New designs for a sign detector and a residue to binary convertor

A. Hiasat

Indexing terms: Residue arithmetic, Residue to binary convertor, Sign detector, Addition, Binary numbers

Abstract: In the paper, two algorithms have been developed. The first one has been exploited in implementing a residue to binary convertor (R/B) for the moduli set $\{2^k - 1, 2^k, 2^k + 1\}$. The new convertor is memoryless, which implies that its upper bound is not limited by a memory size. Furthermore, this convertor represents a new significant reduction in both the hardware requirements and conversion time. A full conversion cycle consists of three consecutive additions each of $(k + 1)$ bits. The same implementation can be modified slightly to implement a sign detector for the same moduli set. Another algorithm has been developed based on the mixed radix conversion technique. This algorithm was used to implement a new sign detector. This new detector is very efficient in the sense that it requires only two addition cycles each of $(k + 1)$ bits. A further reduction in execution time is possible if pipelining is used. All the circuits presented can be implemented using VLSI technology, which gives rise to a reduction in the integrated circuit area.

1 Introduction

Owing to the fact that addition, subtraction and multiplication are very fast operations, residue arithmetic has received considerable attention in the last decade. Unlike weighted systems, where an overflow or sign detection can be easily done by comparing the data patterns, residue number system techniques seem to be very complex and time consuming. Research on special classes of moduli sets has been undertaken [1–6]. Such moduli sets are very efficient in their binary representation. Furthermore, they are very attractive in the sense that they can be easily scaled using the available logic technology.

In this paper, moduli set $\{2^k - 1, 2^k, 2^k + 1\}$ has been studied carefully. The problems mentioned above concerning sign and overflow detection are no longer real ones. A very efficient sign detector has been devised. This new sign detector can also be used in constructing an overflow detector for any addition or subtraction operation within the same moduli set.

Furthermore, in order to utilise digital or analogue signals in a residue arithmetic system, they must be first converted to a residue number system, processed, and then converted back. A new residue to binary convertor has been presented here to facilitate such an interfacing with other digital or analogue systems.

The fact that only binary adders are being used in the suggested implementations presented here gives rise to very fast operations. Moreover, using binary adders only enables very wide dynamic ranges to be implemented without any technical limitations being imposed.

2 Residue to binary convertor

Consider the three moduli set $\{m_1, m_2, m_3\}$, where

$$m_1 = 2^k - 1$$

$$m_2 = 2^k$$

$$m_3 = 2^k + 1$$

Any integer X given by the residue representation $\{r_1, r_2, r_3\}$ can be uniquely represented if it belongs to the interval $[0, M - 1]$ where

$$r_1 = |X|_{2^k - 1}$$

$$r_2 = |X|_{2^k}$$

$$r_3 = |X|_{2^k - 1}$$

$$M = m_1 m_2 m_3$$

The binary representation of X can be expressed in $3k$ bits as

$$X = \sum_{i=0}^{3k-1} 2^i b_i \tag{1}$$

X can be rewritten in the form

$$X = \sum_{i=0}^{k-1} 2^i b_i + A \cdot 2^k + B \cdot 2^{2k} \tag{2}$$

where

$$A = \sum_{i=k}^{2k-1} 2^{i-k} b_1 \tag{3}$$

$$B = \sum_{i=2k}^{3k-1} 2^{i-2k} b_i \tag{4}$$

Now applying mod 2^k operator to both sides of eqn. 2

$$|X|_{2^k} = \left| \sum_{i=0}^{k-1} 2^i b_i + A \cdot 2^k + B \cdot 2^{2k} \right|_{2^k} \tag{5}$$

recalling that

$$|2^k|_{2^k} = 0 \quad |2^{2k}|_{2^k} = 0$$

This will reduce eqn. 5 to

$$r_2 = \sum_{i=0}^{k-1} 2^i b_i \tag{6}$$

Paper 9388G (C2, E10), first received 18th November 1991 and in revised form 27th October 1992

The author is at PO Box 280, Satt, Jordan

This implies that the least significant K bits of X are really bits of r_2.

