SIGN DETECTION IN THE SYMMETRIC RESIDUE NUMBER SYSTEM

Saroj Kaushik and R.K. Arora

Computer Centre, Indian Institute of Technology, Delhi
New Delhi 110016, India

ABSTRACT

This paper is concerned with the algebraic sign detection of a number in the Symmetric Residue Number System. A new approach has been suggested which completely avoids the time consuming process of the Symmetric Mixed Radix Conversion (SMRC). An algorithm based on the above approach, implementable in parallel for sign detection is also presented. The hardware representation of the above algorithm is shown. The time and hardware complexity required for the process have also been computed.

INTRODUCTION

Several kinds of representation for residue numbers have been proposed, each of which has merits and demerits. The Symmetric Residue Number System (SRNS) has many advantages over the Residue Number System (RNS) viz., it is easy in the SRNS to find the additive inverse of a residue digit, it gives an effective solution for sign sensing and magnitude determination etc. Many solutions to the problem of sign detection in the RNS have been suggested, but in the SRNS, there is no specific method for detecting the sign of a number except the SMRC process. Certainly it computes the sign but is relatively time consuming because of the cascaded process used. Besides, it furnishes more information (all symmetric mixed radix co-efficients). The purpose of this paper is to present a new approach for algebraic sign determination in the SRNS, which completely avoids the time consuming process of the SMRC.

SYMMETRIC RESIDUE ARITHMETIC FUNDAMENTALS

Before considering the problem of sign detection, the basic definitions and operations in the SRNS are introduced to make this paper self-contained.

Definition 1: The symmetric residue of $X$ modulo $m$ is the least remainder in absolute value when an integer $X$ is divided by another positive integer $m$. Here $m$ is called a modulus. A commonly used form of this condition is:

$$x \equiv x^* \ mod \ m + |x|m/;$$

where $|x| = \lfloor x/2 \rfloor$.

$x/m$ is the notation for symmetric residue of $x$ with respect to $m$. When $m$ is odd, there residues are symmetric with respect to the origin, but when $m$ is even, perfect symmetry is lost. In this case, the quantity $x^*$ is again the closest integer to $x$, except that if $X$ is of the form $nm/2$, where $n$ is odd, then the quantity $x^*$ is the closest integer to $x - 1/m$.

Definition 2: The ordered $n$-tuple $(X/m_1, X/m_2, \ldots, X/m_n)$ of symmetric residues of $X$ with respect to an ordered set $(m_1, m_2, \ldots, m_n)$ of $n$ moduli form the symmetric residue representation of $X$.

Definition 3: The SRNS is defined as a number system where the integers in the interval $[-m/2, +1, \frac{m}{2}, \ldots]$ (upper square brackets indicate a closed interval), where $m = m_1 m_2$ are represented by $i = 1$.

If these moduli are chosen to be pairwise relatively prime, then any integer in the interval $[-m/2, +1, \frac{m}{2}, \ldots]$ is uniquely $\lfloor y \rfloor$ denotes the closest integer to $y$.

**$\lfloor y \rfloor$ denotes the floor of $y$ i.e., the largest integer $\leq y$.**
represented by its symmetric residue representation and it is denoted by
\[
X \leftrightarrow \left( \frac{x}{m_1}, \frac{x}{m_2}, \ldots, \frac{x}{m_n} \right).
\]

The addition, subtraction and multiplication of two numbers \(X\) and \(Y\)
in the SANS is defined as follows as long as the operands and the result are falling in the interval
\[
\left[ \frac{m}{2} \right] + 1, \left[ \frac{m}{2} \right].
\]

\[
X \times Y \leftrightarrow \left( \frac{x}{m_1} \times \frac{y}{m_1}, \frac{x}{m_2} \times \frac{y}{m_2}, \ldots, \frac{x}{m_n} \times \frac{y}{m_n} \right)
\]

where the symbol \(\times\) represents addition, subtraction or multiplication of two numbers.

