$, $[1\bar{1}10]_{SD}$, or $[10\bar{1}0]_{SD}$. Zero is, however, uniquely represented.

The binary representation of an SD digit, y_i , requires two binary bits, y_i^- and y_i^+ . The binary coding used in this paper consists of $[00]$, $[01]$ and $[10]$ to represent 0 , 1 and $\bar{1}$ respectively. That is, $y_i = [y_i^- y_i^+]$, where $Y^+ = [y_{n-1}^+ \dots y_0^+]$ and $Y^- = [y_{n-1}^- \dots y_0^-]$ are binary n-bit vectors. With this coding, the value of an n-digit SD integer, Y , can be written as

$$\sum_{i=0}^{n-1} y_i^+ \times 2^i - \sum_{i=0}^{n-1} y_i^- \times 2^i = Y^+ - Y^- . \quad (1)$$

The rules for adding two SD integers, X and Y , are shown in Table 1, in which c_i denotes the carry and u_i the interim sum.

Table 1. Rules for adding SD numbers

$x_i y_i$	00	01	0 $\bar{1}$	$0\bar{1}$	11	$\bar{1}\bar{1}$	$1\bar{1}$
$x_{i-1} y_{i-1}$	–	neither is $\bar{1}$	at least one is $\bar{1}$	neither is $\bar{1}$	at least one is $\bar{1}$	–	–
c_{i+1}	0	1	0	0	$\bar{1}$	1	0
u_i	0	$\bar{1}$	1	$\bar{1}$	1	0	0

The facts that $[01]_{SD} = [1\bar{1}]_{SD}$ and $[0\bar{1}]_{SD} = [\bar{1}1]_{SD}$ are used in the rules above to avoid any carry propagation when the final sum is calculated, $s_i = c_i + u_i$, $s_i \in \{\bar{1}, 0, 1\}$. Below is an example of adding two SD integers with the rules in Table 1.

$$\begin{array}{r}
 100\bar{1}0 \quad (14) \\
 + \quad \bar{1}1\bar{1}11 \quad (-9) \\
 \hline
 00\bar{1}01 \quad c_i \\
 + \quad 0110\bar{1} \quad u_i \\
 \hline
 s_i \quad 00011\bar{1} \quad (5)
 \end{array}$$

Note that since the SD system can represent a negative integer without any special sign digit, it is possible to represent any integer and its negation with an equal number of digits (unlike the ordinary binary system).

The negation of an integer is a very simple operation in the SD system, $y_i = [y_i^- y_i^+]$ becomes $(-y)_i = [y_i^+ y_i^-]$ when negated as can be seen by negating Eq. (1).

3 Arithmetic Units

In the first and second parts of this section, we suggest improvements to existing SD modulo adders and multipliers. We show that speed and area gains can be made by taking a known highly efficient SD addition cell and adapting it for residue operation. We also show that the area gains can be further improved in the modulo multipliers by using a better multiplication algorithm.

In the third and fourth part of this section, we present novel converters for the RNS+SD $\{2^n - 1, 2^n, 2^n + 1\}$ representation. Although there has been prior work on arithmetic units that combine RNS and SD notations, none of that work has dealt with

the important issue of conversion between RNS+SD and conventional binary. One of the main contributions of this paper is the design of appropriate converters.

3.1 SD Modulo Addition

An implementation of the SD adder cell circuit description in [4] is used as a foundation for the presented SD modulo adder. We also considered the adder cells suggested in [5] and [6], but, when implemented and synthesized, neither had better delay, area nor power performance than the one chosen. Eq. set 2 describes the logic for this SD adder cell.

$$\begin{aligned}
 p_i &= \overline{x_{i-1}^- + y_{i-1}^-} \\
 xid_i &= x_i^- + x_i^+ \\
 yid_i &= y_i^- + y_i^+ \\
 z_{i+1} &= \overline{xid_i \cdot \overline{yid_i} \cdot p_i + \overline{xid_i} \cdot yid_i \cdot p_i + x_i^+ \cdot yid_i + y_i^+ \cdot xid_i} \\
 t_i &= xid_i \cdot \overline{yid_i} \cdot p_i + \overline{xid_i} \cdot yid_i \cdot p_i + xid_i \cdot yid_i \cdot \overline{p_i} + \overline{xid_i} \cdot \overline{yid_i} \cdot \overline{p_i} \\
 s_i^- &= t_i \cdot z_i \\
 s_i^+ &= \overline{t_i} \cdot \overline{z_i}
 \end{aligned}$$

Eq. set 2. SD addition

Our SD modulo adders are based on the end-around-carry logic for SD adders described in [1] and [2], but we will now apply this basic concept to the more efficient SD adder described above.

