

# VHDL Library of Nonstandard Arithmetic Units

Anders Lindström, Michael Nordseth and Lars Bengtsson

Technical Report 03-01, August 2003.

Department of Computer Engineering  
Chalmers University of Technology

**Abstract** - This report presents a new VHDL library that contains circuit designs for various arithmetic operations within the residue number and the signed-digit systems. Besides components for residue number system (RNS) and signed-digit (SD) arithmetic operations such as addition, multiplication and modulo arithmetic, the library also contains converters for SD, RNS and for the new RNS with SD representation approach.

**Keywords** - VHDL, signed-digit (SD), residue number system (RNS), modulo arithmetic, converter.

## 1 Introduction

The residue number system (RNS) and the signed-digit (SD) system have mostly been used in applications which require high speed computation, such as real-time digital signal processing, due to the systems ability to perform parallel and carry free arithmetic.

A residue number system 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. This provides a speed-up for arithmetic operations that are inherently dependent on operand length, such as addition and multiplication. The disadvantages of using an RNS are the complexity involved in division, magnitude comparisons and conversions between binary and residue numbers.

The SD representation is a redundant number system and therefore facilitates carry-free addition [1], SD numbers can be added in constant time independently of operand length. The overhead involved in using SD arithmetic is foremost the reverse conversion to binary and the extra bits necessary for representing a number. Also, sign detection and comparisons are more costly than in ordinary binary representation.

If the disadvantages of these systems can be avoided or are small compared to the gains, then the properties of RNS and SD can be used to reduce circuit delay, area and/or power consumption. These conditions are often true in applications with a high multiplication or addition frequency such as digital filtering but RNS and SD have also been used in cryptography circuits and in fault tolerant systems due to their inherent special properties.

However, there exist very few reference designs of SD and RNS arithmetic circuits and those that exist do not provide a complete set of units for a given application. Especially, complete solutions with RNS represented in the SD system (RNS+SD) seem absent.

A library of arithmetic circuits for these representations (SD, RNS and RNS+SD) has been developed in this work as parameterized (operand length generic) and synthesizable VHDL. These circuits are meant to be used as building blocks in applications such as digital signal processors or as a tool to evaluate the use of alternative number systems in an application.

In the following sections a brief mathematical overview is given followed by a presentation of the library components with a functional description and a brief overview of implementation aspects. Additionally, some speed, area and power results obtained from implementing these circuits in 0.13  $\mu\text{m}$  CMOS will be discussed.

## 2 Foundations

### 2.1 The Residue Number System

In a residue number system an integer  $X$  is represented as an ordered set of  $r$  residues  $\{x_1, x_2, \dots, x_r\}$ , where  $x_i = X \bmod m_i$ . The system is defined by a set of relative prime numbers  $\{m_1, m_2, \dots, m_r\}$  called the moduli. Any integer in the range  $[0, M)$  can be uniquely represented where  $M = m_1 \cdot m_2 \cdot \dots \cdot m_r$  is called the dynamic range of the moduli set.

The choice of moduli set is crucial to the representational efficiency and to the complexity and delay circuits based on the system. The moduli set  $\{2^n - 1, 2^n, 2^n + 1\}$  is used throughout this work. This set makes high speed

forward and reverse conversion achievable, even if the parallelism is somewhat limited due to the fixed number of residues. These desirable properties results from the existence of suitable closed form inverses for numbers of this kind. Furthermore, using this set makes it possible to give a parameterized generic description of the circuits and thus making them easy to adapt for different applications.

## 2.2 The Signed Digit number system

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

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

where  $y_i \in \{\bar{1}, 0, 1\}$ . This representation permits several ways to represent an integer, thus making it a redundant number system.

The binary representation of an SD digit,  $y_i$ , requires at least two binary bits,  $y_i = [y_i^-, y_i^+]$ . The binary coding used throughout this report is  $0 = [00]$ ,  $1 = [01]$  and  $\bar{1} = [10]$ . With this coding the value of an n-digit SD integer can be written as

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

All carry propagation arising in addition can be avoided with the SD system, which is the primary reason for its use. Note that since a negative integer can be represented without any special sign digit, it is possible to represent any integer and its negation with an equally number of digits.

## 3 The Arithmetic units

An overview of the available units in the VHDL library follows in Table I. This sections will give a brief overview of the presented units and section 4 will give a more detailed presentation of the implementation of these units.