**Symmetric Mixed Conversion Process:** It is used to convert a residue code of a number to its symmetric mixed-radix representation. A number \(X\) may be expressed in symmetric mixed radix form as
\[
X = r_n m_1^n + r_{n-1} m_1^{n-1} + \ldots + r_2 m_1^2 + r_1 m_1 + r_0,
\]

where \(r_i\) and determined sequentially from equation \((1)\).

**General Approach for Sign Detection in the SANS**

We prove the following theorem based on which an algorithm has been suggested. An implementation of this algorithm has also been proposed which is parallel in nature. For the sake of simplicity, we write \(X_{/ m_1} = x_i, i = 1, 2, \ldots, n\). Let \(\hat{m}_1 = \frac{m}{m_1}\).

**Theorem**

For any modulus \(m_1, 1 \leq i \leq n\), the sign of a number \(X\) in the interval
\[
\left[ \frac{m}{2} \right] + 1, \left[ \frac{m}{2} \right]
\]
is established by a proposition \(P\) in the following way:

**Case (a):** If \(\hat{m}_1\) is odd, then \(X\) is non-negative iff
\[
P = \bigvee_{j=1}^{\hat{m}_1} \left\{ x_j = \frac{x}{j m_1} \wedge x_j \geq 0 \right\}
\]
is true for \(j = 1, 2, \ldots, i-1, i+1, \ldots, n\);

**Case (b):** If \(\hat{m}\) is even, then \(X\) is non-negative iff
\[
\left( \left[ \frac{m}{2} \right] \right)^{-1}
\]

\[
P = \bigvee_{j=1}^{\hat{m}_1} \left\{ x_j = \frac{x}{j m_1} \wedge x_j \geq 0 \right\}
\]

where \(\bigvee, \bigwedge\) are logical 'OR' and 'AND' operators.

**Proof:** Proof is given in an appendix.

**Remark 1:** The number of iterations can be reduced by choosing \(m_1\) to be the largest modulus.

**Algorithm for Sign Detection of a Residue Number**

The following algorithm, based on the above theorem is used to detect the sign of a residue number. Choose \(m_1\) to be the largest modulus and let \(P = \frac{m}{2}\).

**Algorithm**

**Input:** The number \(X \leftrightarrow \left( x_1, x_2, \ldots, x_n \right)\).

**Output:** The sign of \(X\).

**Procedure**

**Step 1:** Start the processes \(Q_1\) and \(Q_2\) simultaneously:

\(Q_1\) and \(Q_2\): Obtain the symmetric residue representation of \(x_i\) for \(i = 1, 2, \ldots, n\).

\(m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n\) as

\[
x_1 \leftrightarrow \left( x_1^1, x_2^1, \ldots, x_i^1, \frac{x_{i+1}^1}{m}, \ldots, x_n^1 \right),
\]

\[
m_1 \leftrightarrow \left( m_1^1, m_2^1, \ldots, m_{i-1}^1, m_{i+1}^1, \ldots, m_n^1 \right),
\]

where

\[
x_i^1 = x_i / m_j \text{ and } m_j^1 = m_j / m_i, j = 1, 2, \ldots, n
\]

**Step 2:** Start the processes \(R_0, R_1, R_2, \ldots, R_p, P, R^p\).

**Comment - Description of \(R_j\),\n**

\(j = 0, 1, \ldots, p\) and \(R^p\) is given at the end of this procedure.
Step 3: Find $S' = (\xi \cdot \alpha ^1 \cdots \alpha ^{p-1})$.

Step 4: If $\xi _i$ is even, then go to the next step else form $S = S', \alpha ^p$ and go to step 6.

Step 5: Form $S = S', \alpha ^p$.

Step 6: If $S = 0$ then $X$ is non-negative else it is negative. Stop.

**Description of the process $R_1^p$:**

Start the processes $R_0^1, R_1^2, \ldots , R_0^p$

where $R_0^p$ is $\bmod m_0$ comparator, which compares $x_k$ and $x_k^p$. If compared, then set $\beta _0^k = 1$;

otherwise $\beta _0^k = 0$, for $k = 1, 2, \ldots , i-1, i+1, \ldots , n$.