Modulo 2^{n+1} addition (where n is the word length) can be performed by including an end-around-carry in the computations described by Eq. Set 2. This inclusion is described in Eq. set 3.

$$\begin{aligned}
 p_0 &= \overline{x_{n-1}^+ + y_{n-1}^+} \\
 z_0 &= \overline{xid_{n-1} \cdot \overline{yid_{n-1}} \cdot \overline{p_{n-1}} + \overline{xid_{n-1}} \cdot yid_{n-1} \cdot \overline{p_{n-1}} + x_{n-1}^- \cdot yid_{n-1} + y_{n-1}^- \cdot xid_{n-1}}
 \end{aligned}$$

Eq. set 3. 2^{n+1} end around carry

For modulo 2^n-1 addition, the end-around operation is much simpler:

$$\begin{aligned}
 p_0 &= \overline{x_{n-1}^- + y_{n-1}^-} \\
 z_0 &= z_n.
 \end{aligned}$$

This results in modulo adders that are just a few gates larger than the normal SD adder but which are equally fast.

The use of a more efficient SD adder cell makes the SD modulo adders presented here faster and smaller than the proposed circuit implementation described in [2] as the

SD adder cell in our modulo adders only uses one bit for the carry (z_i) compared to the two bit solution in [2].

In contrast with binary residue adders, the carry-free nature of SD addition means that there is no carry propagation involved in the addition of the end-around-carry. In the specific case of a modulo $2^n + 1$ adder, the SD counterpart of the binary residue adder also has the advantage that it only needs n -digits to represent the result. If the result happens to be greater than $2^n - 1$ then the result will instead be taken from the negative range, for example

$$\langle [110] + [010] \rangle_9 = [1000]_9 = [00\bar{1}]_{SD} = -1 = 8 - 9$$

where $\langle X \rangle_m$ is the operation $X \bmod m$. In fact, the negative range will be used whenever it is suitable.

The hardware logic needed for an SD adder can be dramatically simplified if the input to the adder is in binary representation. The rules in Table 1 can in this case be simplified to those shown in Table 2.

Table 2. Simplified rules for binary input

$x_i y_i$	00	01	10	11
c_{i+1}	0	1	1	1
u_i	0	$\bar{1}$	$\bar{1}$	0

It is also possible to reduce the complexity of the SD adder even if just one of the inputs to the SD adder is in binary representation. Table 1 may be simplified to Table 3 in this case.

Table 3. Rules for mixed input

$x_i y_i$	00	01	10	11	$\bar{1}0$	$\bar{1}1$
c_{i+1}	0	1	1	1	0	0
u_i	0	$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0

The translation of Table 2 and Table 3 into logic is straightforward and can be used directly for modulo $2^n - 1$ addition by using a simple end-around-carry, $c_0 = c_n$. However, this is not feasible in the case of modulo $2^n + 1$ addition where instead Eq. set 2 and Eq. set 3 can be simplified for binary or mixed input. SD modulo subtraction with one or both inputs in binary representation can be simplified in a similar way.

3.2 SD Modulo Multiplication

We have improved on existing implementations of RNS+SD multiplication by presenting a simple way to reduce the number of partial products to be added, and, therefore,

the logical complexity of the multiplier. Such a reduction has negligible effect on the operational time of the multiplier.

Modulo m multiplication is defined as

$$\langle x \cdot y \rangle_m = \left\langle \sum_{i=0}^{n-1} y_i \langle 2^i \cdot x \rangle_m \right\rangle_m$$

where n is the word length. The partial products (i.e. $y_i \langle 2^i \cdot x \rangle_m$) can be summed in a tree of SD modulo adders as described in [1] and [2].

Conventional schemes for reducing the number of partial products cannot be used here as both x and y are in SD format. However, we will now show that SD arithmetic allows the number of partial products to be reduced, without the complex multiplier-recoding that is normally required in conventional schemes.

The number of partial products can be reduced to $n/2$ by considering two digits of y simultaneously as follows

$$\langle x \cdot y \rangle_m = \left\langle \sum_{i=0}^{\frac{n}{2}-1} \langle 2^{2i} (y_{2i} \cdot x + y_{2i+1} \cdot 2x) \rangle_m \right\rangle_m = \left\langle \sum_{i=0}^{\frac{n}{2}-1} \langle 2^{2i} \cdot rp_i \rangle_m \right\rangle_m = \left\langle \sum_{i=0}^{\frac{n}{2}-1} \langle pp_i \rangle_m \right\rangle_m$$

where the value of rp_i are as shown in Table 4. These values can be generated using only one SD modulo adder (to generate $3x = 2x + x$) as shift, rotation (see Eq. set 4) and negation does not require any additional gates.