Substituting eqn. 6 into eqn. 2 and then taking mod $(2^k - 1)$ yields

$$r_1 = |r_2 + A \cdot |2^k|_{2^k-1} + B \cdot |2^{2k}|_{2^k-1}|_{2^k-1} \quad (7)$$

recalling that

$$|2^k|_{2^k-1} = 1 \quad |2^{2k}|_{2^k-1} = 1$$

eqn. 7 can be rewritten [6]:

$$r_1 = |r_2 + A + B|_{2^k-1} \quad (8)$$

Similarly, applying mod $(2^k + 1)$ operator to eqn. 2 yields

$$r_3 = |r_2 + A \cdot |2^k|_{2^k+1} + B \cdot |2^{2k}|_{2^k+1}|_{2^k+1} \quad (9)$$

recalling that

$$|2^k|_{2^k+1} = |-1|_{2^k+1} \quad |2^{2k}|_{2^k+1} = 1$$

Eqn. 9 can be expressed as

$$r_3 = |r_2 - A + B|_{2^k+1} \quad (10)$$

Reconsider eqn. 8. It is obvious that the quantity $(r_2 + A + B)$ is limited by the following range (recall that $X < M$. This implies that A and B can not both be $(2^k - 1)$ at the same time):

$$0 \leqslant r_2 + A + B < 3(2^k - 1) \quad \text{Rel. (1)}$$

Hence eqn. 8 can be written as

$$r_1 = r_2 + A + B - n(2^k - 1) \quad (11)$$

where $n = 0$, 1, or 2.

The same procedure can be applied to eqn. 10, where

$$-(2^k - 1) \leqslant r_2 - A + B \leqslant 2(2^k - 1) \quad \text{Rel. (2)}$$

hence eqn. 10 can be written as

$$r_3 = r_2 - A + B - l(2^k + 1) \quad (12)$$

where $l = -1$, 0, or 1.

Adding and then rearranging eqns. 11 and 12, to determine B

$$2B = (l + n)2^k + (l - n) + r_1 + r_3 - 2r_2 \quad (13)$$

Using the same two eqns. 11 and 12,

$$2A = -(l - n)2^k - (l + n) + r_1 - r_3 \quad (14)$$

Now let

$$c = l + n$$

$$d = l - n$$

Then eqns. 13 and 14 can be expressed as

$$2A = r_1 - r_3 - d_2^k - c \quad (15)$$

$$2B = r_1 + r_3 - 2r_2 + c_2^k + d \quad (16)$$

It is obvious that the value of X (our objective) can be determined if A and B are known. To determine those values, we proceed as follows: all possible combinations of c and d are shown in Table 1.

Recalling that integers $2A$, $2B$ form the left-hand side (LHS) of eqns. 15 and 16, respectively, the following conditions can be stated:

(i) $0 \leqslant 2A \leqslant 2(2^k - 1)$

$0 \leqslant 2B \leqslant 2(2^k - 1)$

(ii) $2A$, $2B$ are both even integers

Using eqn. 16 it is clear that, whenever the sum $(r_1 + r_3)$ is even, d has to be even, and whenever the same sum is odd, d has to be odd.

Table 1

Case no.	1	n	c	d
1	−1	0	−1	−1
2	−1	1	0	−2
3	−1	2	1	−3
4	0	0	0	0
5	0	1	1	−1
6	0	2	2	−2
7	1	0	1	1
8	1	1	2	0
9	1	2	3	−1

Similarly, using eqn. 15, it is obvious that, whenever $(r_1 - r_3)$ is odd, c has to be odd, and whenever the result is even, c has to be the same.

Hence it can be concluded that, whenever the least significant bits (LSB) of r_1 and r_3 are different, c and d have to be odd. However, if LSBs of r_1, r_3 are the same; c and d have to be even.

Table 2 summarises all the even combinations of c and d along with the permissible range associated with each

Table 2

(c, d)	$W = r_1 + r_3 - 2r_2$	$Z = r_1 - r_3$
(0, −2)	$2 \leqslant W \leqslant 2(2^k - 1)$	$-2^k \leqslant Z \leqslant -2$
(0, 0)	$0 \leqslant W \leqslant 2(2^k - 1)$	$0 \leqslant Z \leqslant 2^k - 2$
(2, −2)	$-2(2^k - 1) \leqslant W \leqslant 0$	$-2^k \leqslant Z \leqslant 0$
(2, 0)	$-2(2^k - 1) \leqslant W \leqslant -2$	$2 \leqslant Z \leqslant 2^k - 2$

case: (for proof see Appendix 7). Figs. 1 and 2 indicate graphically the ranges given in Table 2. Fig. 1 shows the possible ranges of the variable Z. However, Fig. 2 shows the same ranges concerning the variable W.