If $x_i \geq 0$, then set $a = 1$ else $a = 0$.

Compute a logical variable $\alpha ^0$ as

$$\alpha ^0 = (\beta _0^1 \beta _0^2 \cdots \beta _0^{i-1} \beta _0^{i+1} \cdots \beta _0^n a)$$

**Description of the process $R_j$, $j=1, 2, \ldots , p$:**

Let $j, m_j \leftrightarrow (\frac{j, m_j^1}{m_j^1}, \ldots , \frac{j, m_j^{i-1}}{m_j^{i-1}})$

then $z_j = x_j m_j \leftrightarrow (z_j^1, z_j^2, \ldots , z_j^{i-1}, z_j^{i+1}, \ldots , z_j^n)$,

where $z_j^k = \frac{1}{x_j^{k+1}} / \frac{j, m_j^{k+1}}{m_j^{k+1}}$, $k = 1, 2, \ldots , n$.

Start the processes $R_j^1, R_j^2, \ldots , R_j^p$

where $R_j^p$ is $\bmod m_j$ comparator which compares $x_k$ and $z_j^k$. If compared, then set $\beta _j^k = 1$;

otherwise $\beta _j^k = 0$, for $k = 1, 2, \ldots , i-1, i+1, \ldots , n$.

Compute a logical variable $\alpha _j^p$ as

$$\alpha _j^p = (\beta _j^1 \beta _j^2 \cdots \beta _j^{i-1} \beta _j^{i+1} \cdots \beta _j^n)$$

**Description of the process $R_1^p$:**

If $x_1 \geq 0$, set $b = 1$ else $b = 0$.

Compute a logical variable $\alpha _1^p$ as

$$\alpha _1^p = (\beta _1^1 \beta _1^2 \cdots \beta _1^{i-1} \beta _1^{i+1} \cdots \beta _1^n b)$$

where $\beta _1^k = k = 1, 2, \ldots , i-1, i+1, \ldots , n$ are obtained in the same manner as obtained in the case of the process $R_p$ above.

**Example**

Consider the SINS of 4 pairwise relatively prime moduli $7, 5, 12$. Let $X \leftrightarrow (1, 1, 1, 0)$ be the number whose sign is to be detected. Choose any modulus, say $m_1 = 7$ and $p = 15$.

Step 1: Start the processes $Q_1$ and $Q_2$.

$Q_1: x_1 = x_1 \leftrightarrow (1, 1, 1)$ with respect to the moduli $5, 3, 2$.

$Q_2: m_1 = m_1 \leftrightarrow (2, 1, 1)$ with respect to the moduli $5, 3, 2$.

Step 2: Start the processes $R_0, R_1, \ldots , R_{15}$.

**Process $R_0$:**

Start the process $R_0^2, R_0^3, R_0^4, R_0^5, R_0^0$

$R_0^2$: Mod $m_2$ comparator which compares $x_2$ and $\frac{x_2}{m_2}$, i.e.,

compare $-1$ and $-1$, in mod 5 comparator. Since they compare, set $\beta _0^2 = 1$.

Similarly the processes $R_0^3$ and $R_0^4$ yield, $\beta _0^3 = 1$ and $\beta _0^4 = 1$.

Now $x_1 = x_1 \geq 0$, set $a = 1$.

Compute a logical variable $\alpha ^0$ as

$$\alpha ^0 = (\beta ^0_0 \beta ^0_1 \beta ^0_2 \beta ^0_3 \beta ^0_4 a) = 1.$$
Process $R_1$: $z_1 = x_1 \cdot m_1 \leftrightarrow (z_2^1, z_3^1, z_4^1) = (2, -1, 0)$

Start the processes $R_2, R_3, R_4$.

$R_1$: Compare $x_2$ and $z_2^1$ in mod 5 comparator. They do not compare, so set $\beta_1 = 0$.

The processes $R_2, R_3$ and $R_4$ give,

$$\beta_2 = 0 \text{ and } \beta_3 = 1.$$ 

Compute $\alpha = (\begin{array}{c} 1 \\ \beta_2 \\ \beta_3 \\ \beta_1 \end{array}) = 1$.