Table 4. Possible values of rp_i

y_{2i+1} y_{2i}	0	1	$\bar{1}$
0	0	$2x$	$-2x$
1	x	$3x$	$-x$
$\bar{1}$	$-x$	x	$-3x$

The partial products, $pp_i = \langle 2^{2i} \cdot rp_i \rangle_m$, can be generated by rotation if the moduli is $2^n - 1$, 2^n or $2^n + 1$ by using the rules in Eq. Set 4. These rotations can be accomplished by wiring connections appropriately.

$$\begin{aligned} \langle 2^a \cdot y \rangle_{2^p-1} &= [y_{p-1-a} \dots y_0 y_{p-1} \dots y_{p-a}] \\ \langle 2^a \cdot y \rangle_{2^p} &= [y_{p-1-a} \dots y_0 0_{p-1} \dots 0_{p-a}] \\ \langle 2^a \cdot y \rangle_{2^p+1} &= [y_{p-1-a} \dots y_0 (-y_{p-1}) \dots (-y_{p-a})] \\ p &\geq a, y = [y_{p-1} \dots y_0] \end{aligned}$$

Eq. set 4. Rotation

The partial products, pp_i , can then be summed in a tree of SD modulo adders. Compared to an addition tree that sums all n partial products without recoding, the top level consisting of $n/2$ adders is replaced by one adder and some logic. The suggested reduction of partial products results in lesser hardware complexity with a negligible slow down of the circuit.

3.3 RNS+SD Forward Converter

We have developed a new forward converter (i.e. for conversion from conventional binary to RNS+SD) for the $\{2^n - 1, 2^n, 2^n + 1\}$ moduli set. Essentially, the new converter structure is based on the binary-to-RNS converter described in [7] but we have adapted it for RNS+SD arithmetic. The use of SD arithmetic is very convenient because, in contrast with a conventional binary implementation, the use of simplified SD residue adders reduces the complexity of the converter.

For a $3n$ -bit binary integer, Y , in the range $[0, M)$, where $M = (2^n - 1)2^n(2^n + 1)$, the vectors

$$\begin{aligned} k_0 &= y_{n-1} \cdots y_0 \\ k_1 &= y_{2n-1} \cdots y_n \\ k_2 &= y_{3n-1} \cdots y_{2n} \end{aligned}$$

are created. With these vectors the following operations

$$\begin{aligned} p_1 &= \langle k_2 + k_0 \rangle_{2^n - 1} \\ r_1 &= \langle p_1 + k_1 \rangle_{2^n - 1} \\ r_2 &= k_0 \\ p_2 &= \langle k_2 + k_0 \rangle_{2^n + 1} \\ r_3 &= \langle p_2 - k_1 \rangle_{2^n + 1} \end{aligned}$$

Eq. set 5. Modulo operations

are then performed to obtain the residues $\{r_1, r_2, r_3\}$ which is the RNS representation of Y .

The modulo operations in Eq. set 5 become trivial with the SD system and are no more complex than normal SD additions. Also, since the input, Y , is a binary integer the modulo SD adders calculating p_1 and p_2 can be greatly simplified as they will only require a few gates per bit. This results in a very small and fast (constant time) converter. Fig. 1 shows a schematic overview of the converter.

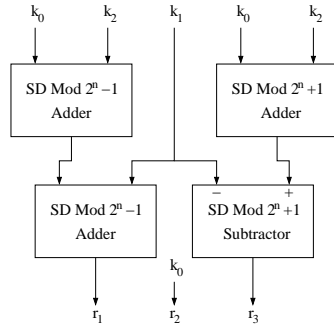


Fig. 1. Forward converter

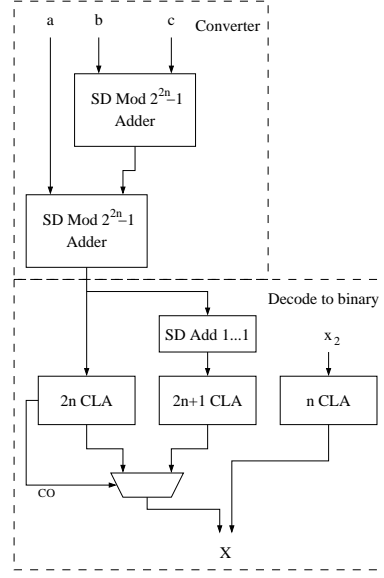


Fig. 2. Backward converter

3.4 RNS+SD Backward Converter

A backward converter for the moduli set $\{2^n - 1, 2^n, 2^n + 1\}$ has also been developed. The backward converter is based on a modified version of the New Chinese Remainder Theorems (CRT-1 as stated in [8]). We will present our modifications of CRT-1 and also our new SD implementation of the modified theorem in the following paragraphs. CRT-1 states