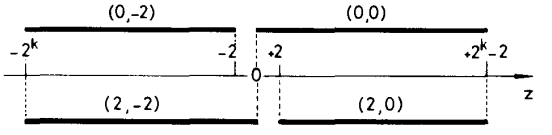

Fig. 1 *Ranges of Z for even values of (c, d)*

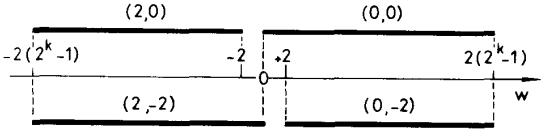

Fig. 2 *Ranges of W for even values of (c, d)*

Table 3 summarises all the valid combinations for odd values of c and d (see Appendix 7 for proof).

The cases shown in Table 3 are graphically represented in Fig. 3. It shows each permissible range, along with the corresponding combination of (c, d).

Table 3

(c, d)	$W = r_1 + r_3 - 2r_2$
(−1, −1)	$2^k + 1 \leqslant W \leqslant 2(2^k - 1)$
(1, −1)	$-(2^k - 1) \leqslant W \leqslant 2^k - 1$
(3, −1)	$-2(2^k - 1) \leqslant W \leqslant -(2^k + 1)$

Referring to Fig. 1, it is obvious that, whenever $Z > 0$, $d = 0$, and whenever $Z < 0$, $d = -2$. Similarly, Fig. 2 shows that, whenever $W > 0$, $c = 0$; however, if $W < 0$, $c = 2$.

Concerning the case when $Z = 0$ AND/OR $W = 0$, consider the following:

(i) $Z = 0$, $W \neq 0$: From Fig. 2, if $W > 0$, $c = 0$. Referring to Fig. 1, this implies that $d = 0$. This is so because the combination (0, 0) satisfies the requirement that

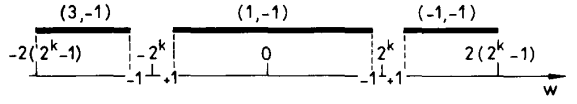

Fig. 3 *Ranges of W for odd values of (c, d)*

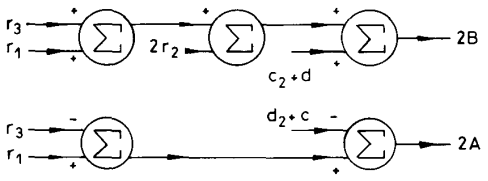

Fig. 4 *New memoryless residue to binary convertor*

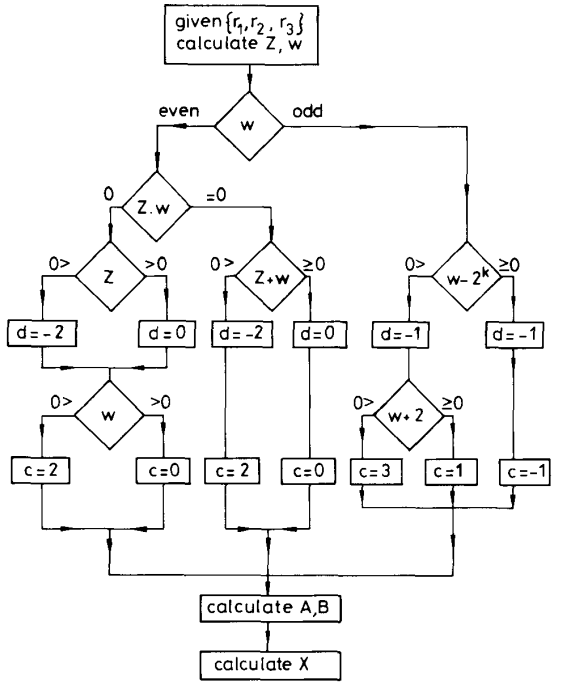

Fig. 5 *Flow chart for R/B conversion algorithm*

$Z = 0$. On the other hand, the combination (2, −2) does not satisfy the requirement that $c = 0$ although it satisfies the requirement $Z = 0$. Similarly, if $W < 0$, the valid combination is (2, −2).