TABLE I Library units

Name	Description
<b>SD adders and subtractors (operate on two n-digit SD inputs)</b>	
sd_add	SD adder
sd_sub	SD subtractor
sd_add_mod2Nm1	SD modulo $2^n-1$ adder
sd_add_mod2Np1	SD modulo $2^n+1$ adder
<b>Miscellaneous units (operate on one n-digit SD or binary input)</b>	
sd_encode_uns	Binary (unsigned) to SD converter (only wiring)
sd_encode_sgn	Binary (2's compl.) to SD converter (only wiring)
sd_decode	SD to binary converter
sd_neg	SD negation (only wiring)
sd_sgn_det	SD zero and sign detection
<b>Binary SD adders and subtractors (operate on two n-bit binary inputs)</b>	
bin2sd_add_uns	SD adder with binary inputs (unsigned)
bin2sd_add_sgn	SD adder with binary inputs (2's compl.)
bin2sd_sub_uns	SD subtractor with binary inputs (unsigned)
bin2sd_sub_sgn	SD subtractor with binary inputs (2's compl.)

**TABLE I** Library units

Name	Description
bin2sd_add_mod2Nm1	SD modulo $2^n-1$ adder with binary inputs
bin2sd_add_mod2Np1	SD modulo $2^n+1$ adder with binary inputs
<b>Mixed SD adders and subtractors (operates on two n-digit inputs)</b>	
sdbin_add_uns	SD adder with one binary input (unsigned)
sdbin_add_sgn	SD adder with one binary input (2's compl.)
sdbin_sub_uns	SD subtractor with one binary input (unsigned)
sdbin_sub_sgn	SD subtractor with one binary input (2's compl.)
sdbin_add_mod2Nm1	SD modulo $2^n-1$ adder with one binary input
sdbin_add_mod2Np1	SD modulo $2^n+1$ adder with one binary input
<b>SD multipliers (operate on two n-digit SD inputs)</b>	
sd_mul	SD multiplier
sd_mul_mod2Nm	SD modulo $2^n$ multiplier
sd_mul_mod2Nm1	SD modulo $2^n-1$ multiplier
sd_mul_mod2Np1	SD modulo $2^n+1$ multiplier
<b>RNS converters</b>	
rns2bin	RNS $\{2^{n+1}, 2^n, 2^n-1\}$ to binary converter
bin2rns	Binary to RNS $\{2^{n+1}, 2^n, 2^n-1\}$ converter
<b>RNS+SD converters</b>	
bin2rns_sd	Binary to RNS+SD converter (residues in SD)
sd_rns2bin	RNS+SD to Binary converter (residues in SD)

### 3.1 SD adders

The most common algorithm for adding two SD integers,  $X$  and  $Y$ , is shown in Table II in which  $c_i$  denotes the (non-propagating) carry and  $u_i$  the interim sum.

**TABLE II** Rules for adding SD numbers

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

These rules avoid any carry propagation when the final sum is calculated,  $s_i = c_i + u_i, s_i \in \{\bar{1}, 0, 1\}$ . The VHDL component “sd\_add” takes two n-digit SD integers and adds them with an algorithm similar to Table II (but optimized for the SD digit binary coding used), the output is the (n+1)-digit sum.

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 equation (1). This operation simply changes the sign of each digit,  $1 \Leftrightarrow \bar{1}$ , and can be done without any gates. This makes subtraction very easy but the component “sd\_sub” is provided for convenience (an SD adder with the correct wiring).

The SD modulo adders are based on the algorithm found in [3] but applied to the SD adder described above. This makes the modulo adders just a few gates larger than the normal SD adder (and equally fast) and compared to binary modulo adders there is no carry propagation involved in the end around carry (due to the carry free nature

of SD adders). Compared to a binary  $2^n + 1$  modulo adder (when not using diminished-1 representation), the SD counterpart also has the advantage that it only needs  $n$ -digits to represent the result. If the result is greater than  $2^n - 1$  then the result will instead be taken from the negative range, for example  $\langle [110] + [010] = [1000] \rangle_9 = [00\bar{1}] = -1 = 8 - 9$  where  $\langle X \rangle_m$  is the operation  $X \bmod m$ . In fact, the negative range will be used by every SD modulo adder in this library whenever it is suitable to reduce the circuit complexity.

The complexity of an SD adder or SD modulo adder can be dramatically reduced if the inputs are in binary representation (called binary SD adders in Table I). It is also possible to reduce the complexity if just one of the inputs is in binary representation (called mixed SD adders in Table I).