Similarly the processes $R_2, R_3, \ldots, R_5$ are executed simultaneously to yield

$$\alpha_2 = \alpha_3 = \cdots = \alpha_6 = 1 \text{ and } \alpha_7 = 0.$$

Process $R_7$: Since $x_1 \geq 0$, set $b = 0$.

Compute a logical variable $\alpha_{15}$ as

$$\alpha_{15} = (\begin{array}{c} 1 \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \\ \beta_{15} \end{array}) = 1.$$

Step 3: Find $s' = (0, 1, 1, \ldots, 1) = 1$.

Step 4: Since $s'_{15}$ is even, form

$$s = s'_{15} \cdot \alpha_{15} = 1.$$

Step 5: Since $s = 1$, the number $X$ is negative. Stop.

**HARDWARE REPRESENTATION**

A hardware representation of this algorithm to detect the sign of $X = (x_1, x_2, x_3)$, is shown in Fig. 1, for the case when the SRNS consists of the moduli 5, 3, 2. The symmetric residue corresponding to any modulus will be represented in binary representation. It requires one extra bit for sign viz., symmetric residues corresponding to the moduli 5, 3, 2 are represented in 4, 3 and 2 bits respectively.

In Fig. 1 circled 's' denotes the modulo addition. The comparators have output 1 only for identical inputs. We use semiconductor gates in our design. The 'e' denotes ' wired OR' function.

**Remark 2:** The proposed scheme for sign detection can be generalized to one in which the selected modulus is replaced by a selected subset of the set $(m_1, m_2, \ldots, m_n)$ of the moduli. $m_1$ is then the product of the remaining moduli.

**Remark 3:** In the above implementation, we have considered the case when $s_{15}$ is even. But if $s_{15}$ is odd then in the proposed implementation, disconnect the output of the last comparator.

**TIME AND HARDWARE COMPLEXITY REQUIRED FOR THE PROCESS OF SIGN DETECTION**

In this section, we estimate the time required to detect the sign of a number in its symmetric residue representation and the hardware complexity.

**Time Required to Compute the Sign:**

Let $T_S =$ time required to detect the sign of a residue number.

$$T_C = \text{time required to convert } x_i \text{ to } (x_i / m_1, x_i / m_2, \ldots, x_i / m_n);$$

$$x_i / m_{i+1}, \ldots, x_i / m_n).$$

Fig. 1 Hardware representation of the scheme for sign detection
t_a = time required for one residue addition,
\[ t^* = \text{time required by comparator to compare,} \]
\[ /x_i^k m_k \text{ with } x_k^k, \ k = 1, 2, ..., i-1, \]
\[ i+1, ..., n, \]
\[ t_s = \text{time required to compute } S \text{ with} \]
\[ \text{the help of } \alpha^k. \]

Then
\[ T_S = t_c + t_a + t^* + \Delta + t_s, \text{ where } \Delta \text{ is a single gate delay.} \]

Assume that combinational logic is used to convert \( x_i^k \) to \( /x_i^k m_i^k, ..., /x_i^k m_i^{i-1}, /x_i^k m_i^{i+1}, ..., /x_i^k m_i^n \) is used, then
\[ T_S \approx 8 \Delta + t_s, \text{ where } t_a = 3 \Delta \]

**Hardware Complexity**

Let \( L_k \) be the number of gates required to convert \( x_i^k \) to \( /x_i^k m_k \).

\[ A_k = \text{the number of gates required for mod } m_k \text{ adder,} \]
\[ C_k = \text{the number of gates required for mod } m_k \text{ comparator,} \]
\[ D_k = \text{the number of gates required for comparator, comparing} \]
\[ \text{whether } x_i^k \geq 0 \text{ or } x_i^k \leq 0, \]
\[ \text{for } k = 1, 2, ..., i-1, i+1, ..., n. \]