(ii) $w = 0$, $z \neq 0$: From Fig. 1, if $Z > 0$, $d = 0$. Referring to Fig. 2, this implies that $c = 0$. This is so because the combination (0, 0) satisfies the requirement that $W = 0$. On the other hand, the combination (2, −2) does not satisfy the requirement that $d = 0$ although it satisfies the requirement $W = 0$. Similarly, if $Z < 0$ the valid combination is (2, −2).

(iii) $Z = 0$, $W = 0$: It seems that one of the following combinations is valid; either (0, 0) or (2, −2). Referring to eqns. 15 and 16, one can conclude that the first combination leads to $A = 0$, $B = 0$; while the other combination, that is (2, −2), leads to $A = 2^k - 1$, and $B = 2^k - 1$. If the last values for A and B are substituted in eqn. 2, the resulting value of X would be outside the range $[0, M - 1]$.

Hence, whenever $Z = 0$, $W = 0$, $c = 0$ and $d = 0$.

Now, to determine the value of any integer X which is expressed in residue representation as $\{r_1, r_2, r_3\}$, ranges in which W and Z lie should be recognised. This can be done as follows:

(i) Z: consider the following three possibilities (using 2s complement notation to express binary arithmetic)

(*a*) $Z > 0$, when the overflow bit resulting from subtracting r_3 from r_1 is 1

(*b*) $Z < 0$, when overflow bit of Z is 0

(*c*) $Z = 0$, when the result of ORing all bits of Z and complement of its overflow bit is 0

(ii) W: consider the following:

(*a*) $0 \leqslant W < 2^k$; when the overflow bit of W is 1 AND the MSB of W is 0.

(*b*) $2^k \leqslant W < 2(2^k - 1)$; when the overflow bit of W is 1 and MSB of W is 1.

(*c*) $-2^k \leqslant W < 0$; when the overflow bit of W is 0 and MSB of W is 1.

(*d*) $-2(2^k - 1) \leqslant W < -2^k$; when the overflow bit of W is 0 and MSB of W is 0.

(*e*) $W = 0$; when the result of ORing all bits of W and its overflow bit is 0.

(*f*) W is odd if the LSB of W is 1, otherwise it is even.

Fig. 4 shows the suggested hardware implementation for this new algorithm, which converts the residue digits into their binary equivalent. The output is obtained by dropping the LSB in each of outputs $2A$ and $2B$, then sticking bits of A, B and r_2 together.

The implementation of the residue to binary (R/B) convertor does not show the combinational circuit which produces the two outputs $c_2^k + d$ and $d_2^k + c$. Such a circuit requires few logic gates. The operation of this convertor is self-explanatory. It is obvious that it requires five binary adders, three of which are $(k + 1)$ bits while the others are of $(k + 2)$ bits. A full conversion cycle consists of three consecutive additions.

The algorithm presented here can also be used in constructing a sign detector for the same moduli set.

Since X belongs to the range $[0, M - 1]$, where $M = 2^{3k} - 2^k$. We will consider X positive if it belongs to the range $[0, 2^{3k-1} - 2^{k-1}]$ and negative in the range $[2^{3k-1} - 2^{k-1}, M - 1]$. But since the MSB of $2B$ Fig. 4 is 1 in the range $[2^{3k-1}, M]$, and it is 0 elsewhere, this bit can serve as a tool in detecting the sign bit for all values of X, except when X belongs to the interval

$$[2^{3k-1} - 2^{k-1}, 2^{3k-1} - 1]$$

It can be easily verified that this interval is uniquely characterised by $Z = 0$ AND $W = -2^k$. Hence a logic gate can detect this value of W and can check for $Z = 0$. The bit resulting from ANDing those conditions is NORed with MSB of $2B$. The result is the SIGN BIT needed. It is obvious that the hardware given in Fig. 4, needs a few more logic gates to incorporate sign detection function.