### 3.2 SD multipliers

An SD multiplier is constructed from SD adders in a tree structure, similar to conventional binary multipliers. The tree structure used is a two to one reduction tree (i.e. a binary tree) and is therefore most effective if the word length of the two operands is a power of two. The SD multiplier in the VHDL library takes two  $n$ -digit SD integers as input and outputs a  $(2n + 1)$ -digit product.

Conventional schemes for reducing the number of partial products, as used in binary multipliers, cannot be used in SD multipliers as both of the operands are in SD format. However, SD arithmetic allows reducing the number of partial products without any complex recoding of the multiplier as used in conventional schemes. This type of partial product reduction is used in both the normal multiplier and the modulo multipliers and results in lesser hardware complexity with a negligible slow down of the circuit.

SD modulo multipliers have a simpler structure than SD multipliers as no shifting is needed inside the tree and therefore all modulo adders in the tree are of the same digit length.

### 3.3 SD converters

Converting a binary integer to SD representation can be done without any gates (any binary integer is also a valid SD integer). When handling signed (2's complement) binary integers the sign bit needs to be negated ( $1 \leftrightarrow \bar{1}$ ), which also can be done without any gates.

Converting from SD to binary can be done with a binary subtraction as shown in equation (1) and this is usually done with a carry look ahead (CLA) adder. There also exist slightly more efficient converters as described in [4] and [5], but the first approach is used in this work for simplicity.

### 3.4 SD zero/sign detectors

Zero detection is done in the same way as normal binary zero detection (a reduced OR operation) and the delay of the circuit will thus depend on the operand length. Sign detection in the SD system is also a complex operation as the sign of the most significant non-zero digit needs to be found. In this work these two operations are done at the same time due to the fact that zero detection can easily be done during sign detection.

### 3.5 RNS converters

The binary to RNS (forward) converter implements the procedure described in [6]. The residues are constructed by dividing a  $3n$ -bit binary input integer into three equal sized parts and then performing modulo addition on those. The three output residues are  $n$ -bit binary integers except for the residue corresponding to modulo  $2^n + 1$  which is a  $n + 1$ -bit integer. The unit is described in behavioural VHDL with  $n$  as generic input parameter.

The RNS to binary (backward) converter is a ROM-less adder based converter based on the "Converter III" approach presented in [7]. One of the modifications to this approach is a simplification of the selector-logic which reduces the units complexity. The converter calculates the  $3n$ -bit binary representation of the RNS integer  $X$  from its three residues  $x_1$ ,  $x_2$ , and  $x_3$ .

### 3.6 RNS+SD converters

These converters utilizes both number systems trying to exploit their advantages. Instead of representing the RNS integer residues in binary they are represented as SD integers. Suggestions of using this combination have previously been made in [3]. The advantages of this is the SD systems ability to perform fast modulo  $2^n - 1$  and modulo  $2^n + 1$  addition. Furthermore some synergy effects occur resulting in even smaller and faster converters.

The moduli set used is, as before,  $\{2^n - 1, 2^n, 2^n + 1\}$ . The forward converter takes a  $3n$ -bit binary integer and outputs its three  $n$ -digit residues in SD representation. Modulo arithmetic can then be performed on these three residues as normal in RNS systems but substituting the normal binary modulo circuits with their faster SD modulo equivalent. It is important to keep the results of all residue arithmetic operations within the dynamic range,

$[0, M)$ ,  $M = (2^n - 1)(2^n)(2^n + 1)$ , or the results will overflow (just as in normal RNS arithmetic). The three resulting  $n$ -digit residues can then be translated to normal binary with the backward converter.

The forward (binary to RNS+SD) converter is based on the same procedure as the binary to RNS converter but utilizes the SD system to reduce complexity. The modulo operations become no more complex than normal SD additions unlike corresponding binary unit. In addition, considering that the input is a binary integer, simplified modulo SD adders can be used. This results in a small and fast (constant time) converter.

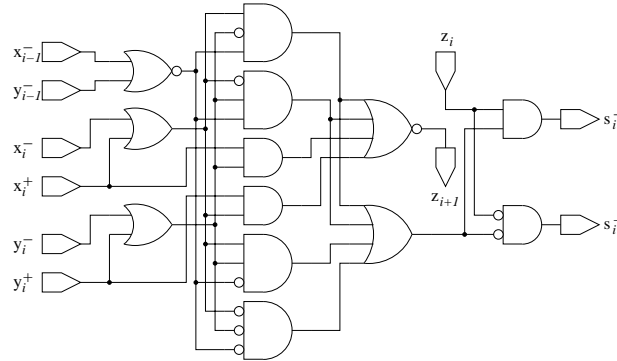
A backward (RNS+SD to binary) converter is based on a modified version of the New Chinese Remainder Theorems (CRT-1 as described in [8]). The backward converter is more complex than the forward converter as it needs to detect and handle the negative range used by the SD modulo adders when converting to binary.

