Overflow Effects In Redundant Binary Number Systems

D. Timmermann, B.J. Hosticka

Fraunhofer Institute of Microelectronic Circuits and Systems
Finkenstr. 61, 4100 Duisburg 1, Germany
Tel.: 49 203 3783 219, FAX 49 203 3783 266

Abstract

This work describes a specific overflow behaviour encountered when working with redundant binary numbers, possibly yielding wrong results. This peculiarity is not encountered in conventional number systems and, to the knowledge of the authors, has not been published in the open literature. To avoid these unnecessary overflows we present simple correction schemes.

Redundant binary number system

The *n*-digit redundant binary number (RBN) *a* is defined by $a = \sum_{i=0}^{n-1} a_i 2^{-i}$ with a_i belonging to the digit set {-1, 0, 1} /1/. The maximum and minimum algebraic values of *a* that can be represented are given by $2\text{-}2^{-n+1}$ and $-2+2^{-n+1}$, respectively. Allowing different (i.e. redundant) representations for the same number any carry propagation over more than two digits during addition can be avoided. Thus significant speed improvements are achieved when carrying out as much subsequent operations as possible within the RBN system.

An optimized addition algorithm for RBNs has been described by Takagi /2/. Consider two RBNs *a* and *b*. Then the sum $s = a + b$ is given by Table I and $s_i = z_i + d_{i+1}$, using an interim carry *d* and an interim sum *z*, both fulfilling the equation $a_i + b_i = 2\ d_i + z_i$. The structure of a redundant binary adder (RBA) implementing this addition algorithm is depicted in Fig. 1

a_i	b_i	a_{i+1} b_{i+1}	d_i	z_i
1	1	don´t care	1	0
1	0	both ≥ 0	1	-1
0	1	else	0	1
$a_i + b_i = 0$		don´t care	0	0
0	-1	both ≥ 0	0	-1
-1	0	else	-1	1
-1	-1	don´t care	-1	0

Table I: Addition rule for RBNs /2/

Overflow

To illustrate the overflow characteristic of RBNs let us consider the addition of a three digit number $a = 0.25_{10} = 1.\bar{1}\bar{1}$ ($n = 3$). Computing a+a using Table I results in:

$$
\begin{array}{lccccc}
 & & 1 & . & \bar{1} & \bar{1} \\
+ & & 1 & . & \bar{1} & \bar{1} \\
\hline
z & & 0 & . & 0 & 0 \\
d & 1 & \bar{1} & . & \bar{1} & \\
\hline
s & 1 \,| & \bar{1} & . & \bar{1} & 0 \; = \; -1.5_{10} \\
\text{overflow} \Rightarrow & | & & & &
\end{array}
$$

Obviously incorrect results are caused by discarding the overflow digit. This effect occurs in spite of the fact that the wordlength is sufficiently large to represent the correct result 0.5_{10}. Therefore, we call it a pseudo-overflow to distinguish it from a true overflow which cannot be corrected. Pseudo-overflows occur mainly in subsequent calculations using RBNs.

An example of a classification of various overflow types (true: not correctable as the wordlength is exceeded, pseudo: correctable, potential: impossible to determine whether an overflow occurs or not) for fraction representation is given in Table II.

Significance 2^1	2^0	Type of overflow
d_0	s_0	
1	-1	pseudo, Case I
1	0	potential, Case III
1	1	true
-1	-1	true
-1	0	potential, Case IV
-1	1	pseudo, Case II
0	X	no overflow

Table II: Overflow characteristic

One countermeasure is to increase the wordlength by one digit each time an overflow could be generated. In general, however, this would be too costly so we propose simple solutions for treatment of pseudo-overflows and potential overflows in the next two sections.

Pseudo-overflow correction

Pseudo-overflows can occur inside the entire number range of the RBN, e.g. for $|a| \leq 2 - 2^{-(n-1)}$. For a pseudo-overflow the expression $d_0 2^1 + s_0 2^0$ equals 2^0 or -2^0 for Cases I and II, respectively. Thus the following easy-to-implement correction scheme for computing an overflow-corrected s_0, denoted s_0', in the most significant digit (MSD) can be used:

d_0	s_0	s_0'
1	-1	1
-1	1	-1
0	X	X

Table III: Pseudo-overflow correction for the MSD

That is, $s_0' = d_0$ if $|d_0| = 1$, else $s_0' = s_0$. This can be easily incorporated into the logic design of a standard RBA (given e.g. in /3/) in the MSD position for either integer or fraction representation. It generates only a marginal additional delay. Fig. 2 depicts a RBA with pseudo-overflow correction using an additional 4:2 multiplexer. It is assumed that {-1,0,1} are coded in two bits (x^s,x^a), (s)ign and (a)mplitude, namely {(1,1),(0,0),(0,1)}. Then $d_0^a = 1$ selects $s_0' = d_0$ and $d_0^a = 0$ outputs $s_0' = s_0$.