However, if a sign detector is to be implemented without having a residue to binary convertor, all the hardware of Fig. 4 is needed except the adder which outputs $2A$.

LSI/VLSI technology can be used to construct the above suggested circuit. A thorough comparison of this convertor with other similar reported convertors [1–5] proves that it requires less hardware and conversion time with, or without, pipelining (Fig. 5).

3 Sign detector

Consider any integer X that belongs to the range $[0, M-1]$, using the mixed radix-conversion (MRC) technique [7]; then X can be expressed as

$$X = (2^{2k} - 1)a_3 + (2^k + 1)a_2 + a_1 \quad (30)$$

where: a_1, a_2, a_3 are the MR coefficient associated with the moduli $(2^k + 1)$, $(2^k - 1)$ and 2^k, respectively.

Thus

$$0 \leqslant a_1 < 2^k + 1$$

$$0 \leqslant a_2 < 2^k - 1$$

$$0 \leqslant a_3 < 2^k$$

To determine whether X is positive, or negative it is merely sufficient to determine if $a_3 < 2^{k-1}$ or not, simply because whenever $a_3 = 2^{k-1}$, then eqn. 30 can be written as

$$X = M/2 + (2^k + 1)a_2 + a_1 \quad (31)$$

Hence, if $a_3 < 2^{k-1}$, X is positive; otherwise it is negative. In the next analysis, the value of a_3 will be determined.

Applying mod $(2^k + 1)$ operator to eqn. 30 yields

$$r_3 = a_1 \quad (32)$$

Similarly, applying mod $(2^k - 1)$ operator to eqn. 30 yields

$$r_1 = |a_1 + 2 \cdot a_2|_{2^k-1}$$

or

$$r_1 = |r_3 + 2 \cdot a_2|_{2^k-1} \quad (33)$$

Taking mod 2^k for eqn. 30, then

$$r_2 = |a_1 + a_2 - a_3|_{2^k}$$

or

$$r_2 = |r_3 + a_2 - a_3|_{2^k} \quad (34)$$

Since $0 \leqslant a_2 \leqslant 2^k - 2$, eqn. 33 can be rewritten as

$$r_1 = r_3 + 2 \cdot a_2 - n(2^k - 1) \quad (35)$$

where $n = 0$, 1, or 2 depending on the value of $(r_3 + 2 \cdot a_2)$.

Similarly, eqn. 34 can be rewritten as

$$r_2 = r_3 + a_2 - a_3 - l \cdot 2^k \quad (36)$$

where $l = -1, 0$, or 1.

Multiplying eqn. 36 by 2, then substituting the value of $2 \cdot a_2$ from eqn. 35 into this multiplied equation, and rearranging:

$$2 \cdot a_3 = r_1 + r_3 - 2r_2 + n(2^k - 1) - 2 \cdot l \cdot 2^k$$

or

$$2 \cdot a_3 = W + f2^k n \quad (37)$$

where

$$W = r_1 + r_3 - 2r_2$$

$$f = n - 2, l$$

Since any integer X will be considered positive only if

$$0 \leqslant a_3 < 2^{k-1}$$

or

$$0 \leqslant 2 \cdot a_3 < 2^k$$

Using eqn. 37, then the condition for X to be positive is

$$0 \leqslant W + f2^k - n < 2^k \quad \text{Rel. (18)}$$

Table 4 shows all possible combinations of l and n. Since W might be any integer within the range

$$-2(2^k - 1) \leqslant W \leqslant 2(2^k - 1)$$

Table 4

l	n	f
−1	0	2
−1	1	3
−1	2	4
0	0	0
0	1	1
0	2	2
1	0	−2
1	1	−1
1	2	0

a thorough look at Table 4 shows that the only possible values of f which satisfy rel. 18 are $f = -1, 0, 1$ or 2.