## 4 Implementation details

### 4.1 SD addition

An  $n$ -digit SD adder consists of  $n$  identical adder cells, operating in parallel and independently of each other. The SD adder cell used in the VHDL library is an implementation of the circuit described in [2]. Eq. Set 2 describes the logic for the  $i$ :th cell ( $i = 0 \dots n - 1$ ).

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



Eq. Set 2 SD addition

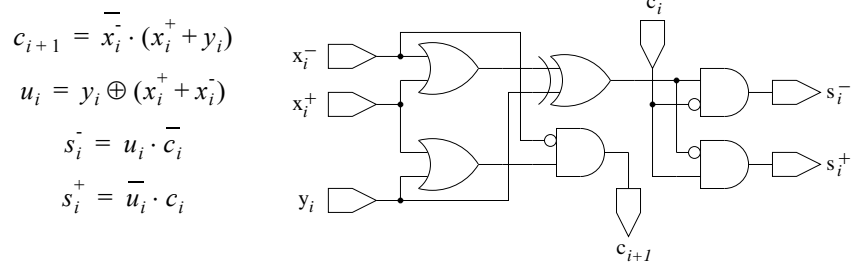
The least significant digit should be calculated with  $z_0 = 1$ . SD adders, like normal adders, also have a “carry-out”,  $s_n^- = \overline{p_n} \cdot z_n$  and  $s_n^+ = p_n \cdot \overline{z_n}$ .

If one of the inputs to the adder is in binary representation, then Table II can be simplified into Table III.

TABLE III Rules for mixed input

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

These rules are very simple to translate into boolean logic as is shown in Eq. Set 3.



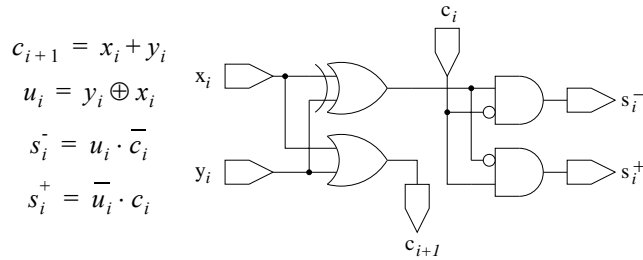
**Eq. Set 3** Mixed SD addition

If both of the inputs are in binary representation, then Table III can be further simplified into Table IV.

**TABLE IV** Rules for binary input

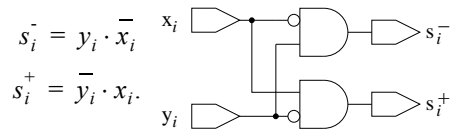
$x_i y_i$	00	01	10	11
$c_{i+1}$	0	1	1	1
$u_i$	0	$\bar{1}$	$\bar{1}$	0

The translation of these rules into boolean logic is shown in Eq. Set 4.



**Eq. Set 4** Binary SD addition

SD subtraction can be done with SD adders in the case of two or one SD inputs, as SD negation only involves wiring (the negation can easily be transferred to the SD input,  $x - y = -(-x + y)$ ). However, if both of the inputs are in binary representation then the logic can be reduced to



It is interesting to note that the result of this subtraction has the same operand width as the two inputs (n-digits). This makes it possible to also use this circuit for SD modulo  $2^n - 1$ ,  $2^n$  and  $2^n + 1$  subtraction with two binary inputs as the results never will be greater than  $2^n - 1$ .

The paragraphs above assume that the binary inputs are unsigned, but it is also possible to use signed binary (2's complement). A signed binary integer can be converted to the correct SD representation by negating the most significant bit ( $1 \Leftrightarrow \bar{1}$ ), but this will, however, require a full SD adder or modification to be made to the mixed or binary SD adder. The modifications needed can be simplified by the fact that the SD representation of the signed binary  $n$ -bit integer  $y$  can be written as

$$(-y_{n-1})y_{n-2} \dots y_0 = (-y_{n-1})y_{n-1}y_{n-2} \dots y_0$$

where  $y_{n-1} \dots y_0$  can be handled as an normal binary input. The only modification that needs to be made is in the

calculation of the “carry-out”, Table V shows this in the case of mixed input.