$$X = x_2 + 2^n \langle x_2 - x_3 + 2^{n-1}(2^n + 1)(x_1 - 2x_2 + x_3) \rangle_{2^{2n-1}} \quad (6)$$

where $\{x_1, x_2, x_3\}$ is the RNS representation of X . Eq. (6) can be written as

$$X = x_2 + 2^n \langle x_2 - 2^{2n}x_2 - x_3 - 2^n x_2 + 2^{n-1}(2^n + 1)(x_1 + x_3) \rangle_{2^{2n-1}} \quad (7)$$

and by using the first rotation rule in Eq. set 4 with $y = [0 \dots 0x_2]$ and $a = p = 2n$ Eq. (7) simplifies to

$$X = x_2 + 2^n \langle -x_3 - 2^n x_2 + 2^{n-1}(2^n + 1)(x_1 + x_3) \rangle_{2^{2n-1}}$$

Now, by using the fact that x_3 is n -digits in the SD system the above expression can be written as

$$X = x_2 + 2^n \langle a + b + c \rangle_{2^{2n-1}}$$

where a, b and c have the digit length $2n$ and are formed by concatenation and rotation as follows

$$\begin{aligned} a &= [(-x_2) (-x_3)] \\ b &= [x_1 x_{1_{n-1}} \dots x_1 x_{1_{n-1}} x_1] \\ c &= [x_3 x_{3_{n-1}} \dots x_3 x_{3_{n-1}} x_3]. \end{aligned}$$

Forming a, b and c can be done without using any gates but adding a, b and c will however require two $2n$ -digit SD modulo adders. The last step, i.e. the computation of $x_2 + 2^n \langle \dots \rangle_{2^{2n-1}}$, may also be done with concatenation and will thus not require any additional gates.

Converting the SD representation of X to binary can be done by using a $3n$ -bit carry-look-ahead adder (CLA) (see Eq. (1)), but the fact that the lower part of X consists of the residue x_2 can be used to convert each part independently. The upper part of X , $\langle a + b + c \rangle_{2^{2n-1}}$, can be converted by a $2n$ -bit CLA and the lower part, x_2 , can be converted by a n -bit CLA. Any carry-outs from the CLAs can safely be ignored as the two parts are bound by their moduli.

X will be in the range $(-M, M)$, $M = (2^n - 1)2^n(2^n + 1)$ due to the fact that SD residue adders also use the negative range. If it is necessary that the output from the converter to be in the range $[0, M)$, then the positive value of $\langle X \rangle_M$ is the correct output which can be calculated by adding $(2^n - 1)2^n(2^n + 1) = [1_{2n-1} \dots 1_0 0_{n-1} \dots 0_0]$ to X if X is negative. This is done by adding 1's to the upper $2n$ -bits of X . The addition is very simple in the SD system as the rules in Table 1 can in that case be simplified to Table 5.

Table 5. Adding ones to an SD integer

$x_i y_i$	01	11	$\bar{1}1$
c_{i+1}	1	1	0
u_i	$\bar{1}$	0	0

When converting to binary, it is simplest to just first add 1's to the upper $2n$ -bits of X , before converting, and then convert the result and the original $2n$ -bits to binary using two $2n$ -bit CLAs operating in parallel. The correct binary value can then be selected by examining the sign bit of the converted original $2n$ -bits. The complete arrangement is shown in Fig. 2.

4 Implementation Results

The presented circuits have been implemented in parameterized (word length generic) VHDL (see [9]) and performance estimation with various word lengths has been carried out. The units have been synthesised and optimized for the UMC13 0.13 μ m standard cell library under typical operating conditions using Synopsys Design Compiler. The

circuits are synthesized without timing constraints and can thus be made faster at the cost of area. Some of the results are presented in Table 6.