To determine the value of f and the corresponding n, the following two cases are discussed:

(i) When W lies in either one of the following ranges

1st range: $2 - 2^{k+1} \leqslant W < -2^k$

2nd range: $0 \leqslant W < 2^k$

The possible values of f that satisfy rel. 18 are 2 for the 1st range, and 0 for the 2nd range. But, since eqn. 35 requires that $2 \cdot a_2$ is a positive, even integer, this implies that $n = 0$ when $r_1 \geqslant r_3$ and $n = -2$ otherwise. Thus we define a new variable W' to be

$$W' = \begin{cases} W & r_1 \geqslant r_3 \\ W - 2 & \text{otherwise} \end{cases}$$

According to eqn. 36, the quantity $(W + f2^k - n)$ has to be even. This implies that only even values of W' in the above given ranges satisfy rel. 18.

When using the 2s complement notation to express binary numbers, any value of W' with LSB = 0 AND MSB = 0 satisfies rel. 18.

(ii) When W lies in one of the following ranges:

3rd range: $-2^k \leqslant W < 0$

4th range: $2^k \leqslant W < 2^{k+1}$

Once more, the only values of f that satisfy rel. 18 are 1 for the 3rd range, and -1 for the 4th range. According to Table 4, the associated n for both ranges is $n = -1$.

Recalling that eqn. 36 requires that $(W + f2^k - n)$ has to be even, this implies that W has to be odd, simply because n is odd (actually $n = -1$).

For odd values of W, we will consider $W' = W$ rather than $W' = W - 1$. This will be done for the sake of simplicity, since it will not affect the range of W (i.e. W and $W - 1$ lie in the same operating range). However, this cannot be applied when W is even, because subtracting 2 from an even integer might change its operating range. For example, if $W = -2^k$ it belongs to the 3rd range defined above, but if $n = -2$, W' lies in the 1st range. Such a change would affect the sign determination.

Similarly, using the 2s complement notation, it can be stated that, whenever the LSB of W' is 1 AND the MSB is 1 too, such values of W' satisfy rel. 18. Fig. 6 summarises the above presented analysis.

Depending on previous analysis, the following algorithm can be stated to detect the sign of any integer X represented by $\{r_1, r_2, r_3\}$:

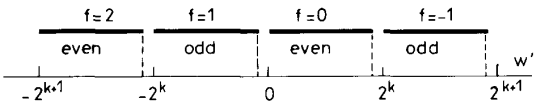

Fig. 6 *Different ranges of W'*

3.1 Algorithm for sign detection

(i) Given the residue representation of X; calculate W

(ii) Determine whether W is odd or even, and whether $r_1 \geqslant r_3$

For odd W $\quad W' = W$

For even W $\quad W' = \begin{cases} W & r_1 \geqslant r_3 \\ W-2 & \text{otherwise} \end{cases}$

(iii) If W' is EVEN, X is POSITIVE if W' lies in the 1st or 2nd range. If W' is ODD, then X is POSITIVE if it lies in the 3rd or 4th range.

Fig. 7 shows a flowchart which serves to simplify the algorithm.

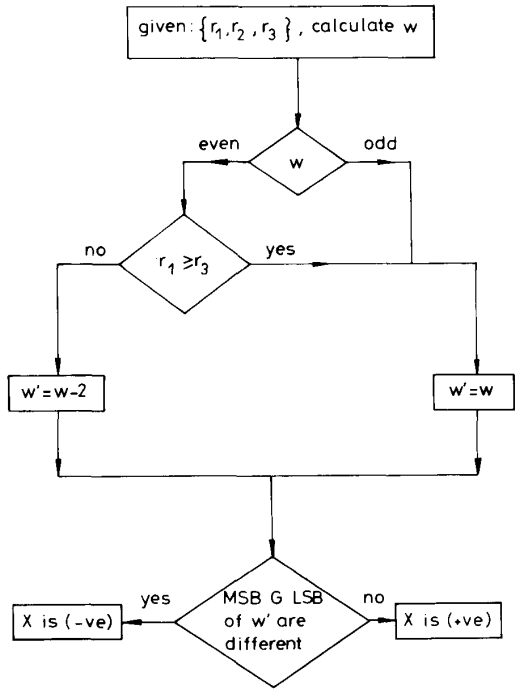

Fig. 7 *Flow chart for sign detection algorithm*

3.2 Hardware implementation

The suggested hardware for the new sign detector is shown in Fig. 8. This detector assumes using 2's complement subtractors; that is $X - Y = (X + 1'$ sC of $Y + 1)$. The detector operates as follows:

(i) Adder 1 adds r_1 to r_3.