**TABLE V** Carry-out calculation

$-y_{n-1}$	$c_n$	$c_{out}$
0	0	0
0	1	1
$\bar{1}$	0	$\bar{1}$
$\bar{1}$	1	0

If both inputs are in signed binary format then the correct “carry-out” is

$$\begin{aligned} c_{out}^- &= x_{n-1} \cdot y_{n-1} \\ c_{out}^+ &= 0. \end{aligned}$$

#### 4.2 SD modulo addition

Modulo  $2^n - 1$ ,  $2^n$  and  $2^n + 1$  addition can be done easily in the SD system, as these three moduli only requires an carry end-around operation. To handle modulo addition, the rules in Table II needs to be extended by the rules in Table VI.

**TABLE VI** Modulo operations

Moduli	Carry end-around
$2^n - 1$	$c_0 = c_n$
$2^n$	$c_0 = 0$
$2^n + 1$	$c_0 = -c_n$

The end-around operation for modulo  $2^n - 1$  addition can be trivially applied to Eq. Set 2 and becomes

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

Modulo  $2^n$  addition can be implemented even more simply, as a normal SD adder can be used if the “carry-out” is ignored. Modulo  $2^n + 1$  addition is however more complex, as the end-around needs to be negated which can not be done directly in Eq. Set 2. To negate  $c_{i+1}$  in Table II,  $x_i$  and  $y_i$  needs to be negated and the logic in the second row must be inverted. Applied to Eq. Set 2, this end-around carry is shown in Eq. Set 5.

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

**Eq. Set 5**  $2^n+1$  end around carry

Like normal SD adders, SD modulo adders can be simplified if one or both of the inputs are in binary representation. Modulo  $2^n$  and modulo  $2^n - 1$  addition are easy to implement in both cases, modulo  $2^n$  addition just ignores the carry out as normal and modulo  $2^n - 1$  addition has the simple inclusion in Eq. Set 3 and Eq. Set 4 of an end-around carry,  $c_0 = c_n$ .

In the case of modulo  $2^n + 1$  addition with one or both inputs in binary representation, Eq. Set 2 and Eq. Set 5

needs to be simplified instead. Eq. Set 6 shows the result with one binary input and Eq. Set 7 with two binary inputs.

$$\begin{aligned}
p_0 &= \overline{x_{n-1}^+ + y_{n-1}} \\
p_i &= \overline{x_{i-1}^-} \\
xid_i &= x_i^- + x_i^+ \\
z_0 &= \overline{xid_{n-1} \cdot y_{n-1} \cdot p_{n-1} + \overline{xid_{n-1} \cdot y_{n-1} \cdot p_{n-1}} + y_{n-1} \cdot x_{n-1}^-} \\
z_{i+1} &= \overline{xid_i \cdot p_i + y_i \cdot p_i + y_i \cdot xid_i} \\
t_i &= xid_i \cdot \overline{y_i} \cdot p_i + \overline{xid_i} \cdot y_i \cdot p_i + xid_i \cdot y_i \cdot \overline{p_i} + \overline{xid_i} \cdot \overline{y_i} \cdot \overline{p_i} \\
s_i^- &= t_i \cdot z_i \\
s_i^+ &= \overline{t_i} \cdot \overline{z_i}
\end{aligned}$$

**Eq. Set 6** Modulo  $2^n+1$  addition with  $y$  in binary representation

$$\begin{aligned}
p_0 &= \overline{x_{n-1} + y_{n-1}} \\
z_0 &= 1 \\
z_1 &= \overline{x_0 \cdot p_0 + y_0 \cdot p_0 + y_0 \cdot x_0} \\
z_{i+1} &= \overline{x_i + y_i} \\
t_0 &= x_0 \cdot \overline{y_0} \cdot p_0 + \overline{x_0} \cdot y_0 \cdot p_0 + x_0 \cdot y_0 \cdot \overline{p_0} + \overline{x_0} \cdot \overline{y_0} \cdot \overline{p_0} \\
t_i &= x_i \oplus y_i \\
s_i^- &= t_i \cdot z_i \\
s_i^+ &= \overline{t_i} \cdot \overline{z_i}
\end{aligned}$$

**Eq. Set 7** Modulo  $2^n+1$  addition with two binary inputs

SD modulo subtraction,  $x - y$ , where  $y$  is in binary representation can be accomplished by using a SD modulo adder as  $\langle x - y \rangle_m = -\langle -x + y \rangle_m$ .

### 4.3 SD multiplication

Multiplication is defined as

$$x \cdot y = \sum_{i=0}^{n-1} y_i (2^i \cdot x)$$

where  $n$  is the word length. The partial products (i.e.  $y_i(2^i \cdot x)$ ) can be summed in a tree of SD adders.