Table 6. Performance evaluation

n	16		32		64	
	delay (ns)	area (μm^2)	delay (ns)	area (μm^2)	delay (ns)	area (μm^2)
SD Adder	0.49	1142.1	0.49	2277.3	0.49	4544.2
SD Mod 2^n-1 Adder	0.49	1143.8	0.49	2277.3	0.49	4544.1
SD Mod 2^n+1 Adder	0.49	1142.1	0.49	2275.5	0.49	4542.4
BIN-SD Mod ^a 2^n-1 Adder	0.19	414.6	0.19	829.1	0.19	1658.2
BIN-SD Mod ^a 2^n+1 Adder	0.33	459.5	0.33	872.3	0.24	1694.6
Fast Binary CLA Adder ^b	0.51	1475.6	0.8	3236.4	1.4 ^c	6758 ^c
n	6		12		21	
SD Mod 2^n+1 Multiplier	2.13	2861.4	3.14	10827.1	3.73	32783.5
BIN to RNS+SD converter ^d	0.68	945.0	0.68	1852.0	0.67	3215.1
RNS+SD to BIN converter ^e	3.04	3146.9	4.63	6207.3	6.93	10798.7

a. SD modulo adder with binary inputs

b. Conventional binary carry-look-ahead adder for reference

c. Linear estimation

d. The input has the bit length $3n$ and the three outputs each have the digit length n .

e. The three inputs each have the bit length n and the output has the bit length $3n$.

The results confirm that the complexity of the SD modulo adders are almost exactly the same as for normal SD adders.

For the simplified SD modulo adders (those with binary inputs) used in the BIN to RNS+SD converter, the complexity is greatly reduced. This results in a very efficient forward converter, especially compared to conventional binary converters that at least have the area complexity of three CLA adders and a delay of at least two CLA adders.

The backward converter is however less efficient, since the conversion of the intermediate SD integer into binary adds some delay.

5 Conclusions

Improved RNS+SD arithmetic circuits have been presented and new RNS+SD converters for the moduli $\{2^n - 1, 2^n, 2^n + 1\}$ have been described.

By using a more efficient SD adder cell, we have been able to implement faster and smaller SD modulo adder circuits. These circuits can be simplified if one or both of the inputs are in conventional binary format.

The number of partial products have been reduced to half in our modulo multiplier which results in lesser hardware complexity and combined with the faster modulo adders also in improved speed performance.

The developed RNS+SD converters make it possible to integrate RNS+SD circuits with conventional binary systems. Converting to RNS+SD from conventional binary can be done very efficiently but the reverse operation, converting from RNS+SD to conventional binary, is a more costly operation. The reverse operation is however very suitable for pipelining in two or more steps, to reduce an otherwise perhaps limiting delay.

References

1. Shugang Wei and Kensuke Shimizu, "Fast residue arithmetic multipliers based on signed-digit number system", in IEEE 8th Conf. Electronics, *Circuits and Systems*, vol. 1, pp. 263-266, 2001.
2. Shugang Wei and Kensuke Shimizu, "A novel residue arithmetic hardware algorithm using a signed-digit number representation", *IEICE Trans. inf. & syst.*, vol. E83-D, no. 12, pp. 2056-2064, Dec. 2000.
3. Israel Koren, *Computer Arithmetic Algorithms*, 2nd edition. Natick, MA: A K Peters Ltd, 2002. ISBN: 1-56881-160-8.
4. Naofumi Takagi, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", *IEEE Transactions on Computers*, vol. c-34, no. 9, pp. 789-796, Sep. 1985.
5. S. Kuninobu, T. Nishiyama, H. Edamatu, T. Taniguchi and N. Takagi, "Design Of High Speed MOS Multiplier And Divider Using Redundant Binary Representation", in *IEEE Proc. 8th Symp. Computer Arithmetic*, pp. 80-86, 1987.
6. A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer and P. G. A. Jespers, "A New Carry-Free Division Algorithm and its Application to a Singler-Chip 1024-b RSA Processor", *IEEE Journal of solid-state circuits*, vol. 25, no. 3, pp. 748-756, Jun. 1990.
7. B. Vinnakota and V.V.Bapeswara Rao, "Fast Conversion Techniques for Binary-Residue Number systems", *IEEE Trans. Circuits and Systems*, vol. 41, no. 12, Dec. 1994.
8. Y. Wang, "Residue-to-binary converters based on new chinese remainder theorems", *IEEE Trans. Circuits Syst. II*, pp. 197-206, Mar. 2000
9. A. Lindström and M. Nordseth. (2003, Mars). *VHDL Library of Nonstandard Arithmetic Units*. [Online]. Available: <http://www.ce.chalmers.se/arithdb/>