(ii) Subtractor 1 subtracts r_3 from r_1. This subtractor is merely used to determine the sign bit of $r_1 - r_3$.

(iii) Subtractor 2 subtracts $2r_2$ from the output of Adder 1. To implement the case when W is even and $r_1 < r_3$ that is $n = -2$, the LSB of subtractor 1 is logically ORed with its sign bit. The result of this OR gate is applied to the input carry of subtractor 2. However, the result of applying the same two bits to a NOR gate is added to $2 \cdot r_2$ (i.e. it is replacing the LSB of $2 \cdot r_2$ which is zero). The result of subtractor 2 is $(W - 2)$ if W is even and $r_1 < r_3$. Otherwise, the result is W.

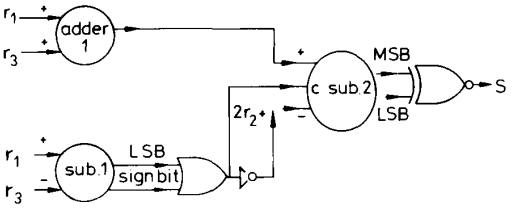

Fig. 8 *New sign detector*

(iv) The output of this detector which is (denoted as S) is the result of applying the MSB and LSB of subtractor 2 to an EX-NOR gate. The residue number is considered positive if S is 1 and negative if S is 0.

It is obvious that the sign of a residue encoded integer can be determined using two addition cycles, each of $(k + 1)$ bits. The total hardware requirements consist of three $(k + 1)$ bit adders.

4 Overflow detector

An addition/subtraction overflow detector can be constructed using three sign detectors. Two of them are to determine the sign of each addition or subtraction operand, while the third is to determine the sign of their result. Very few logic gates are needed to indicate that an overflow took place (i.e. if the sign of input operands are similar, but different from the sign of their result).

5 Conclusions

Two algorithms have been developed. The first has been used in implement: a residue to binary convertor. This convertor serves as a sign detector. Furthermore, the convertor requires five binary adders of $(k + 1)$ bits. A full conversion cycle consists of three consecutive $(k + 1)$ bit additions. The second algorithm was used in constructing a sign detector. This detector requires only three $(k + 1)$ bit adders. A full sign detection cycle consists of two binary additions each of $(k + 1)$ bits. The whole suggested hardware does not use any memory. This gives rise to a very large dynamic range utilisation.

6 References

1 JENKINS, W.K.: 'Recent advances in residue, number techniques for recursive digital filtering', *IEEE Trans.*, 1979, **ASSP-27**, pp. 19–29

2 TAYLOR, F.J., and RMNARAYANAN, A.S.: 'An efficient residue to decimal converter', *IEEE Trans.*, 1981, **CAS-28**, pp. 1164–1169

3 BERNARDSON, P.: 'Fast memoryless, over 64 bits, residue to binary converter', *IEEE Trans.*, 1985, **CAS-32**, pp. 288–300

4 SWEIDAN, A., and HIASAT, A.: 'New efficient memoryless, residue to binary converter', *IEEE Trans.*, 1988, **CAS-35**, pp. 1441–1444

5 IBRAHIM, K., and SALOUM, S.: 'An efficient residue to binary converter design', *IEEE Trans.*, 1988, **CAS-35**, pp. 1156–1158

6 HIASAT, A.: 'A novel design for a multipurpose residue-to-mixed-radix decoder'. Presented at the 35th Midwest Symposium on *Circuits and Systems*, Washington DC, 9th–12th Aug. 1992

7 SZABO, N.S., and TANAKA, R.I.: 'Residue arithmetic and its applications to computer technology' (McGraw-Hill, New York, 1967)

7 Appendix

Consider the even cases of c and d: Let $Z = r_1 - r_3$, and $W = r_1 + r_3 - 2r_2$,