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

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

where the value of  $rp_i$  are as shown in Table VII. These values can be generated using only one SD adder (to gen-

erate  $3x = 2x + x$ ) as shift and negation does not require any additional gates.

**TABLE VII** Possible values of  $rp_i$

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

Modulo multiplication is done in a similar way as above. 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 [9].

The number of partial products can in this case also 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 VII. These values can be generated using only one SD modulo adder (to generate  $3x = 2x + x$ ) as shift, rotation (see Eq. Set 8) and negation does not require any additional gates.

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 8. 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 8** Rotation

The partial products,  $pp_i$ , can then be summed in a tree of SD modulo adders.

#### 4.4 RNS forward converter

This is an implementation of the procedure described in [6]. For a  $3n$ -bit binary integer  $Y$  in the range  $[0, M)$ , where  $M = (2^n - 1)2^n(2^n + 1)$ , the three vectors  $k_0$ ,  $k_1$ , and  $k_2$  are created.

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

Finally the RNS representation of  $Y$  (i.e.  $\{r_1, r_2, r_3\}$ ) is obtained through the modulo operations in Eq. Set 9.

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

**Eq. Set 9** Converter modulo operations

A limiting factor of this converter are these modulo operations, therefore the final delay of the circuit is highly dependent on the modulo adder implementation. As a guideline, the highest delay is the delay of four  $n$ -bit additions.

#### 4.5 RNS backward converter

The RNS to binary converter is a modified implementation of the ‘‘Converter III’’ approach described in [8]. It calculates the  $3n$ -bit binary value  $X$  from the three input residues  $x_1$ ,  $x_2$ , and  $x_3$ . The mathematical foundation for its operation is described in the following equations, which are derived from the New Chinese Remainder Theorem I (New CRT-1 [8]):

$$\begin{aligned} X &= x_2 + 2^n \cdot Y \\ \text{where } Y &= \langle A + 2^n \cdot B \rangle_{2^{2n} - 1} \\ A &= \left\lfloor \frac{(x_1 + (x_{1_0} \oplus x_{3_0}) \cdot 2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor \\ B &= \left\lfloor \frac{(x_1 + (x_{1_0} \oplus x_{3_0}) \cdot 2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor \end{aligned}$$

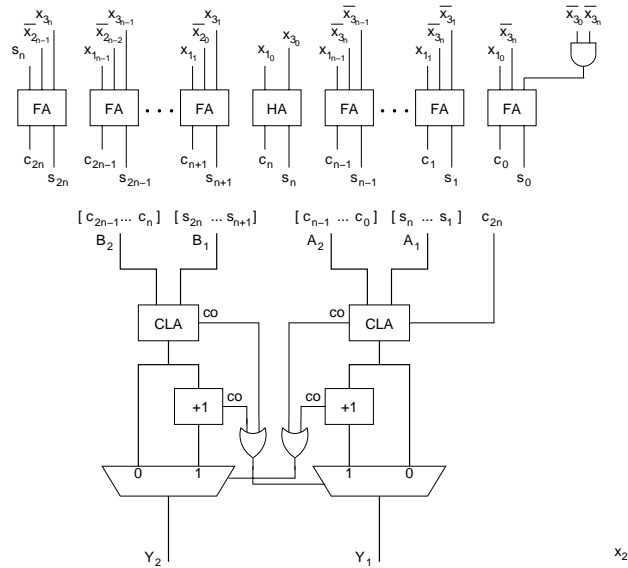
Four intermediate  $n$ -bit numbers  $A_1$ ,  $A_2$ ,  $B_1$  and  $B_2$  are calculated using  $2n$  full-adders and one half-adder as shown in Figure 1. The delay of this operation is  $t_{HA} + t_{FA}$ .  $A$  and  $B$  are then processed by two parallel data paths

$$\begin{aligned} A_1 &= [s_n s_{n-1} \dots s_1] \\ A_2 &= [c_{n-1} c_{n-2} \dots c_0] \\ B_1 &= [s_{2n} s_{2n-1} \dots s_{n+1}] \\ B_2 &= [c_{2n-1} c_{2n-2} \dots c_n] \end{aligned}$$

which produces the output  $Y_1$  and  $Y_2$ . Both paths consists of a CLA, a plus 1 counter, and a multiplexer to select  $Y_x$  as either the CLA output or the counter output depending on the carry-outs of opposite data path.