Potential overflows

As can be seen from Table II, given a potential overflow situation (Cases III and IV) an inspection of all sum digits would be necessary in the worst case to determine whether a true overflow happens or not. This is shown in more detail in Table IV. Such an inspection would require an additional hardware and sacrifice consequently the speed benefits associated with RBNs, so we propose another solution.

Significance			Type of overflow
2^1	2^0	2^{-1}	
d_0	s_0	s_1	
1	0	-1	pseudo, Case I
1	0	0	potential
1	0	1	true
-1	0	-1	true
-1	0	0	potential
-1	0	1	pseudo, Case II

Table IV: Potential overflow conditions

In order to detect a true overflow from the inspection of only a few MSDs we have to restrict (by modifications on the system/algorithmic level) the maximum absolute value of the RBN. Then a potential-overflow can be corrected by modifying some MSD sum digits in about the same number as for pseudo-overflows. Table V depicts the interaction between the number of MSDs ($s_0..s_{d-1}$) which have to be modified and the numeric range in which the RBN can be lying.

Number range	d
$\|a\| \le 1$	1
$\|a\| \le 1.5$	2
$\|a\| \le 2 - 2^{-(n-1)}$	> 2

Table V: Numeric range restrictions and number of digits to be modified

For example, for $|a| \le 1$ a representation 10.xxxx or -10.xxxx cannot occur and the only overflow is a pseudo-overflow. This can be corrected using the scheme described above. It can be shown by induction that the inequality $|a| \le 2 - 2^{-(d-1)}$ is valid.

If we want to exploit the increased numeric range defined by $|a| \le 1.5$, we have to extend Table III to account for the potential overflow situation as shown in Table VI (s_1' denotes a modified s_1). Obviously for d digits to be modified we have to inspect d_0 and d-1 MSDs.

d_0	s_0	s_0'	s_1'
1	-1	1	s_1
-1	1	-1	s_1
1	0	1	1
-1	0	-1	-1
0	X	s_0	s_1

Table VI: Overflow correction for $|a| \le 1.5$

Discussion

The analysis above yields an important conclusion which states that, in general, the conversion of n-bit 2's-complement arithmetic to n-digit RBN computations poses no problem, as a 2´s-complement number b is defined to be $-1 \le b \le 1-2^{-(n-1)}$. Therefore, the range restriction $|a| \le 1$ is fulfilled and only pseudo-overflows are possible. However, if one wants to exploit the increased numeric range of RBNs the above mentioned MSD modifications have to be implemented. Alternatively, the word length can be increased by one digit in the MSD position.

References:

/1/ Avizienis, A., "Signed-digit number representations for fast parallel arithmetic," IRE Transactions on Electronic Computers, vol. **EC-10**, pp. 389-400, Sept. 1961

/2/ Takagi, N., "Studies on hardware algorithms for arithmetic operations with a redundant binary representation," Ph.D. dissertation, Department of Information Science, Faculty of Engineering, University of Kyoto, August 1987

/3/ Kuninobu, S., et.al., "Design of high speed MOS multiplier and divider using redundant binary representation," Proc. 8th Symp. on Computer Arithmetic, New York, pp. 80-86, 1987

List of Figures

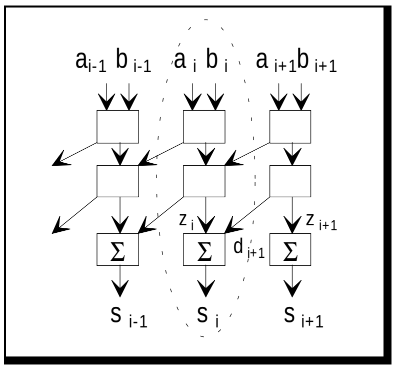

Fig. 1: Structure of a redundant binary adder

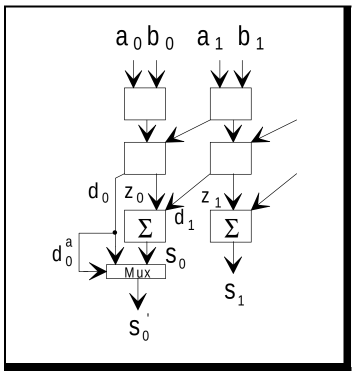

Fig. 2: Redundant binary adder with pseudo-overflow correction