(i) $c = 0, d = -2$;
applying those values to eqn. 16,

$$2B = W - 2 \tag{17}$$

recalling that

$$0 \leqslant 2B \leqslant 2(2^k - 1) \quad \text{Rel (3)}$$

$$-2(2^k - 1) \leqslant W \leqslant 2(2^k - 1) \quad \text{Rel (4)}$$

The last two relations coincide in accordance to eqn. 17 to determine the permissible range of W for $c = 0$, $d = -2$;

$$2 \leqslant W \leqslant 2(2^k - 1) \quad \text{Rel (5)}$$

Similarly, applying $c = 0$, $d = -2$ to eqn. 15 yields

$$2A = Z + 2 \cdot 2^k \tag{18}$$

recalling that

$$0 \leqslant 2A \leqslant 2(2^k - 1) \quad \text{Rel (6)}$$

but since

$$-2^k \leqslant Z \leqslant 2^k - 2 \quad \text{Rel (7)}$$

then applying the last two relations to eqn. 18 yields

$$-2^k \leqslant Z \leqslant -2 \quad \text{Rel (8)}$$

(ii) $c = d = 0$
Eqn. 16 can be written as

$$2B = W \tag{19}$$

applying rel. 3 and 4 to eqn. 19 yields

$$0 \leqslant W \leqslant 2(2^k - 1) \quad \text{Rel (9)}$$

Similarly, applying $c = d = 0$ to eqn. 15

$$2A = Z \tag{20}$$

applying rel 6 and 7 to eqn. 20

$$0 \leqslant Z \leqslant 2^k - 2 \quad \text{Rel (10)}$$

(iii) $c = 2, d = -2$
Substituting these values into eqn. 16 results in

$$2B = W + 2 \cdot 2^k - 2 \tag{21}$$

Applying rel 3 and 4 to eqn. 21 yields

$$-2(2^k - 1) \leqslant W \leqslant 0 \quad \text{Rel (11)}$$

similarly, eqn. 15 can be written for this combination as:

$$2A = Z + 2 \cdot 2^k - 2 \tag{22}$$

Using rel 6 and 7 the following relation is valid

$$-2^k \leqslant Z \leqslant 0 \quad \text{Rel (12)}$$

(iv) $c = 2, d = 0$
Applying these values to eqn. 16 results in

$$2B = W + 2 \cdot 2^k \tag{23}$$

applying rel 3 and 4 to eqn. 23 yields

$$-2(2^k - 1) \leqslant W \leqslant -2 \quad \text{Rel (13)}$$

For the same values of c and d, eqn. 15 can be written as

$$2A = Z - 2 \tag{24}$$

Hence, using rel 6 and 7,

$$2 \leqslant Z \leqslant 2^k - 2 \quad \text{Rel (14)}$$

Consider the odd cases of c and d:
(i) $c = -1, d = -1$
For those values of c and d eqn. 16 can be written as

$$2B = W - 2^k - 1 \tag{25}$$

Hence the variable W, can be expressed according to rel 3 and 4 by

$$2^k + 1 \leqslant W \leqslant 2(2^k - 1) \quad \text{Rel (15)}$$

(ii) $c = 1, d = -1$
Once more, for this case eqn. 16 can be written as

$$2B = W + 2^k - 1 \tag{26}$$

The corresponding range of W is given according to rel 3 and 4 by

$$-(2^k - 1) \leqslant W \leqslant (2^k - 1) \quad \text{Rel (16)}$$

(iii) $c = 3, d = -1$
Applying these values to eqn. 16,

$$2B = W + 3 \cdot 2^k - 1$$

using rel 3 and 4,

$$-2(2^k - 1) \leqslant W \leqslant -(2^k + 1) \quad \text{Rel (17)}$$

Consider the last two cases:
(iv) $c = 1, d = -3$
Substituting these values into eqn. 15

$$2A = Z + 3 \cdot 2^k - 1$$

According to such a combination, rel 6, and 7 cannot be satisfied. Hence it is an impossible case.
(v) $c = 1, d = 1$
Substituting these values into eqn. 15

$$2A = Z - 2^k - 1$$

According to such a combination, rel 6 and 7 cannot be satisfied. Hence it is an impossible case.

It is noticed that the 1st three odd combinations of c, d have not been applied to eqn. 15. However, it can be easily verified that this will not add any other *new* information.