The converter output is finally formed by simply concatenating  $Y_2$ ,  $Y_1$  and the residue  $x_2$  which each consists

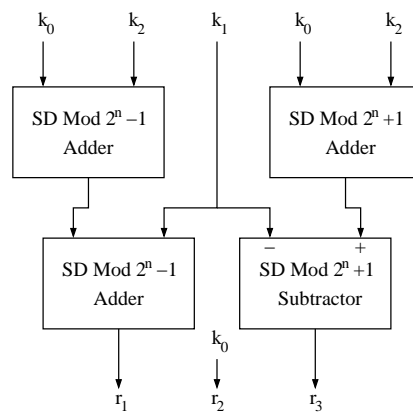
of  $n$ -bits.



**FIGURE 1** Binary backward converter

#### 4.6 RNS+SD forward converter

The modulo operations in Eq. Set 9 become trivial with the SD system and are no more complex than normal SD addition. 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. A schematic view of this circuit is shown in Figure 2.



**FIGURE 2** Forward converter

#### 4.7 RNS+SD backward converter

The backward converter is based on a modified version of the New Chinese Remainder Theorems (CRT-1 as stated in [8]). CRT-1 states

$$X = x_2 + 2^n \langle x_2 - x_3 + 2^{n-1}(2^n + 1)(x_1 - 2x_2 + x_3) \rangle_{2^{2n-1}} \quad (10)$$

where  $\{x_1, x_2, x_3\}$  is the RNS representation of  $X$ . Equation (10) can be written as

$$X = x_2 + 2^n \cdot \langle x_2 - 2^{2n}x_2 - x_3 - 2^n x_2 + 2^{n-1}(2^n + 1)(x_1 + x_3) \rangle_{2^{2n-1}} \quad (11)$$

and by using the first rotation rule in Eq. Set 8 with  $y = [0 \dots 0x_2]$  and  $a = p = 2n$  Equation (11) 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

$$a = [(-x_2) (-x_3)]$$

$$b = [x_{1_0} x_{1_{n-1}} \dots x_{1_0} x_{1_{n-1}} x_{1_1}]$$

$$c = [x_{3_0} x_{3_{n-1}} \dots x_{3_0} x_{3_{n-1}} x_{3_1}].$$

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 Equation (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.

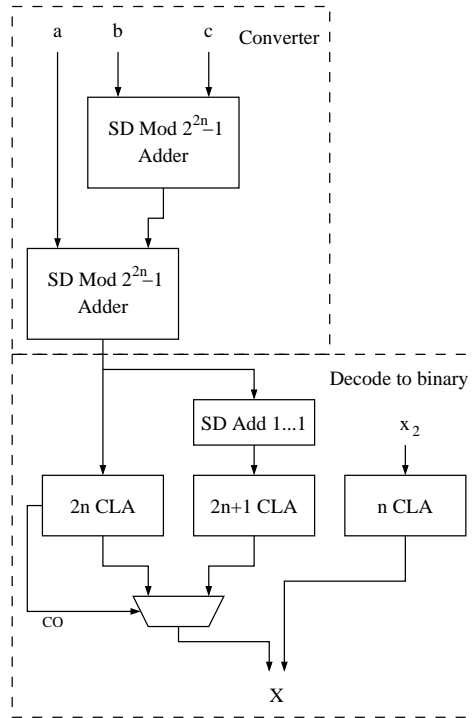
$X$  will be in the range  $(-M, M)$ ,  $M = (2^n - 1)2^n(2^n + 1)$  due to the fact that SD residue adders also use the negative range. If it is necessary that the output from the converter to be in the range  $[0, M)$ , then the positive value of  $\langle X \rangle_M$  is the correct output which 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 II can in that case be simplified to Table VIII.

**TABLE VIII** 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. A schematic view of

this circuit is shown in Figure 3.



**FIGURE 3** Backward converter

## 5 Synthesis for 0.13 $\mu\text{m}$

Performance estimation of the units for various bit lengths have been carried out. The units have been synthesised and optimized for the UMC13 (0.13  $\mu\text{m}$  CMOS) standard-cell library under typical PTV conditions using Synopsys Design Compiler (DC). No constraints were applied. The results are presented in Table IX.