For the implementation suggested here, the number of mod \( m_k \) adders used are \( (m_k - 1) \), the number of mod \( m_k \) comparators used are \( m_k \),

\[ \text{for } k = 1, 2, ..., i-1, i+1, ..., n. \]

The number of NAND gates used are
\[ \left\lfloor \frac{m_i}{2} \right\rfloor + 1. \] It is assumed that the NAND gate has fan-in of \( n \). Here, the sign function \( S \) is directly obtained by performing 'Wired OR' function. But, if

the value of \( \left\lfloor \frac{m_i}{2} \right\rfloor \) is very large, then we have to introduce the logical gates. Let \( T^* \) be the number of gates required in the last part of our design to get \( S \), then the total complexity estimated (in terms of gates) is

\[ \sum_{k=1}^{n} \left\lfloor \frac{L_k + A_k (m_k-1) + C_k m_k}{2} \right\rfloor + 2 D_k + \left\lfloor \frac{m_i}{2} \right\rfloor + T^*. \]

Here we see that the hardware complexity increases with the increase in the range of the system. So, this approach is suitable for MSI/LSI realization. But this scheme saves execution time at the cost of hardware complexity.

**CONCLUSION**

The scheme for sign detection of a number in the SAMS presented in this paper completely does away with the time consuming process of the 5MRC. The hardware implementation of the algorithm based on the proposed approach is also suggested. As we have shown that the hardware complexity increases with the increase in the range of the system, it is suitable only for MSI/LSI realization.

**REFERENCES**


**Appendix**

**Proof of the theorem**

\[ X \in \left[ \left\lfloor -\frac{m_i}{2} \right\rfloor + 1, \left\lfloor \frac{m_i}{2} \right\rfloor \right], \]
\[ \text{can be written as} \]
\[ X = j m_i^* / /x_i^k m_i, \text{ for some } j, \]
\[ \left\lfloor -\frac{m_i}{2} \right\rfloor < j \leq \left\lfloor \frac{m_i}{2} \right\rfloor, \]

150
or $X = j m_i + x_i,$ \hfill (2)

\[ X_m = j m_i + x_i / m_k \]

for $k = 1, 2, \ldots, i-1, i+1, \ldots, n,$
or $X_k = j m_i + x_i / m_k$.

It is obvious that $X$ is non-negative in the range $[0, \left[ \frac{m_i}{2} \right]]$. There are two cases.

Case (a) : If $m_i$ is odd, then for non-negative $X$, $j$ is bounded between $1$ and $\left[ \frac{m_i}{2} \right]$,
since for $j = 1$

\[ 0 < X_{x_i} = m_i + x_i < \left[ \frac{m_i}{2} \right] \]

and for $j = \left[ \frac{m_i}{2} \right]$,

\[ 0 < X_{x_i} = \left[ \frac{m_i}{2} \right] - 1 m_i + x_i \leq \left[ \frac{m_i}{2} \right] \]

If $j = 0$ then equation $(2)$ becomes

\[ X = x_i \]

For non-negative $X$,

\[ X \geq 0 = x_i \geq 0. \]

In conclusion, we say that $X$ is non-negative iff

\[ \left[ \frac{m_i}{2} \right] \]

\[ \left\{ x_k = / j m_i + x_i / m_k \right\} \cup \left\{ x_k = / x_i / m_k \right\} \]

\[ j = 1 \]

\[ \land x_i \geq 0 \] is true.

Case (b) : If $m_i$ is even, then $\left[ \frac{m_i}{2} \right] = \frac{m_i}{2}$

For $j = \left[ \frac{m_i}{2} \right]$, we get

\[ X = \left[ \frac{m_i}{2} \right] m_i + x_i, \] by using $(2)$,

Or

\[ X = \frac{m_i}{2} + \frac{m_i}{2} + x_i = \left[ \frac{m_i}{2} \right] + x_i \]

since $M$ is even.

For non-negative $X$, $X \leq \left[ \frac{m_i}{2} \right]$.

Therefore from above equation $x_i \leq 0$.

For $j = 0$, $x_i \geq 0$ (shown in case (a)).