TABLE IX Synthesis results

Circuit	delay (ns)			area ( $\mu\text{m}^2$ )			power (mW)		
	n=16	n=32	n=64	n=16	n=32	n=64	n=16	n=32	n=64
SD adder	0.49	0.49	0.49	1142	2277	4544	0.780	1.575	3.165
SD modulo $2^n-1$ adder	0.49	0.49	0.49	1144	2277	4544	0.797	1.590	3.180
SD modulo $2^n+1$ adder	0.49	0.49	0.49	1142	2276	4542	0.795	1.589	3.178
SD to binary converter	2.07	4.16	8.34	522.0	1047	2098	0.356	0.718	1.443
SD zero and sign detection	3.95	8.57	17.8	407.8	850.2	1735	0.155	0.313	0.626
SD adder with binary inputs (unsigned)	0.21	0.21	0.21	412.9	827.4	1657	0.291	0.590	1.189
SD modulo $2^n-1$ adder with binary inputs	0.19	0.19	0.19	414.6	829.1	1658	0.299	0.598	1.197
SD modulo $2^n+1$ adder with binary inputs	0.33	0.33	0.24	459.5	872.3	1695	0.321	0.617	1.215
SD adder with one binary input (unsigned)	0.29	0.29	0.29	625.5	1261	2533	0.397	0.804	1.621
SD modulo $2^n-1$ adder with one binary input	0.29	0.29	0.29	635.8	1272	2543	0.408	0.816	1.632
SD modulo $2^n+1$ adder with one binary input	0.43	0.43	0.43	958.9	1909	3816	0.687	1.353	2.694
SD multiplier	3.10			21150			16.00		
	n=6	n=12	n=21	n=6	n=12	n=21	n=6	n=12	n=21
SD modulo $2^n$ multiplier	1.77	3.07	4.02	2352	8517	25630	1.672	5.999	16.86
SD modulo $2^n-1$ multiplier	2.13	3.14	3.65	2865	10820	32820	2.304	8.724	27.11
SD modulo $2^n+1$ multiplier	2.13	3.14	3.73	2861	10830	32780	2.295	8.695	26.98
	n=6	n=12	n=21	n=6	n=12	n=21	n=6	n=12	n=21
RNS $\{2^{n+1}, 2^n, 2^{n-1}\}$ to binary converter	1.89	3.16	4.19	1298	2596	4547	0.985	2.010	3.450
Binary to RNS $\{2^{n+1}, 2^n, 2^{n-1}\}$ converter	3.25	5.4	7.11	1868	3776	7238	1.342	2.783	5.710
Binary to RNS+SD converter (residues in SD)	0.68	0.68	0.67	945	1852	3215	0.734	1.469	2.568
RNS+SD to Binary converter (residues in SD)	3.04	4.63	6.93	3147	6207	10800	2.569	5.126	8.955

## 6 Conclusions

We have presented a library of arithmetic units for RNS and SD representations. The library components can be used as building blocks in RNS/SD system designs or as reference when designing residue or SD arithmetic.

Units for the new RNS+SD approach are also presented. This makes it possible to explore the use of RNS+SD in systems where only RNS were used before. One could expect to gain in the conversion process with these units and probably also in arithmetic operations within the system. There are however, an additional delay introduced in the backward conversion process of this system.

The presented VHDL library is available online, see [10].

## References

- [1] Israel Koren. *Computer Arithmetic Algorithms*, 2:nd edition. Natick, MA: A K Peters Ltd, 2002. ISBN: 1-56881-160-8.
- [2] Naofumi Takagi, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Trans. on Computers*, vol. c-34, no. 9, pp. 789-796, september 1985.
- [3] Shugang Wei and Kensuke Shimizu, "Fast residue arithmetic multipliers based on signed-digit number system," *IEEE 8th Conf. Electronics, Circuits and Systems*, vol. 1, pp. 263-266, 2001.
- [4] Sung-Ming Yen, Chi-Sung Lai, Chin-Hsing Chen and Jau-Yien Lee, "An Efficient Redundant-Binary Number to Binary Number Converter," *IEEE Journal of Solid-State circuits*, vol. 27, no. 1, pp 109-112, Jan 1992.
- [5] H. R. Srinivas and Keshab K. Parhi, "A Fast VLSI Adder Architecture," *IEEE Journal of Solid-State circuits*, vol. 27, no. 5, pp 761-767, may 1992.
- [6] 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.
- [7] 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. Signal Processing*, vol. 50, no. 7, Jul 2002.
- [8] Y. Wang, "Residue-to-binary converters based on new chinese remainder theorems," *IEEE Trans. Circuits Syst. II*, pp. 197-206, Mar. 2000.
- [9] 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.
- [10] A. Lindström and M. Nordseth. (2003, Mars). *VHDL Library of Nonstandard Arithmetic Units*. [Online]. Available: <http://www.ce.chalmers.se/arithdb/>

